Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

ESPECIFICACIÓN Y DESARROLLO DE UN MOTOR DE REGLAS PARA LA GESTIÓN DE LA CONECTIVIDAD EN ROUTERS MÓVILES CON MÚLTIPLES INTERFACES

TRABAJO FIN DE MÁSTER

Joaquín Navarro Salmerón

2010

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

Máster Universitario en
Ingeniería de Redes y Servicios Telemáticos

TRABAJO FIN DE MÁSTER

ESPECIFICACIÓN Y DESARROLLO DE UN MOTOR DE REGLAS PARA LA GESTIÓN DE LA CONECTIVIDAD EN ROUTERS MÓVILES CON MÚLTIPLES INTERFACES

Autor
Joaquín Navarro Salmerón

Director
Manuel Álvarez-Campana Fernández-Corredor

Departamento de Ingeniería de Sistemas Telemáticos

2010

Resumen

En los últimos años, las necesidades de los sectores profesionales en materia de comunicaciones móviles han ido creciendo, demandando cada vez mayores garantías de conectividad, ancho de banda y calidad de servicio.

En general, en este tipo de escenarios de comunicaciones móviles, los dispositivos involucrados tienen a su alcance distintos tipos de redes de acceso, por ejemplo, redes celulares, redes WiMAX, redes MESH, etc. Además, suele ser común el contar con un nodo (un router) que haga de *gateway* entre estas redes y los dispositivos de cliente.

Este router, un router móvil, aparte de ofrecer conectividad a todos los miembros de su subred actuando como puerta de enlace, gestiona los mecanismos de movilidad necesarios para ocultar a la subred el hecho de que el router vaya cambiando de punto de acceso a la red. Esta tarea se consigue gracias a la utilización del protocolo *Network Mobility (NEMO) Basic Support*.

En este contexto, se pone de manifiesto la necesidad de incluir un elemento capaz de interactuar con este router móvil, en base a criterios de alto nivel, que permitan definir escenarios complejos de comunicaciones, basados por ejemplo en criterios geográficos, administrativos o económicos.

En este trabajo se desarrolla y se prueba un gestor de reglas basadas en texto XML que permita llevar a cabo tareas de monitorización y control sobre un router móvil, exponiendo una interfaz basada en servicios web mediante la cual las aplicaciones clientes interactuarán con el módulo gestor.

En esta línea, se ha desarrollado una biblioteca de cliente que permite el cómodo desarrollo de aplicaciones de control, ocultando los detalles de comunicación basados en *web services*, centrando así el esfuerzo de los desarrolladores en la lógica de la aplicación como tal. Además a modo de prueba de concepto, se desarrollan un conjunto de aplicaciones de control con el objetivo de permitir a los usuarios del router móvil, gestionarlo de una forma fácil y rápida a través del Gestor de Reglas desarrollado en este trabajo.

En el desarrollo del Gestor de Reglas se ha hecho hincapié en conseguir que dicha herramienta sea lo más genérica posible, con el objetivo de poder aplicarla a otros ámbitos en los que sea necesaria una aplicación de monitorización y control.

Todo esto es posible gracias a la tecnología empleada en el desarrollo de estos elementos, basada en Java, utilizando *frameworks* para la creación y utilización de servicios web (Apache CXF), para el análisis y el procesamiento de texto XML (Apache Xerces-J), y para la gestión de aplicaciones web (HTTPServlets y Java Server Pages, JSP).

Abstract

In recent years, the needs of the professional mobile communications have been growing increasingly, demanding better assurance of connectivity, bandwidth and quality of service.

In general, in this kind of mobile communication scenarios, the devices involved have different types of networks access available, for example, cellular networks, WiMAX networks, mesh networks, etc. In addition, it is often common to have a node (a router) that acts as gateway between these networks and client devices.

This router, a mobile router, provides connectivity to all subnet members acting as a gateway, and it also handles mobility mechanisms in order to hide to the mobile nodes the fact that the mobile router is changing the attachment point to the Internet. This goal is achieved by using the protocol Network Mobility (NEMO) Basic Support.

This context highlights the need for an element capable of interacting with the mobile router, using high level criteria, to define complex communication scenarios, based on geographical, administrative or economic criteria, for example.

In this work we developed and tested a rule engine that can manage complex scenarios described by using XML text. This rule engine allows performing monitoring and controlling task of a mobile router. It also exposes a web services-based interface using by client applications to communicate with the rule engine.

We also have developed a client library that allows an easy development of control applications, hiding the details of web services-based communications, thus focusing the efforts of developers in the logic of the application itself. In addition as a proof of concept, we have developed a set of control applications with the aim of allowing users of mobile router to manage it in an easy and fast way.

One of the main requisites of the rule engine developed on this work is making it as generic as possible, with the aim of using it in other areas in which is necessary a monitoring and control application.

All this is possible thanks to the technology used in developing these elements, based on Java, using frameworks for creating and using web services (Apache CXF) for analysis and processing XML text (Apache Xerces-J), and for managing Web applications (HTTPServlets and Java Server Pages, JSP).

Índice general

Índice de figuras

Siglas

3GPP	*3rd Generation Partnership Project*
ABC	*Always Best Connected*
AJAX	*Asynchronous JavaScript And XML*
AODV	*Ad-hoc On Demand Distance Vector*
AXP	*Application Extension Platform*
BATMAN	*Better Approach To Mobile Adhoc Networking*
CN	*Core Network*
CN	*Correspondent Node*
CoA	*Care-of Address*
CSS	*Cascading Style Sheets*
EDGE	*Enhanced Data rates for GSM of Evolution*
FDD	*Frequency Division Duplex*
GGSN	*Gateway GPRS Support Node*
GPRS	*General Packet Radio Service*
GPS	*Global Positioning System*
GSM	*Global System for Mobile Communications*
HA	*Home Agent*
HARQ	*Hybrid Automatic Repeat-Request*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
HTTPS	*HyperText Transfer Protocol Secure*
IETF	*Internet Engineering Task Force*
JAX-WS	*Java API for XML Web Services*
JRE	*Java Runtime Environment*
JSP	*Java Server Pages*
JSR	*Java Specification Request*
LTE	*Long Term Evolution*
MIPv6	*Mobile IP v6*
MN	*Mobile Node*
MNN	*Mobile Network Node*

MNP	*Mobile Network Prefix*
MR	*Mobile Router*
MSC	*Mobile services Switching Centre*
NEMO	*Network Mobility*
OLSR	*Optimized Link State Routing protocol*
PMI-IP	*Prototipo de Movilidad e Interoperabilidad IP*
PWRP	*Predictive Wireless Routing Protocol*
RNC	*Radio Network Controller*
SGSN	*Serving GRPS Support Node*
SOAP	*Simple Object Access Protocol*
SOFDMA	*Scalable Orthogonal Frequency-Division Multiple Access*
TDD	*Time Division Duplex*
TETRA	*Terrestrial Trunked Radio*
UDDI	*Universal Description, Discovery and Integration*
UE	*User Equipment*
UMTS	*Universal Mobile Telecommunications System*
USIM	*User Service Identity Module*
UTRAN	*UMTS Terrestrial Radio Access Network*
VLR	*Visitor Location Register*
W3C	*World Wide Web Consortium*
W-CDMA	*Wideband Code Division Multiple Access*
WMN	*Wireless MESH Networks*
WS	*Web Services*
WSDL	*Web Services Definition Language*
WS-S	*Web Services - Security*
XML	*eXtensible Markup Language*
XSD	*XML Schema Definition*

1 Introducción

1.1 Contexto

La aparición de nuevas tecnologías de banda ancha y gran alcance como WiMAX, WiFi-MESH, 3G y 4G celulares (UMTS y LTE) y la evolución de tecnologías actuales como TETRA y redes satelitales, posibilitan nuevos escenarios de comunicaciones móviles.

Estos nuevos escenarios añaden valor y capacidades de comunicación en sectores como el de transporte (por carretera, marítimo, ferroviario, etc.) y la seguridad (cuerpos y fuerzas de seguridad del estado, cuerpos de emergencias, así como el resto de cuerpos que hagan uso de comunicaciones profesionales). En la actualidad, las soluciones de comunicación en estos sectores tienen una capacidad escasa de transmisión de datos y no soportan, en muchos casos, el envío de flujos multimedia.

Las tendencias actuales en estos sectores apuntan hacia una demanda creciente del ancho de banda ofrecido por las redes inalámbricas con el fin de satisfacer los requisitos demandados por la aparición de nuevas aplicaciones en entornos móviles. Estas nuevas aplicaciones hacen, en muchas ocasiones, un uso intensivo de ancho de banda.

Por otro lado, en las redes de comunicaciones móviles utilizadas en entornos de defensa y seguridad, existe una gran variedad de dispositivos de usuario que utilizan diferentes tecnologías y protocolos de comunicaciones, en muchos casos propietarios. Si a esto le unimos la tendencia, cada vez más patente, de evolución de las arquitecturas de red profesionales hacia IP, resulta necesaria la definición de un nodo interoperable capaz de proporcionar conectividad entre los sistemas existentes, con lo cual se multiplicarán las posibilidades de interconexión entre dispositivos de diferentes tecnologías.

A lo anterior hay que sumar la necesidad de que estas redes sean ubicuas, esto es, que permitan a sus usuarios acceder al núcleo de red desde cualquier sitio y en cualquier momento. Además, especialmente en el caso de las comunicaciones para fuerzas de seguridad y cuerpos de emergencias, se exigen fuertes capacidades de robustez y alta disponibilidad en las comunicaciones, así como de un alto grado de seguridad en las mismas.

Es en este entorno descrito, donde se hace patente la necesidad de un router IP móvil, capaz de proporcionar las funcionalidades necesarias que permitan obtener

conectividad entre múltiples redes heterogéneas de forma transparente y segura para el usuario, siempre ofreciéndole las mejores opciones de conectividad de entre todas las disponibles, en base a una lógica compleja configurable mediante un conjunto de reglas. Se trata de un paso más en el camino hacia el Paradigma ABC (*Always Best Connected*), que contempla la posibilidad de comunicación en cualquier lugar, en cualquier momento y a través de la tecnología más apropiada en cada caso, factor determinante en algunos escenarios como son los de emergencia o de seguridad pública.

1.2 Objetivos

Este trabajo se centra en el estudio de las comunicaciones móviles IP en entornos profesionales, más concretamente en el diseño, desarrollo y prueba de un motor de reglas genérico y reutilizable, con el que poder definir escenarios complejos de comunicaciones sobre routers IP móviles, en base a políticas de alto nivel.

En primer lugar, se realizará un estudio de las distintas redes de acceso más relevantes en los escenarios de comunicaciones móviles profesionales, destacando las ventajas e inconvenientes que presentan en su aplicación a los sectores anteriormente comentados. Además, se hará una revisión de los distintos protocolos con los que poder gestionar el problema de la movilidad en redes IP, prestando especial atención al protocolo de movilidad de redes IPv6, *Network Mobility (NEMO) Basic Support Protocol*. También se hará un repaso a la tecnología de servicios web, la cual será utilizada como interfaz para el Gestor de Reglas desarrollado.

Posteriormente se diseñará, desarrollará y probará un Gestor de Reglas, capaz de aplicar sobre un router móvil un conjunto de acciones que sirvan para modelar escenarios complejos de comunicaciones profesionales. Estos escenarios, en general involucran la monitorización de parámetros de red o de sistemas auxiliares (módulos de posicionamiento geográfico, etc.) y la aplicación de configuraciones sobre la plataforma. El Gestor de Reglas, se diseñará como una aplicación independiente, escrita en *Java* y contará con una interfaz basada en *Web Services*, mediante la cual será posible realizar su configuración y gestión.

Estas operaciones se realizarán gracias a un Módulo Cliente el cual se encargará de enviar al Gestor de Reglas la definición de los escenarios de comunicaciones, mediante texto XML con una determinada sintaxis que se definirá más adelante. Además, el módulo cliente servirá como base para el desarrollo de interfaces de usuario con las que gestionar los distintos escenarios de comunicación que se definan. En este trabajo se propondrá un escenario de referencia, y se desarrollarán interfaces de usuario (como aplicaciones *Java* de escritorio y basadas en web) con las que poder hacer uso del mismo.

Por último se extraerán las conclusiones más importantes, y se apuntarán las líneas futuras de desarrollo a seguir para la mejora de la gestión de escenarios complejos mediante reglas, en entornos de comunicaciones móviles profesionales.

1.3 Estructura de la memoria

La memoria de este trabajo presenta la siguiente estructura:

- **Capítulo 1. Introducción:** Contexto, objetivos y estructura del trabajo.

- **Capítulo 2. Estado del arte:** Consiste en una revisión del estado actual de la tecnología involucrada en este trabajo.

- **Capítulo 3. Entorno de desarrollo:** En este capítulo se describen las herramientas y las tecnologías involucradas en la implementación del gestor de reglas que se desarrolla en este trabajo.

- **Capítulo 4. Descripción y análisis de requisitos:** Aquí comienza el ciclo de desarrollo del gestor de reglas de este trabajo, mediante la fase de captura de requisitos. Además se realiza la primera descripción formal del mismo.

- **Capítulo 5. Diseño e Implementación:** Continuando con el ciclo de desarrollo, se realizan las fases de diseño e implementación del gestor de reglas, detallando los módulos en los que se compone, y su consecuente implementación.

- **Capítulo 6. Pruebas y validación:** Define las pruebas realizadas para la validación del gestor, junto con los resultados obtenidos.

- **Capítulo 7. Conclusiones y líneas de trabajo futuras:** Conclusiones finales del trabajo, y líneas futuras de trabajo para la continuidad del mismo.

2 Estado del arte

2.1 Redes de acceso en comunicaciones móviles profesionales

A continuación se realiza una breve revisión de las distintas tecnologías de redes de acceso más destacadas para el sector de las comunicaciones móviles profesionales multimedia, destacando las ventajas e inconvenientes que cada una de ellas presenta.

2.1.1 Wireless MESH Networks

Las redes WMN (*Wireless MESH Networks*), o redes malladas inalámbricas, están compuestas de routers *mesh* y clientes *mesh*. Cada nodo opera no solo como cliente, sino que también se comporta como un router que encamina paquetes en nombre de otros nodos que no están dentro del área de alcance del nodo destino. Una red mallada se auto-organiza, se auto-configura y se auto-repara de forma dinámica, estableciendo y manteniendo la conectividad entre nodos. Estas características acarrean grandes ventajas, como pueden ser el bajo coste de despliegue y configuración, la facilidad de mantenimiento, gran robustez, etc.

El concepto de red mallada es absolutamente general y no depende de la tecnología utilizada, existiendo incluso nodos *mesh* que actúan como *gateways* entre redes MESH basadas en distintas tecnologías. Una de las tecnologías que se están utilizando con mayor asiduidad en entornos MESH es WiFi. WiFi proporciona un soporte ideal para redes locales en términos de alcance y ancho de banda, ampliamente usado en la actualidad.

En cuanto a las arquitecturas de las redes MESH, existen tres tipos distintos en función de las funcionalidades de los nodos que la componen [1]:

- *Backbone WMNs*: Este tipo de redes incluyen un conjunto de routers *mesh* que constituyen una infraestructura para dar soporte a los clientes que se conectan a ella, para permitir la integración de distintas redes inalámbricas ya existentes y para ofrecer acceso a Internet.
- *Client WMNs*: En este tipo de redes no se requieren routers *mesh* (no hay infraestructura) ya que la malla se constituye únicamente entre clientes. En este tipo de redes un paquete destinado a algún nodo, salta a través de múltiples nodos hasta alcanzar su destino, por lo que todos los nodos deben soportar funciones de encaminamiento.
- *Hybrid WMNs*: Este tipo de arquitectura es la combinación de las dos anteriores. Los clientes pueden acceder a la red a través de los routers *mesh*, o directamente a través de algún otro cliente.

Independientemente de la arquitectura o de la tecnología de base, el concepto clave de las redes MESH es el protocolo de enrutamiento, el cual requiere características significativamente diferentes en función de la escala de la red y de las características de los dispositivos. Actualmente hay un gran número de esquemas competentes para el encaminamiento de paquetes a través de redes MESH: AODV [2] (*Ad-hoc On Demand Distance Vector*), B.A.T.M.A.N. (*Better Approach To Mobile Adhoc Networking*) [3], PWRP (*Predictive Wireless Routing Protocol*) [4], OLSR (*Optimized Link State Routing protocol*) [5], etc. El problema más relevante en este sentido está relacionado con la falta de estándar (más que con las prestaciones); por ese motivo, el IEEE está desarrollando un conjunto de estándares bajo el título 802.11s para definir una arquitectura y un protocolo para redes MESH.

2.1.2 WiMAX

WiMAX (*Worldwide Interoperability for Microwave Access*), es un estándar de comunicación inalámbrica (IEEE 802.16 [6]) que proporciona accesos concurrentes en áreas de radio hasta a 50km con velocidades de hasta 70 Mbps, utilizando tecnologías que, en principio, no requieren visión directa con las estaciones base.

En la versión original del estándar WiMAX, la capa física operaba en el rango de 10 a 66 GHz, la cual fue extendida también al rango de 2 a 11 GHz. La modulación utilizada es SOFDMA (*Scalable Orthogonal Frequency-Division Multiple Access*) con 256 sub-portadoras. En cuanto a la capa MAC, WiMAX utiliza un algoritmo de planificación mediante el cual la estación del cliente necesita competir solo una vez para el acceso inicial a la red. Una vez que se ha accedido, se le asigna una ranura de acceso de la estación base. Dicho *slot* de tiempo puede aumentarse o disminuirse, pero permanece asignado al cliente. Además se permite controlar la calidad de servicio (QoS), mediante la asignación de tiempo en dicha ranura.

Estas características están definidas en el estándar actual de WiMAX, que es el IEEE 802.16-2004 junto con la revisión IEEE 802.16e-2005, cuyas novedades más destacadas son las siguientes:

- Soporte a la movilidad (*soft* y *hard handover* entre estaciones base), base de *Mobile WiMAX*.
- Espaciado constante de portadoras (SOFDMA, *Scalable OFDMA*), lo cual se traduce en una mayor eficiencia espectral en canales anchos y en una reducción de coste en canales estrechos.
- Introducción de esquemas avanzados de diversidad de antenas y HARQ (*Hybrid Automatic Repeat-Request)*.
- Introducción de Turbo Coding

Por todas estas características, WiMax se dirige a redes de área metropolitana (MAN) para prestar servicios de tipo comercial altamente escalables a un gran número de usuarios. La facilidad de adición de canales orientada a la maximización de las capacidades de las células, junto con anchos de banda flexibles para el soporte de antenas inteligentes y con la posibilidad de uso combinado con otros estándares (WiFi, por ejemplo), hacen de WiMAX uno de los estándares de mayor ascenso tanto a nivel de investigación como a nivel comercial.

2.1.3 UMTS

UMTS (*Universal Mobile Telecommunications System*) es la tecnología base de las actuales redes de móviles de tercera generación (3G) desarrollada por el 3GPP [7] (*3rd Generation Partnership Project*). La tercera generación contempla y soporta la introducción masiva de servicios de datos además del tradicional servicio de voz, además de introducir un mayor ancho de banda efectivo que permite las aplicaciones multimedia altamente limitadas en los sistemas GRPS y EDGE.

Los servicios proporcionados por UMTS permiten maximizar el número de usuarios en la red global del sistema, incrementar la velocidad de conexión para el usuario móvil hasta a 2Mbps, garantizar *roaming* y diferentes formas de tarificación. Una de las características más relevantes es la posibilidad de acceso a alta velocidad a Internet que permite el soporte de aplicaciones de tipo multimedia y de transmisión de video en tiempo real, ampliando, de manera determinante, los horizontes de la tecnología.

En cuanto a la arquitectura de las redes UMTS [8], están compuestas por tres dominios: el núcleo de red (CN, *Core Network*), la red de acceso radio (UTRAN, *UMTS Terrestrial Radio Access Network*) y los terminales de usuario (UE, *User Equipment*). El núcleo de red básico de UMTS está basado en GSM y GPRS. La UTRAN proporciona la interfaz de acceso radio para los terminales de usuario. Las estaciones base se denominan Nodos-B y el equipo de control asociado a ellas se conoce como RNC (*Radio Network Controller*). Ver Figura 1.

2.1.3.1 Core network

El núcleo de red se divide en dos dominios, el dominio de conmutación de circuitos y el dominio de conmutación de paquetes. Los elementos que forman el dominio de conmutación de circuitos son el MSC (*Mobile services Switching Centre*), VLR (*Visitor Location Register*) y el *Gateway MSC*. En cuanto al dominio de conmutación de paquetes, está compuesto por el *Serving GRPS Support Node* (SGSN) y el *Gateway GPRS Support Node* (GGSN). Existen además algunos elementos compartidos por ambos dominios, como son el EIR, HLR, VLR y AUC.

La transmisión de datos en el núcleo de red está basada en ATM, utilizando las capas AAL2 (ATM Adaptation Layer type 2) y AAL5.

2.1.3.2 UTRAN

La tecnología utilizada en el interfaz aéreo de la UTRAN es W-CDMA (Wideband Code Division Multiple Access), con dos modos básicos de operación: FDD (*Frequency Division Duplex*) y TDD (*Time Division Duplex*).

En cuanto a las funciones que realizan los Nodos-B, se pueden destacar las siguientes: transmisión y recepción en el interfaz radio, modulación y demodulación, codificación CDMA, gestión de errores, control de potencia, etc. En cuanto a los RNC, destacan el control de recursos radio, el control de admisión, la asignación de canales, control del handover, cifrado, segmentación y ensamblado, señalización *broadcast*, etc.

2.1.3.3 Terminales de usuario

El estándar UMTS no restringe la funcionalidad de los terminales de usuario. En UMTS los terminales de usuario (UE) son uno de los extremos de la interfaz radio (en contraposición con los Nodos-B). En cuanto a la UMTS IC tiene las mismas características físicas que la SIM GMS, pudiendo llevar a cabo las siguientes funciones: soporte de aplicaciones USIM (*User Service Identity Module*), soporte de perfiles en la USIM, funciones de seguridad, autenticación de usuario, etc.

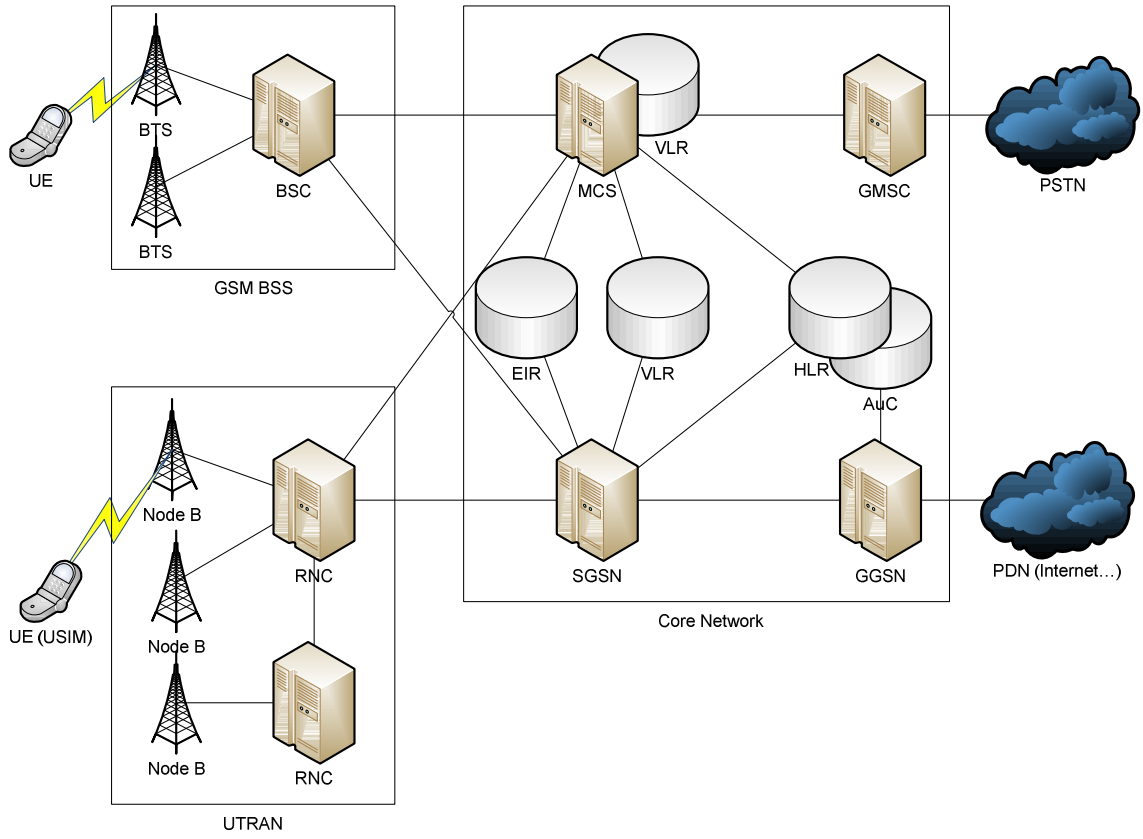

Figura 1. Arquitectura GSM - UMTS

2.1.4 Comparativa

Una vez que se han presentado algunas de las tecnologías de acceso inalámbricas más comunes actualmente, se van a analizar las ventajas y los inconvenientes de cada una de ellas, prestando especial atención a las necesidades de los sectores de comunicaciones profesionales.

- MESH basadas en WiFi
    - Ventajas
        - Tecnología accesible
        - Coste relativamente bajo
    - Inconvenientes
        - Riesgos de interferencias
        - Gestión de la calidad de servicio ineficiente
- WiMAX
    - Ventajas
        - Capacidad elevada
        - Amplias zonas de cobertura
        - Calidad de servicio
    - Inconvenientes
        - Escasa implementación
        - Posibles interferencias en función del modo de implementación
- UMTS – GPRS
    - Ventajas
        - Zonas de cobertura muy elevada
        - Gestión eficiente de la movilidad
    - Inconvenientes
        - Gestión y administración de la tecnología inaccesible (en manos de operadores)
        - No ofrece calidad de servicio

Una vez comentadas las principales ventajas e inconvenientes de estas tecnologías, se presenta una tabla resumen en la que se categorización como bueno, regular o malo, algunas de las principales características de los medios inalámbricos estudiados.

Tabla 1. Comparativa de tecnologías de acceso

	MESH WiFi	WiMAX	UMTS - GPRS
Cobertura urbana	X	X	X
Cobertura rural	X	X	X
Capacidad	X	X	X
Calidad de servicio	X	X	X
Movilidad	X	X	X
Coste	X	X	X

El objetivo de la tabla anterior no es realizar un profundo análisis de las tecnologías, sino poner de manifiesto que ninguna tecnología es por sí misma capaz de ofrecer las características requeridas por los usuarios de comunicaciones profesionales, siendo necesario buscar una tecnología que permita gestionar de forma inteligente el uso de varios medios de transmisión de forma simultánea.

2.2 Movilidad en redes IP

Conforme el acceso a Internet se hace más ubicuo surge la necesidad de estandarizar protocolos que permitan gestionar la movilidad a nivel IP. Desde el IETF (*Internet Engineering Task Force*) [9] se ha detectado esta tendencia actual de los usuarios a la conexión permanente, por lo que se han propuesto una serie de protocolos que a continuación se estudian.

2.2.1 Mobile IP

En IPv4 se asume que la dirección IP de un nodo identifica el punto de red donde reside el nodo. Por tanto, un nodo debe estar situado en la subred indicada por su IP si desea recibir paquetes destinados a él. Si un nodo deseara cambiar su punto de acceso a Internet sin perder sus capacidades de comunicación debería o bien cambiar su dirección IP, o propagar rutas específicas que deberán ser propagadas a todos los routers de Internet. Como se puede ver, ambas soluciones son inaceptables, por lo que el IETF propone la RFC 3344, IP Mobility Support for IPv4 [10].

El objetivo de este protocolo es hacer que los nodos se puedan comunicar con otros nodos, aunque cambien su punto de acceso a la red, sin cambiar su dirección IP. Todo ello mediante el intercambio del menor número de mensajes posibles, y de forma autenticada.

En esta RFC se introducen conceptos comunes al resto de protocolos de movilidad en IP, como son:

- Nodo móvil: Un host o un router que cambia su punto de acceso desde una red o subred a otra. Un nodo móvil deberá ser capaz de seguir comunicándose con otros nodos a través de su dirección IP original.
- *Home Agent*: Un router en la red propia del nodo móvil que envía datagramas hacia el nodo móvil a través de un túnel. Además mantiene información sobre la localización del nodo móvil.
- *Foreign Agent*: Un router en la red visitada que proporciona servicios de *routing* al nodo móvil mientras se encuentre registrado. El *foreign agent* desencapsula los datagramas enviados a través del túnel por el *home agent*. Para los paquetes originados por el nodo móvil, actúa como router por defecto.

Una vez introducidos estos roles, se presenta el proceso de operación general del protocolo Mobile IP:

- Los *foreign agents* y *home agents* informan de su presencia a través de mensajes *Agent Advertisement*
- El nodo móvil determina si está en una red visitada o en la red propia gracias a los mensajes anteriores
- Si el nodo está en la red propia opera de forma habitual. Si éste ha llegado de una red visitada, cancelará su registro en su *home agent*.
- Si el nodo está en una red visitada, obtendrá una dirección IP provisional, llamada *care-of address*, mediante DHCP, o bien a través de los mensajes del *foreign agent*.
- El nodo móvil registra su *care-of address* contra su *home agent* mediante el intercambio de mensajes *Registration Request* y *Registration Reply*, posiblemente a través de un *foreign agent*.
- Los datagramas enviados al nodo móvil son interceptados por el *home agent*, se cursan a través de un túnel hacia la *care-of address* del nodo móvil, se reciben en el otro extremo del túnel (por un *foreign agent*, o bien por el nodo móvil directamente) y finalmente se entregan al nodo móvil.
- En el sentido contrario, los paquetes se pueden cursar de forma normal, sin tener que pasar por el *home agent*.

2.2.2 Mobile IPv6

Mobile IPv6, definido en la RFC 3775 [11], es un protocolo que persigue los mismos objetivos que Mobile IP, pero orientado a gestionar la movilidad en redes IPv6. MIPv6 aprovecha la experiencia desarrollada durante el diseño de MIPv4, y aparte de estar integrado en IPv6, ofrece una serie de ventajas fundamentales sobre su predecesor:

- No se necesitan *foreing agents*
- Soporta optimización de rutas de forma segura
- Menos *overhead* en los mensajes del protocolo
- Desacoplado de la capa de enlace, mediante el uso de IPv6 Neighbor Discovery en vez de ARP.

El funcionamiento básico del protocolo es análogo al comentado en Mobile IP. El nodo móvil obtiene una dirección IPv6 en la red visitada (*care-of address*) a través de los mecanismos habituales de IPv6 (*stateless o stateful auto-configuration*). En MIPv6, al proceso de asociación entre la dirección de la red visitada y la dirección habitual se le conoce como *binding*, intercambiado mensajes *Binding Update* y *Binding Acknwoledgement*.

En MIPv6 existen dos modos de operación, el primero de ellos, basado en un túnel bidireccional es análogo al caso de MIPv4. El otro modo, basado en optimización de rutas, requiere que el nodo móvil registre su dirección contra el nodo corresponsal (el nodo con el que el nodo móvil desea comunicarse). Con este mecanismo se optimiza la ruta que siguen los datagramas, ya que los datagramas procedentes del nodo corresponsal se cursan directamente hacia la *care-of address* del nodo móvil. Aparte de esta ventaja, también se alivia el tráfico cursado por el *home agent* y se reducen las probabilidades en caso de fallo de éste.

2.2.3 NEMO Basic Support

El protocolo de movilidad de red, *Network Mobility (NEMO) Basic Support*, definido en la RFC 3963 [12], es un paso más en la gestión de la movilidad en redes IP, extendiendo el concepto anterior de movilidad de un nodo hacia la movilidad de toda una red.

Una red móvil se define como una red que cambia su punto de acceso a Internet con el transcurso del tiempo. El router que conecta a esta red móvil con Internet se denomina router móvil (*Mobile Router*, MR). El protocolo NEMO Basic Support (ver Figura 2), asegura la continuidad de las sesiones para todos los nodos de la red móvil, incluso cuando el router móvil cambia su punto de acceso a Internet. Además, ofrece conectividad para todos los nodos de la red móvil (MNNs, *Mobile Network Nodes*) aunque ésta cambie de ubicación.

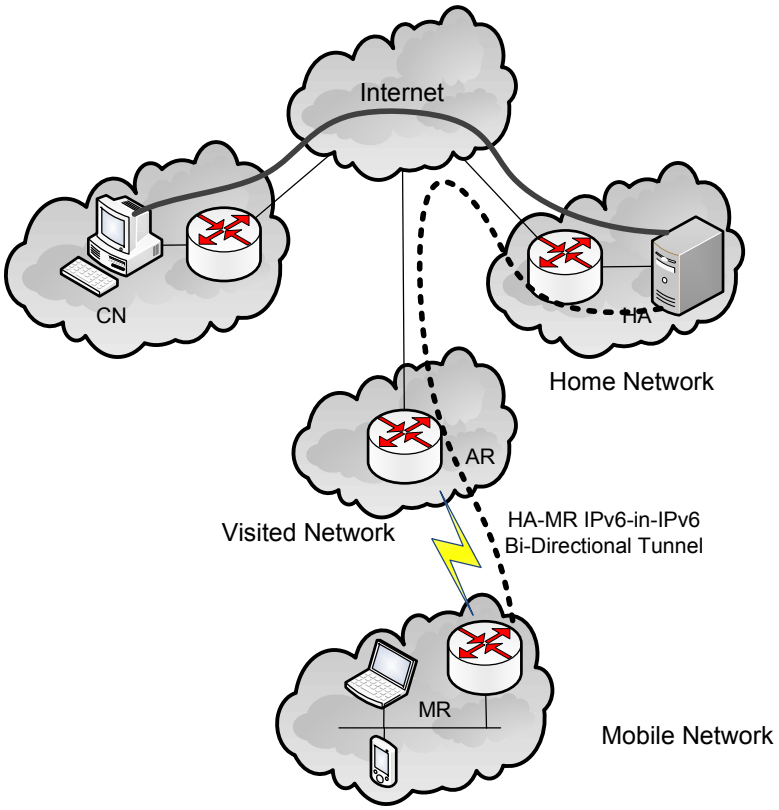

Figura 2. Arquitectura de NEMO Basic Support

La red móvil es parte de la red propia (*Home Network*), por lo que ha sido configurada con un bloque de direcciones de las asignadas a la red propia, lo que se denomina MNP (*Mobile Network Prefixes*). Es importante tener en cuenta que esas direcciones permanecen asignadas a la red móvil cuando ésta se encuentra fuera de su *home network*, por lo que los paquetes con destino alguno de los MNNs serán encaminados hacia la *home network*. Además, cuando el router móvil está fuera de la red propia, adquiere una dirección de la red visitada, la cual se denomina igual que en MIPv6, *care-of address* (CoA).

Cuando un nodo en Internet, denominado nodo corresponsal (CN), desea comunicarse con un MNN, los datagramas IP enviados por el CN llevarán como dirección IPv6 destino la del MNN, la cual pertenece al prefijo de direcciones de la red móvil. Como se dijo anteriormente, este datagrama IP será encaminado por los routers de la Internet hasta la *home network*, donde será encapsulado dentro de un nuevo datagrama IP por un *Home Agent*. El nuevo datagrama se envía hacia la *care-of address* del router móvil, con la IP del HA como dirección origen. Mediante este encapsulado se preserva la transparencia del mecanismo de movilidad, ya que ni los nodos pertenecientes a la red móvil, ni el nodo corresponsal son conscientes del protocolo

NEMO. Así pues, el router móvil recibe el datagrama IP encapsulado, elimina la cabecera IPv6 exterior, y entrega el datagrama original al MNN.

En la dirección opuesta, el funcionamiento es análogo. El router móvil encapsula los datagramas enviados por los MNNs hacia el *home agent*, el cual se encarga de reenviar el datagrama original a su destino (el nodo corresponsal). Este encapsulado es necesario para evitar problemas de filtrado de entrada, ya que muchos routers implementan políticas de seguridad que no permiten el reenvío de paquetes que tienen una dirección origen topológicamente incorrecta.

Por tanto, se puede ver, que de la misma forma que en MIPv6, la solución propuesta por *NEMO Basic Support* está basada en la utilización de un túnel bidireccional entre router móvil y *home agent*. Los mensajes que se intercambian son también análogos a los de MIPv6: Binding Update y Binding Acknowledgement, para asociar la *care-of address* del router, al prefijo de la red móvil.

2.3 Motores de Reglas

Actualmente los motores de reglas se suelen aplicar en entornos empresariales, donde los procesos y la propia lógica de negocio son tan dinámicos que es casi imposible el gestionarlos mediante un programa software como tal. La utilización de motores de reglas, pretende conseguir un desarrollo de herramientas de negocio mucho más ágiles. Un motor de reglas se puede definir como un conjunto de herramientas que permite a los desarrolladores y analistas de negocio construir una lógica de decisiones basada en los datos de la organización. El motor de reglas aplica reglas y acciones definidas por los usuarios sin afectar a cómo la aplicación se ejecuta. La aplicación está diseñada para manejar dichas reglas, las cuales se diseñan por separado.

Como se puede extraer del párrafo anterior, la idea subyacente de los motores de reglas es externalizar la lógica de negocio. Un motor de reglas se puede ver como un intérprete sofisticado de expresiones *if-then*. Una regla se compone de dos partes, una condición y una acción, por lo que cuando se cumple la condición, se lleva a cabo la acción. Así pues, los *inputs* de un motor de reglas son un conjunto de reglas y los objetos de datos sobre los que aplica. Los *outputs* están determinados por estos *inputs* y pueden incluir modificaciones sobre los objetos de datos, nuevos objetos de datos o efectos laterales (facturaciones, envíos de emails, etc.).

Teniendo todo esto en cuenta, la aplicación de motores de reglas en el ámbito del negocio, conlleva las siguientes ventajas:

- Las políticas representadas mediante reglas son fácilmente entendibles

- Las reglas tienen un mayor nivel de independencia que los lenguajes de programación
- Las reglas separan el conocimiento de su implementación
- Las reglas pueden ser reescritas sin cambiar el código fuente, por lo que no se necesita recompilar código fuente

En general, existen dos métodos de ejecución en los motores de reglas: Forward Chaining y Backward Chaining. El primero de ellos, forward chaining, está dirigido por datos (data-driven) y por lo tanto es un método reactivo, con hechos y condiciones en memoria que resultan en la ejecución de reglas. A continuación se presenta una figura donde se puede ver el flujo de ejecución de este tipo de motores:

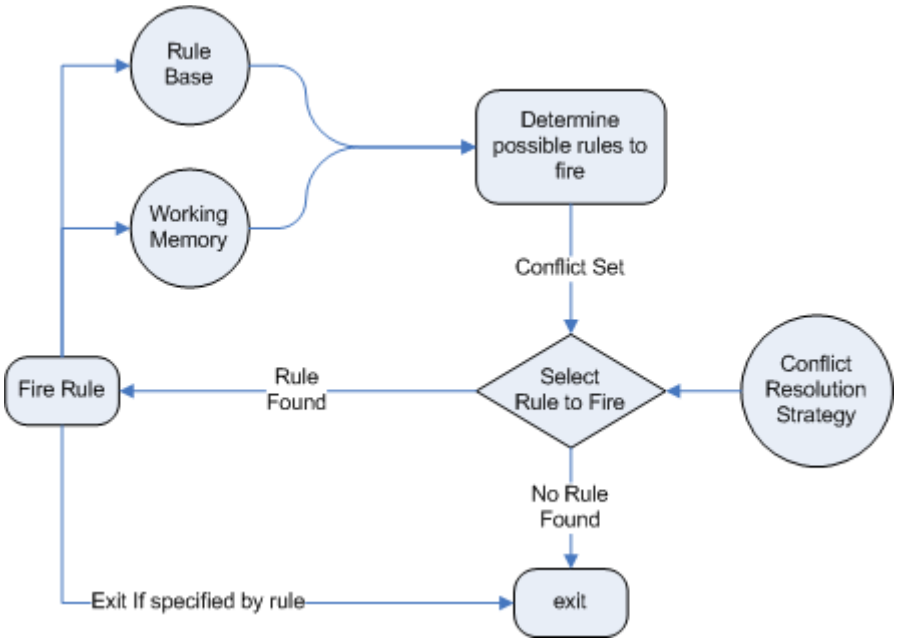

Figura 3. Forward Chaining

El otro método, backward chaining, está dirigido por objetivos, por lo que se comienza con una conclusión (un objetivo) y el motor de reglas intentará satisfacerlo. Si no puede hacerlo, intentará satisfacer un conjunto de sub-objetivos, mediante los cuales satisfacer parcialmente el objetivo global.

En la siguiente figura se muestra con más detalle este proceso:

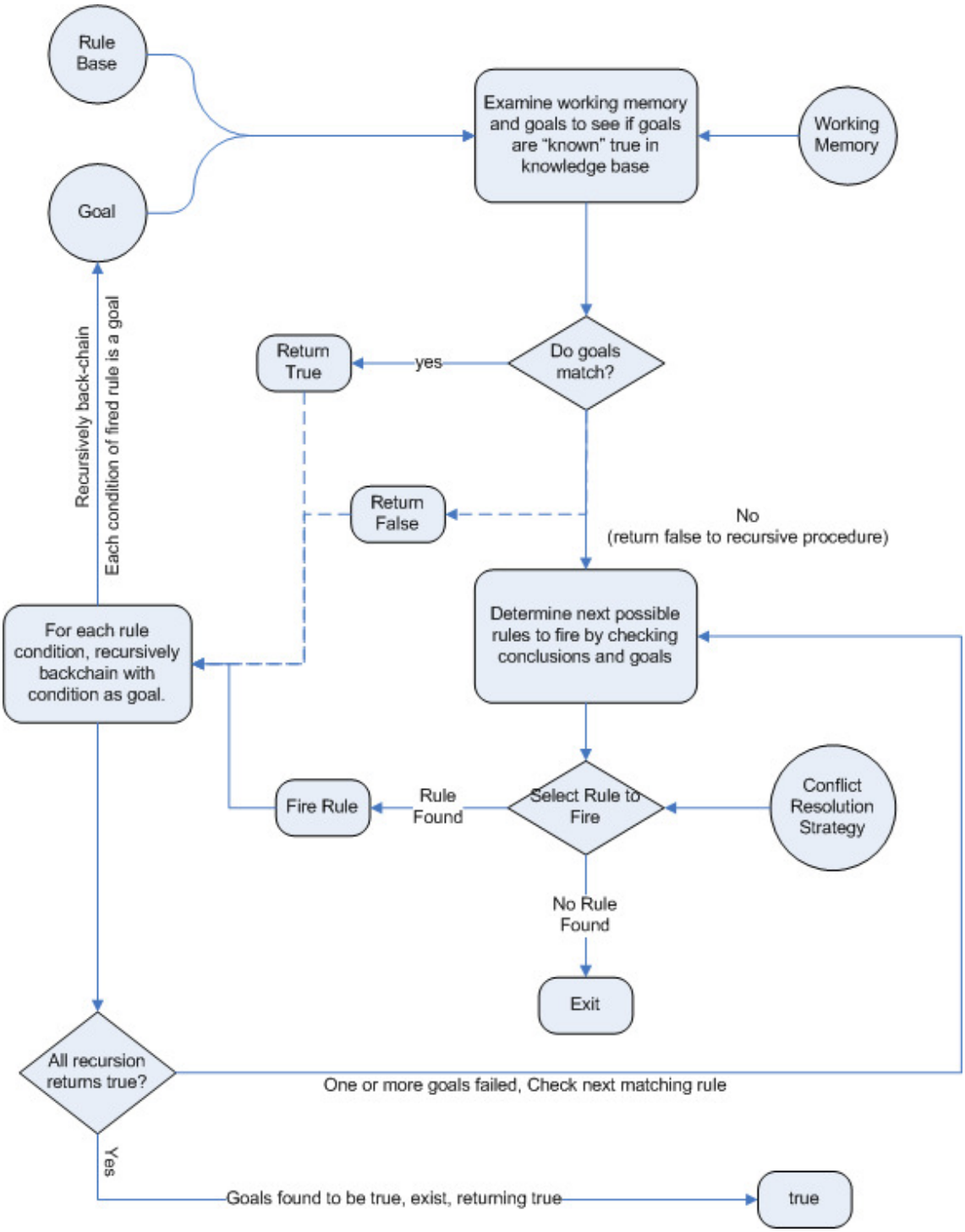

Figura 4. Backward Chaining

En este ámbito de motores de reglas de negocio se está llevando a cabo una iniciativa de estandarización que facilite la integración de aplicaciones y reglas de negocio. *Java Specification Request* 94 (*Java Rule Engine API*) [22] es un intento de estandarizar ciertos aspectos de las implementaciones de motores de reglas Java. Así pues, define un API para registrar y eliminar reglas, parsear y ejecutar reglas y obtener

y filtrar resultados. Sin embargo, no estandariza el motor de reglas en sí, ni el flujo de ejecución del mismo. Tampoco se establece el lenguaje a utilizar para la definición de reglas, ni los mecanismos de despliegue sobre tecnología Java EE.

Existen varios motores de reglas compatibles con la JSR 94, como pueden ser Drools, Fair Isaac Blaze Advisor, WebSphere ILOG JRules, Jess, etc. A continuación se dan más detalles de algunos de los más relevantes actualmente.

2.3.1 Jess

Jess [23] es un motor de reglas escrito en Java por Ernest Friedman-Hill en 1995 en los *Sandia National Laboratories* de Livermore, CA. *Jess* es un motor ligero y rápido, basado en el algoritmo Rete [24], mediante el cual se pueden construir aplicaciones Java con la capacidad de razonar sobre un cierto conocimiento mediante reglas declarativas, mediante la aplicación continua de un conjunto de reglas sobre ciertos datos, gracias a un proceso llamado *pattern matching*. En general, la principal aplicación de *Jess* es la creación de sistemas expertos, o agentes inteligentes.

2.3.2 Drools Expert

Drools es una plataforma de integración de lógica de negocio, la cual se compone de distintos módulos, uno de los cuales, *Drools Expert* [25] es un motor de reglas de negocio. Igual que Jess, Drools implementa y extiende Rete, mediante lo que denominan ReteOO, una mejora y optimización del algoritmo Rete para su uso en sistemas orientados a objetos. Drools utiliza el método de forward chaining, y su arquitectura se muestra en la siguiente figura:

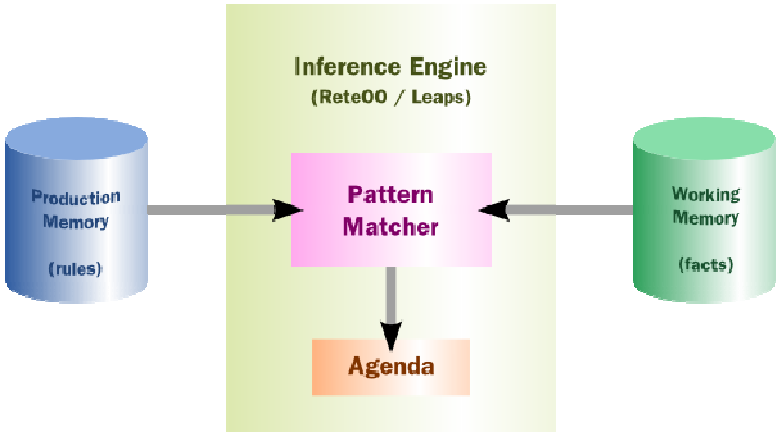

Figura 5. Visión de alto nivel de Drools

Las reglas de negocio se almacenan en la memoria de producción y los hechos sobre los que se aplican se almacenan en la memoria de trabajo. Los hechos son contrastados frente a las reglas mediante un emparejador de patrones. Se debe tener en cuenta también, que en sistemas con gran número de reglas, es posible que las acciones asociadas a las mismas entre en conflicto. Para solucionar este problema, se aplican una

serie de algoritmos en el módulo denominado Agenda, para planificar el orden correcto de las acciones, evitando conflictos.

2.4 Servicios Web

2.4.1 Introducción

Un servicio Web [13] (*Web Service*, en inglés), según el W3C se define como "un sistema software diseñado para soportar interacción entre máquinas en una red. Posee una interfaz descrita en un lenguaje procesable por equipos (WSDL). Otros sistemas pueden interactuar con el Servicio Web, según la forma indicada en su descripción, y usando mensajes SOAP, típicamente transportados mediante HTTP y serializados en XML, en conjunto con otros estándares de la Web".

Dicho de una forma más sencilla, un servicio web se puede definir como un conjunto de tecnologías software estándares (SOAP, WSDL, UDDI) que permiten el intercambio de datos entre aplicaciones. Dichas aplicaciones pueden estar implementadas en múltiples lenguajes, y pueden estar desplegadas en muchos tipos de redes.

2.4.2 Arquitectura

La arquitectura de los *Web Services* [14] involucra múltiples tecnologías relacionadas entre sí mediante capas. Hay varias formas de visualizar dichas tecnologías, de la misma forma que hay distintas maneras de construir y usar un servicio web.

A continuación se presenta una ilustración de algunas de estas familias de tecnologías:

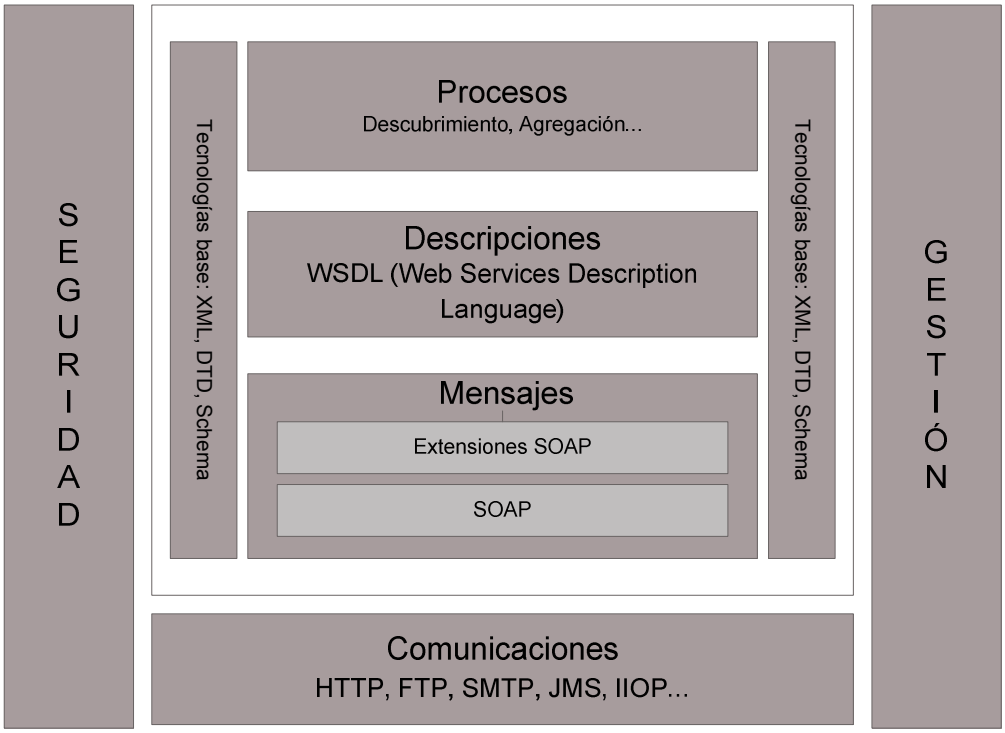

Figura 6. Torre de protocolos de servicios web

La figura representa la denominada Web Services Protocol Stack, que es el conjunto de protocolos y tecnologías que hacen posible definir, implementar, localizar y permitir interacción de servicios web. Estas funcionalidades se reparten en cuatro bloques fundamentales:

- Servicio de transporte: Se corresponde con el bloque "Comunicaciones" que se presenta en la figura. Es el servicio encargado del transporte de mensajes entre aplicaciones en la red. El más utilizado es el protocolo HTTP (*HyperText Transfer Protocol*), el cual se usa para el envío de mensajes XML en la arquitectura de *Web Services*.
- Servicio de mensajería: Se encarga de codificar los mensajes en formato XML. El protocolo más usado a este nivel es SOAP (*Simple Object Access Protocol*), del que se darán más detalle a continuación.
- Descripción del servicio: Todo servicio web debe contar con una interfaz que describa las funcionalidades que ofrece, descrita mediante WSDL (*Web Services Description Language*).
- Procesos: En esta capa se sitúan las actividades necesarias para que la arquitectura de servicios web sea funcional. La más importante de estas tareas es

el descubrimiento de servicios web, basada en UDDI (*Universal Description Discovery and Integration*), el cual es un catálogo XML de servicios que se ofrecen en la red.

Así pues, y mediante el uso de las tecnologías expuestas anteriormente, el intercambio de mensajes de alto nivel que se produce desde que un proveedor publica un servicio, hasta que un usuario lo consume, se puede apreciar en la siguiente figura:

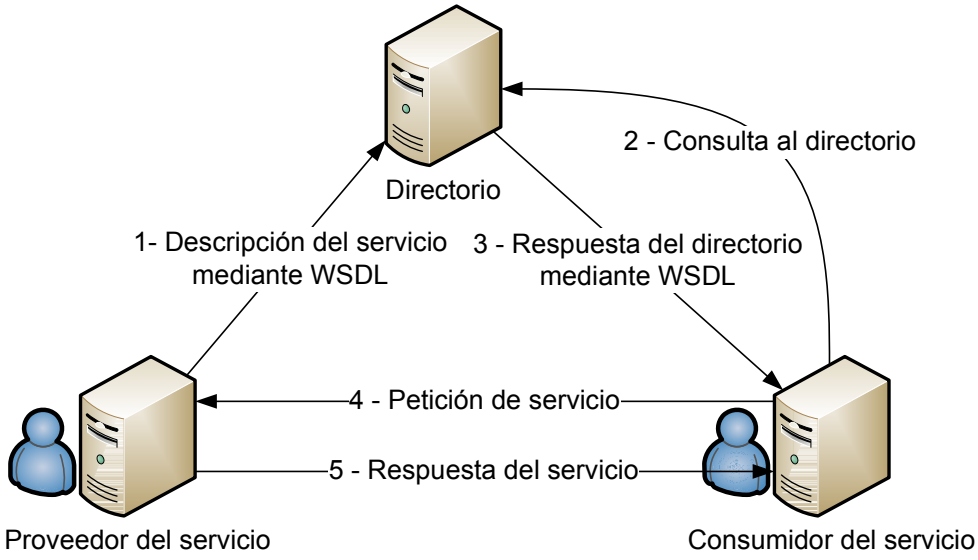

Figura 7. Modelo de utilización de servicios web

A continuación se describen mayor nivel de detalle algunas de las tecnologías clave que hacen posible el funcionamiento de los servicios web.

2.4.3 Arquitectura

SOAP (*Simple Object Access Protocol*) [15] es un protocolo ligero, pensado para intercambiar información estructurada, en un entorno distribuido y descentralizado. Usa tecnología XML para definir un marco de mensajes que pueden ser fácilmente extensibles, ofreciendo además un método de construcción de mensajes que puede ser usado por una gran cantidad de protocolos de nivel inferior. Dicho marco de trabajo ha sido diseñado para ser independiente de modelos de programación o semánticas específicas de implementación.

Los dos principales objetivos de SOAP son la simplicidad y la extensibilidad. SOAP intenta lograr dichos objetivos omitiendo características tales como correlación, seguridad, etc.

Como se ha dicho, SOAP se basa en mensajes XML, los cuales presentan una estructura bien definida [16]. Dichos mensajes presentan la siguiente estructura:

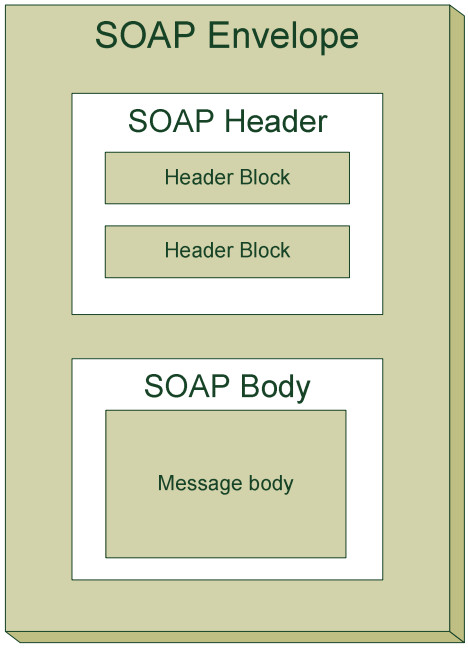

Figura 8. Mensaje SOAP

Como se aprecia en la figura, el primer elemento del mensaje SOAP es el sobre (*envelope*), el cual contiene todo el contenido del mensaje SOAP. Dentro del sobre, pueden aparecer cabeceras SOAP, que son elementos opcionales que ofrecen un mecanismo de extensibilidad para que las aplicaciones puedan intercambiar información no prevista en el estándar. Dentro del elemento *Header* están contenidos los bloques de cabeceras (*Header Blocks*), los cuales representan un conjunto lógico de información, y que además pueden ser asignados a distintos nodos SOAP que se encuentren en el camino entre el usuario origen y el destino (intermediarios SOAP). Por último, el mensaje SOAP presenta un cuerpo (obligatorio) donde se incluye la información extremo a extremo que se quiere transportar.

En cuanto al intercambio de mensajes en SOAP, la forma más común de realizarla es mediante un patrón petición-respuesta, la cual se asemeja a los mecanismos RPC. A parte de este mecanismo SOAP permite a las aplicaciones que utilicen cualquier semántica de envío de mensajes con el que se pueda establecer cierta conversación entre ellas.

En general, el intercambio de mensajes SOAP requiere de un protocolo de transporte que permita hacer llegar el sobre SOAP desde el usuario origen, al destino. SOAP permite el uso de distintos protocolos, incluyendo otros protocolos de nivel de

aplicación, aunque el protocolo más utilizado para el transporte de mensajes SOAP es HTTP.

2.4.4 WSDL

WSDL (*Web Services Description Language*) [17] define una gramática XML mediante la que describir servicios de red como colecciones de nodos finales (*endpoints*) capaces de realizar un intercambio de mensajes. Las definiciones de servicios WSDL proporcionan documentación para sistemas distribuidos y sirven como "receta" para automatizar los detalles involucrados en la comunicación entre aplicaciones.

En general, un documento WSDL contiene los siguientes elementos:

- *type*: es un contenedor de definiciones de tipos de datos, mediante el uso de tipos de sistema.
- *message*: definición de los mensajes que se intercambiarán en la comunicación.
- *operation*: descripción de una acción soportada por el servicio.
- *portType*: conjunto de operaciones soportadas por uno o más *endpoints*.
- *binding*: protocolo y especificación de formato de datos para un *portType* particular.
- *port*: definición de un *endpoint* mediante la combinación de un *binding* y una dirección de red.
- *service*: colección de *endpoints* relacionados.

Es importante observar que WSDL no introduce un nuevo lenguaje de definición de tipos. WSDL es consciente de la necesidad de un importante sistema de tipado a la hora de describir formatos de mensajes, y por ello soporta XSD (*XML Schemas Specification*) como su sistema de tipos canónico. Sin embargo, y debido a que no es razonable esperar que se use una única gramática de tipos para describir todos los formatos de mensajes presentes y futuros, WSDL permite usar otro lenguaje de definición de tipos mediante mecanismos de extensibilidad.

Además, WSDL define un mecanismo de *binding*, el cual es usado para unir un protocolo específico o un formato de datos o una determinada estructura a un mensaje, operación o *endpoint*, lo cual permite reutilizar definiciones abstractas.

3 Entorno de desarrollo

3.1 Introducción

En este capítulo se introducen las principales tecnologías de desarrollo que se han utilizado en la implementación del Gestor de Reglas de este trabajo. En primer lugar se hará un recorrido por las tecnologías Java que se han empleado, destacando las APIs para desarrollo de servicios web JAX-WS, los *Servets HTTP*, y la tecnología JSP (*Java Server Pages*). Además se revisará una de las librerías Javascript más utilizadas en el mundo web, jQuery, la cual será clave en el desarrollo de una de las interfaces de cliente del Gestor de Reglas.

Por último se presentarán las herramientas que se han utilizado en el Departamento de Ingeniería de Sistemas Telemáticos para la realización de este proyecto, como son el entorno de programación Java, servidores de aplicaciones, clientes de servicios web, etc.

3.2 JAX-WS

JAX-WS (*Java API for XML Web Services*) es un API de Java pensada para la creación de servicios web.

La principal función de JAX-WS es realizar un mapeo entre código Java y documentos WSDL. Así pues, se puede generar código Java a partir de un documento WSDL para crear interfaces de *web services* de cliente, y por otro lado, a partir de interfaces Java se pueden generar documentos WSDL para *endpoints* de servicios web.

Su versión más utilizada, la 2.0, se define en la JSR 224 [18] como una extensión al API JAX-RPC 1.1. Con este cambio se quiere reflejar el paso de servicios web con estilo RPC a servicios web basados en documentos.

JAX-WS forma parte de la pila *Metro* que se desarrolla dentro del servidor *Glassfish* y como tal forma parte de Java EE, por lo que es habitual el uso de etiquetas (de tipo *@WebService, @WebMethod*, etc.), que desde la versión 5.0 de Java EE, simplifican el desarrollo y el despliegue de clientes de servicios web, y *endpoints*.

Esta tecnología, junto con el resto de tecnologías auxiliares para servicios web que proporcionan los servidores *Glassfish* (la pila *Metro*), facilita la tarea de publicar métodos de una interfaz en un servicio web, y es especialmente útil en este proyecto para proporcionar a los habilitadores una interfaz de *web services* de una forma cómoda y rápida.

3.2.1 Apache CXF

Apache CXF [19] es un *framework* de código abierto para el desarrollo y la utilización de servicios de web. Su nombre proviene de los dos proyectos en los que está basado, *Celtix* (desarrollado por *IONA Technologies*) y *XFire* (desarrollado por un equipo de *Codehaus*), los cuales fueron fusionados por miembros de la *Apache Software Foundation*.

Las claves del diseño de CXF son las siguientes:

- Separación de los *front-ends* del núcleo del código
- Sencillez en la creación de clientes y *endpoints* sin anotaciones
- Alto rendimiento con poco *overhead* computacional
- Componentes empotrables (con *Spring*, *Geronimo*, etc.)

En cuanto a sus características, destacan las siguientes:

- Soporte a estándares: SOAP, WS-Addressing, WS-Security, etc.
- Implementa JAX-WS
- Implementa JAX-RS: JSR 311 *API for RESTful Web Service Development*

3.3 Java Servlets

3.3.1 Introducción

Los *Servlets* son módulos de código Java que se ejecutan en un servidor de aplicaciones (de ahí el nombre de *Servlets*, en comparación con los *Applets* en el lado de cliente) y que (en general) responden a las peticiones de los clientes generando algún tipo de contenido.

Un *Servlet*, es una instancia de una clase que implementa la interfaz *javax.servlet.Servlet*, aunque la mayoría de los *Servlets*, sin embargo, extienden alguna de las implementaciones estándar de dicha interfaz.

Para inicializar un *Servlet*, un servidor de aplicaciones (o un contenedor de *Servlets* como pueden ser nombrados en ciertos contextos) carga la clase del *Servlet* (y las que éste referencie) y crea una instancia mediante un constructor sin parámetros.

Tras esto, el contenedor de *Servlets* llama al método *init(ServletConfig config)*, en el que el *Servlet* llevará a cabo configuraciones que solo se ejecutarán una vez, y almacenará el objeto *ServletConfig*. Dicho objeto contiene parámetros de *Servlet* y referencia al contexto del mismo. Hay que destacar que el método *init* solo se ejecuta una vez en todo el ciclo de vida del *Servlet*.

Cuando el *Servlet* está inicializado, ya está preparado para recibir las peticiones de los clientes. En cada petición que el cliente realiza, se ejecuta el método

service(ServletRequest req, ServletResponse res). Dicho método puede ser llamado concurrentemente por múltiples hilos.

Por último, cuando el *Servlet* necesita ser descargado, se ejecuta el método *destroy()*, el cual destruye el *Servlet* y libera todos los recursos asignados.

Todo este proceso queda reflejado en la siguiente figura:

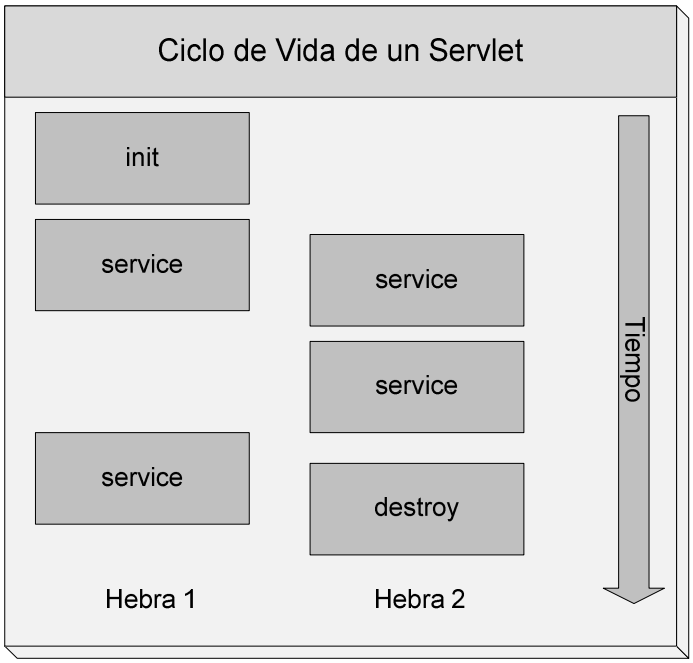

Figura 9. Ciclo de vida de un Servlet

Los *Servlets* no están ligados a ningún protocolo cliente-servidor específico, pero los principales protocolos de aplicación de los mismos son HTTP y SIP. A continuación se profundizará algo más en los *HTTPServlet* los cuales serán utilizados en la implementación de interfaces de usuario basadas en web para el Gestor de Reglas de este trabajo.

3.3.2 HTTP Servlet

El uso masivo del protocolo HTTP ha llevado a que los *servlets* HTTP sean los más utilizados en el desarrollo de servicios basados en Internet.

Los *HTTPServlets* se definen de forma conjunta al *servlet* genérico en la *Java Specification Request* (JSR) 154 [20]. Las APIs de Java definen la clase *HTTPServlet*, que implementa la interfaz *Servlet* y que es extendida por los desarrolladores de servicios para personalizarla a su gusto, esto es, definir el comportamiento que tendrá el *servlet* al recibir cada petición.

La clase *HTTPServlet* añade ciertos métodos que facilitan el procesado de las peticiones HTTP y que son llamadas por el método *service*. Dichos métodos son *doGet*,

doPost, *doPut*, *doDelete*, etc., con los que se gestionan cada uno de los métodos presentes en el protocolo HTTP.

En la JSR 154 se tratan otros temas también importantes, como son la descripción de los objetos de petición y respuesta (*HTTPServletRequest* y *HTTPServletResponse*, respectivamente), aspectos relacionados con las sesiones, y redireccionamiento de peticiones, consideraciones de seguridad, y el mecanismo mediante el cual se asocian a *servlets* a URLs. Esta asociación URL-*servlet* permite que se ejecute un *servlet* determinado en función de qué URL aparece en el método HTTP.

Así pues, la tecnología de *servlets* HTTP permite a un servidor web la generación de contenido dinámico, y la ejecución de cualquier lógica de negocio existente, en función de qué URL se solicita, pudiendo proporcionar al usuario una gran cantidad de servicios.

3.4 JSP

Java Server Pages (JSP) es la tecnología que ofrece la plataforma Java EE para la construcción de aplicaciones que generan dinámicamente contenido web, como HTML, DHTML, XHTML y XML.

La tecnología JSP en su versión 2.0 se especifica en la JSR 152, aunque actualmente se está trabajando en la versión 2.1 en la JSR 245 [21].

La tecnología JSP proporciona los medios necesarios para la creación de una respuesta dinámica basándose en la petición recibida. La tecnología se construye en base a los siguientes conceptos:

- Plantillas: Una porción importante del contenido dinámico es fijo. Textos, o fragmentos XML suelen ser fijos, y se pueden utilizar como plantillas.
- Adición de datos dinámicos: JSP hace posible el introducir contenido dinámico a una plantilla de forma fácil, y además, potente.
- Encapsulado de la funcionalidad: JSP proporciona dos métodos para encapsular las funcionalidades: *JavaBeans* y librerías de etiquetas (*TagLibs*) que ofrecen un amplio abanico de operaciones.

Así pues, la tecnología JSP ofrece una serie de beneficios, entre los que se destacan:

- Separación de roles entre diseñadores y desarrolladores: Esta es una de las principales ventajas de la tecnología JSP, y es que permite que los desarrolladores (aquellos que escriben los componentes que interactúan con los objetos en el lado del servidor) y los diseñadores (los que generan el contenido estático y dan forma a la web en cuanto a presentación) hagan su trabajo con la menor interacción posible.
- Separación de contenido estático y dinámico.

- Reutilización de componentes y librerías.

Proporciona acceso web para aplicaciones corporativas estructuradas en capas: JSP se alinea en la arquitectura Java EE como un *front-end* web, el cual puede interactuar con facilidad con una aplicación corporativa basada en *JavaBeans* conforme al entorno Java EE.

3.5 JavaScript

JavaScript es una implementación del lenguaje *ECMAScript* que típicamente se utiliza para permitir el acceso a objetos dentro de entornos de navegadores web. Es importante tener en cuenta que aunque *JavaScript* y Java tienen una sintaxis (influenciada por C) y convenciones de nombres similares, ambos lenguajes no están directamente relacionados.

JavaScript generalmente se implementa como parte de un navegador web, con el objetivo de enriquecer las interfaces de usuario y los sitios web dinámicos. Sin embargo, se debe destacar que es posible utilizar *JavaScript* fuera del ámbito de los browsers.

Las características más destacadas de *JavaScript* son las siguientes:

- Tipado dinámico: los tipos están asociados con valores, no con variables, por lo que lo una variable puede estar asociada a distintos tipos en distintos instantes de ejecución.
- Orientado a objetos
- Evaluación en tiempo de ejecución
- Con funciones *first-class*: Las funciones se tratan como objetos
- Basado en prototipos

En cuanto al uso que se le da a *JavaScript* en entornos web, que es el que aplica en este trabajo, se debe comentar que el código *JavaScript* viene incluido en el código HTML o asociado al mismo en un fichero independiente. En general, los usos típicos de *JavaScript* han sido abrir *popups*, validar formularios antes de ser enviados al servidor, realizar efectos sobre imágenes, etc. En los últimos años, el uso que se hace de *JavaScript* se ha sofisticado, y las interfaces de usuario de aplicaciones muy utilizadas (como Gmail) le sacan todo el partido posible, realizando la mayor parte de la lógica de la aplicación con *JavaScript*, realizando peticiones al servidor a través de dicho lenguaje. Este último concepto, el de realizar peticiones a servidores web mediante *JavaScript*, se ha explotado ampliamente gracias a la tecnologías AJAX (*Asynchronous JavaScript*).

3.5.1 jQuery

jQuery [26] es una biblioteca o *framework Javascript* de código abierto, creada en un principio por John Resig en 2006, que proporciona una manera sencilla de llevar a cabo operaciones basadas en *Javascript*, que de otro modo requerirían gran cantidad de código. *jQuery* simplifica la manipulación de documentos HTML y hojas de estilos CSS, la gestión de eventos, la animación y las interacciones AJAX, permitiendo un rápido desarrollo web del lado de cliente. Además, está soportado por la mayoría de navegadores, siendo compatible con Firefox 2.0+, Internet Explorer 6+, Safari 2.0.2+ y Opera 9+.

3.6 Cisco AXP

Cisco AXP *(Application eXtension Platform)* [27] es una plataforma de red ideada para proporcionar servicios de red extendidos y facilidades de creación de aplicaciones. AXP se apoya en el paradigma *"Network as a Platform"*, ofreciendo la posibilidad de integrar servicios propios dentro de routers Cisco (Integrated Services Routers).

La arquitectura AXP se muestra en la siguiente figura:

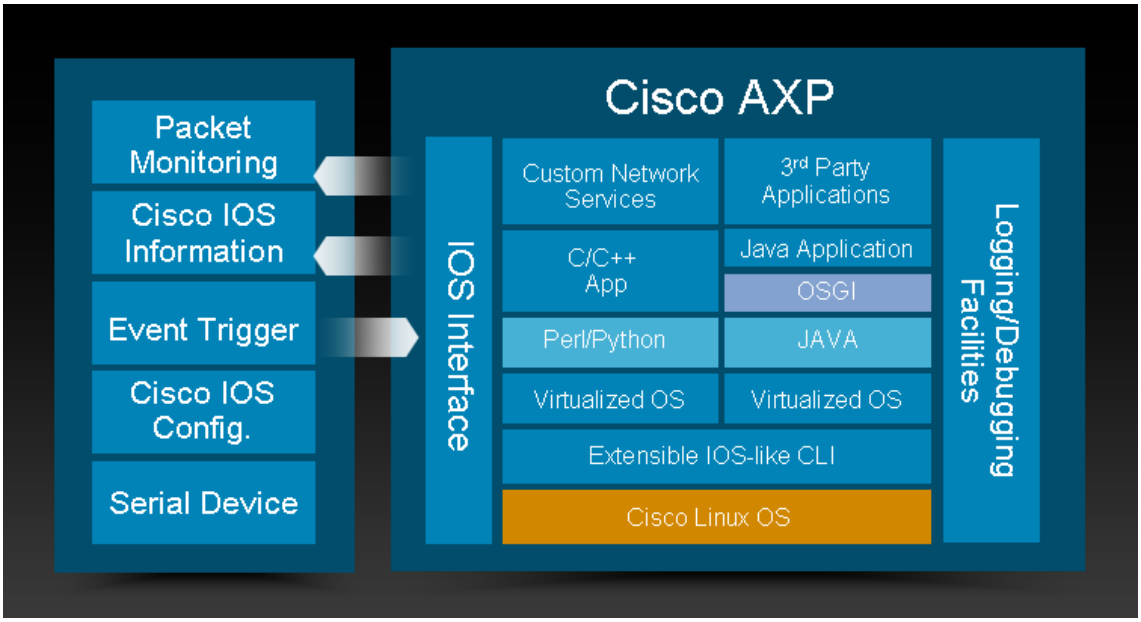

Figura 10. Arquitectura Cisco AXP

Como se puede ver en la figura, Cisco AXP proporciona una serie de recursos sobre un sistema operativo Linux (una versión reducida de CentOS) los cuales ofrecen APIs de alto nivel en lenguajes de programación conocidos, con las que interactuar con el router a través de la interfaz IOS. Sobre estas APIs se construirán los distintos servicios que serán soportados por el router.

A continuación se destacan algunas de las principales ventajas que aporta el uso de Cisco AXP:

- Despliegue de aplicaciones (escritas en Java, C, Perl, Bash, Python, etc.) directamente sobre el router.
- Modificación dinámica de configuraciones en función del tráfico.
- Gestión de eventos.
- Configuraciones a través de APIs de alto nivel.
- Separación entre aplicaciones (protección ante fallos).

4 Descripción y análisis de requisitos

4.1 Introducción

Tras realizar un repaso a las tecnologías y herramientas más relevantes para el desarrollo del Gestor de Reglas de este trabajo, en este capítulo se realizará una descripción formal de dicho gestor, y se hará un estudio de los requisitos que debe cumplir.

El desarrollo de este Gestor de Reglas forma parte de los paquetes de trabajo del proyecto PMI-IP (Prototipo de Movilidad e Interoperabilidad IP), enmarcado en la convocatoria AVANZA2 I+D+i. En concreto, el proyecto PMI-IP potencia el movimiento hacia el Internet del Futuro, influyendo de forma directa en las actuaciones relativas a interfaces avanzadas de radio y soluciones de movilidad, conectividad avanzada, ubicuidad y acceso universal vía comunicaciones por satélite, nuevas plataformas de servicio, seguridad y confianza y la mejora de la eficiencia energética de las redes.

Así pues, el proyecto PMI-IP nació con el propósito de definir un Prototipo dotado de las funcionalidades necesarias para permitir realizar un manejo inteligente de la conectividad a las diferentes redes de acceso móvil disponibles, asegurando al usuario el mantenimiento de una conexión óptima a sus necesidades y a los recursos de red disponibles. Todo ello, garantizando una conectividad segura y robusta frente a caídas o intrusiones.

4.2 Gestor de Reglas

4.2.1 Descripción

En el ámbito de las comunicaciones móviles profesionales, existen escenarios donde la conectividad ofrecida a los usuarios debe ser gestionada en base a distintos parámetros, ya sea el coste, o bien otros factores administrativos. En este contexto es donde el Gestor de Reglas se sitúa, intentando ofrecer una solución eficiente a este problema.

El módulo Gestor de Reglas (*RuleManager* de forma abreviada) es un programa software capaz de gestionar la conectividad proporcionada por un router móvil con múltiples interfaces de forma dinámica, en base a escenarios que se pueden definir siguiendo una determinada sintaxis XML.

De forma genérica, se podría definir al Gestor de Reglas como una herramienta de monitorización y control que permite definir escenarios basados en máquinas de

estados. En dichos escenarios pueden existir parámetros que se deban monitorizar y aplicar acciones en base a su valor. Además se podrán definir transiciones entre estados y se permitirá la ejecución de acciones en momentos arbitrarios indicados por los usuarios del Gestor de Reglas (ejecución de la acción de una regla sin evaluar la condición).

4.2.2 Casos de uso

A continuación, y mediante el estudio de los casos de uso más representativos, se irán concretando los requisitos que debe cumplir el Gestor de Reglas. En estos casos de uso, presentamos diversos escenarios, en los que se describen cómo debería actuar el Gestor de Reglas, y a partir de los cuales se extraerán los principales requisitos del mismo.

Tabla 2. RuleManager - Caso de uso 1

Caso de uso 1	Filtrado de tráfico no prioritario en función del ancho de banda de una determinada interfaz
Actores	Usuarios, Router móvil, Aplicación de Gestión, RuleManager
Descripción	Un conjunto de usuarios están cursando tráfico no prioritario a través de un router móvil en el que está instalado RuleManager. Se desea aplicar una política en la que si el ancho de banda disponible en la interfaz utilizada desciende de un determinado umbral, se filtre el tráfico no prioritario.
Precondiciones	RuleManager está activo y tiene cargado un conjunto de reglas que modelan el escenario descrito. Los usuarios están cursando tráfico no prioritario
Flujo normal	1. RuleManager monitoriza el ancho de banda de la interfaz 2. El ancho de banda desciende 3. RuleManager detecta la bajada del ancho de banda (se sobrepasa el umbral) 4. RuleManager configura un filtrado de tráfico en el router 5. Se notifica a la aplicación de gestión

El caso de uso descrito en la tabla anterior muestra una de las funcionalidades básicas que el Gestor de Reglas debe ser capaz de proporcionar. Se deben poder definir reglas en las que se tenga en cuenta el valor de algún parámetro relevante (en este caso de uso, el ancho de banda de una interfaz) y aplicar acciones en base al mismo (filtrado de tráfico). Como se puede ver, este mecanismo se corresponde completamente con la definición básica de regla, es decir, la ejecución de una acción condicionada a una cierta condición.

Además, se introduce la capacidad de notificar a la aplicación de usuario desde la cual se está controlado al Gestor de Reglas (el actor llamado "Aplicación de Gestión"), con el objetivo de mantener al usuario informado en todo momento de las operaciones que se están llevando a cabo en el Gestor de Reglas.

Tabla 3. RuleManager - Caso de uso 2

Caso de uso 2	Utilización de interfaces en base a posicionamiento geográfico
Actores	Usuarios, Router móvil, Módulo de posicionamiento (GPS), Aplicación de Gestión, RuleManager
Descripción	En un determinado escenario de comunicaciones de un cuerpo de emergencias del estado, se desea condicionar la utilización de la interfaz GPRS al encontrarse fuera del área de cobertura de la red WiFi del centro de control
Precondiciones	RuleManager está activo y tiene cargado un conjunto de reglas que modelan el escenario descrito. El módulo GPS se encuentra activado, y presenta una interfaz accesible desde la que obtener las coordenadas geográficas actuales
Flujo normal	1. RuleManager monitoriza la posición geográfica obtenida por el módulo GPS 2. Si la coordenada corresponde al centro de control, activa la interfaz WiFi y desactiva la interfaz GPRS 3. Si la coordenada corresponde al exterior, activa la interfaz GPRS y desactiva la interfaz WiFi 4. Se notifica a la Aplicación de Gestión

Este caso de uso se basa en la monitorización de información de localización geográfica, proporcionada por un módulo GPS, y en la ejecución de acciones en función de su resultado.

Sin embargo, presenta un matiz frente al caso de uso anterior, o frente a la definición general de regla. En este caso, se plantea la ejecución de acciones tanto si se cumple una determinada condición (estar en el centro de control) como si no se cumple (estar fuera del centro de control). Es decir, se trata de asociar dos condiciones a una misma regla, una a ejecutar si se cumple la condición, y otra a ejecutar si no se cumple. De este modo se optimiza la definición de este tipo de comportamientos, evitando el tener que crear dos reglas complementarias, y reduciendo los conflictos entre reglas.

Además, se pone de manifiesto otro aspecto importante, y es que el Gestor de Reglas debe ser capaz de gestionar información de distintas fuentes. Para la gestión de comunicaciones móviles profesionales no es suficiente con obtener parámetros

proporcionados por el router móvil, sino que también tienen gran relevancia la información que se obtiene de otros sistemas, como en este caso, de un módulo de posicionamiento geográfico.

Tabla 4. RuleManager - Caso de uso 3

Caso de uso 3	Ejecución arbitraria de acciones
Actores	Usuarios, Router móvil, Aplicación de Gestión, RuleManager
Descripción	El usuario, en un determinado momento, puede (a través de la Aplicación de Gestión) ordenar la ejecución de una acción determinada (por ejemplo, aplicar un filtrado de tráfico)
Precondiciones	RuleManager está activo y tiene cargado un conjunto de reglas que modelan el escenario descrito.
Flujo normal	1. El usuario desde la Aplicación de Gestión, inicia la ejecución de una determinada acción, en este caso, un filtrado de tráfico 2. La Aplicación de Gestión se comunica con RuleManager para llevar a cabo la ejecución 3. RuleManager aplica el filtrado de tráfico 4. Se notifica a la Aplicación de Gestión

En este caso de uso, un usuario que esté gestionando el router móvil a través de la aplicación de control de RuleManager, puede aplicar un filtrado de tráfico en cualquier momento, haciendo que la aplicación de control interactúe con el Gestor de Reglas.

Así pues, se pone de manifiesto la posibilidad que ofrece RuleManager, de llevar a cabo acciones sin tener en cuenta ninguna condición, es decir, por voluntad del usuario.

Tabla 5. RuleManager - Caso de uso 4

Caso de uso 4	Escenario de gestión de video de vigilancia en autobús de línea
Actores	Usuarios, Router móvil, Aplicación de Gestión, Cámaras de Abordo, Sistema de posicionamiento geográfico (GPS), RuleManager, Servidores Centrales
Descripción	Se plantea un escenario de flota de autobuses de línea, donde mediante cámaras de abordo se realiza una grabación de vigilancia, tanto del interior del vehículo, como del tráfico.
Precondiciones	RuleManager está activo y tiene cargado un conjunto de reglas que modelan el escenario descrito.
Flujo normal	1. El autobús inicia la marcha desde su estación 2. Comienza la grabación 3. El vídeo se almacena en alta calidad en discos duros instalados en el autobús 4. Por iniciativa del conductor (por alguna incidencia o emergencia), el video puede empezar a transmitirse en tiempo real hasta la sede central 5. RuleManager actúa sobre las cámaras modificando la calidad del vídeo obtenido por las mismas según el estado de la red 6. El autobús llega a las cocheras 7. La grabación se detiene 8. El conductor inicia la descarga del video en los servidores centrales

En este caso de uso, se plantea el escenario de una empresa de autobuses, la cual ha implementado un sistema de vigilancia basado en cámaras de abordo, además de contar con el apoyo de RuleManager para la definición de políticas. En primer lugar, cuando el autobús inicia la marcha desde el punto de salida, RuleManager actúa sobre el sistema de cámaras iniciando la grabación. A partir de ahora, la grabación será almacenada en discos duros instalados en el autobús. En cualquier instante del viaje, el conductor, mediante la aplicación de control de RuleManager, puede imponer que el video se envíe en tiempo real hasta los servidores de la empresa, por motivos de emergencia, por ejemplo. Una vez que el autobús llega a la central, RuleManager detectará la posición geográfica del autobús, y detendrá la grabación. En este momento, el conductor, mediante RuleManager podrá iniciar la descarga del vídeo en los servidores centrales de la empresa.

Gracias a este caso de uso, se ponen de manifiesto aspectos interesantes. En primer lugar, se debe plantear ciertas restricciones en cuanto a la ejecución de reglas de forma arbitraria. Por ejemplo, no tendría sentido que se pudiese iniciar la descarga del video cuando el autobús está en tránsito, o que se comenzase la retransmisión en tiempo real cuando el autobús está en la base.

En una primera aproximación se podría pensar en incluir reglas que evitasen dichas ejecución, sin embargo, y con el objetivo de simplificar y ofrecer mayor nivel semántico, se pueden definir estados, y asociar las distintas reglas a cada uno de estos estados, aplicándose en cada momento las reglas pertenecientes al estado actual del Gestor de Reglas.

Tabla 6. RuleManager - Caso de uso 5

Caso de uso 5	Utilización del Gestor de Reglas tras corte de suministro eléctrico
Actores	Usuarios, Router móvil, Aplicación de Gestión, RuleManager, Servidores Centrales
Descripción	Se plantea un escenario en el que el Gestor de Reglas debe ser capaz de continuar su operación tras un reinicio del mismo, debido por ejemplo a un corte de suministro eléctrico en el router móvil sobre el que opera.
Precondiciones	RuleManager está activo y tiene cargado un conjunto de reglas que modelan el escenario descrito.
Flujo normal	1. Operación habitual de RuleManager 2. Se produce un corte en el suministro eléctrico del router móvil 3. El router móvil está fuera de servicio (RuleManager se cierra de forma "abrupta") 4. Se restablece el suministro eléctrico 5. Se inicia RuleManager 6. RuleManager establece la configuración anterior al fallo 7. Operación habitual de RuleManager

La idea que subyace en este caso de uso es la de dotar al Gestor de Reglas con algún mecanismo que le permita recuperarse a fallos en los que la operación del mismo se ve bruscamente interrumpida.

Así pues, se deberá implementar alguna funcionalidad de persistencia que almacene la estructura de reglas y que permita cargarlas al iniciar el sistema tras un fallo. Sería conveniente además que dicho mecanismo sea configurable, dejando la elección al usuario de utilizarlo en las situaciones en las que sea necesario.

4.2.3 Lista de requisitos funcionales

Una vez analizados los casos de uso más representativos para el Gestor de Reglas (RuleManager), se puede completar una lista con los requisitos funcionales que dicho habilitador debe cumplir.

Tabla 7. RuleManager - Requisitos funcionales

Requisito	Descripción
Definición de elementos de monitorización	El sistema ofrecerá un mecanismo de definición que permita seleccionar los parámetros a monitorizar
Definición de acciones asociadas a elementos de monitorización	El sistema ofrecerá un mecanismo de definición de acciones que permita indicar qué acciones se llevan a cabo en el caso de que se cumpla una determinada condición sobre el elemento monitorizado, o no se cumpla
Definición de estados	El sistema permitirá la definición de distintos estados con los que poder modelar los escenarios de operación del Gestor de Reglas como máquinas de estados finitas
Definición de estados entre transiciones	El sistema ofrecerá un mecanismo que permita indicar de forma arbitraria, o como acción de una regla, el paso de un estado a otro
Mecanismos de asociación de reglas a estados	El sistema será capaz de asociar reglas a estados, imponiendo la ejecución de las mismas solo cuando el sistema se encuentre en el estado adecuado
Mecanismos de ejecución arbitraria de acciones	El sistema permitirá que, en cualquier momento arbitrario, el usuario pueda ejecutar acciones definidas con anterioridad
Utilización de fuentes de información y control diversas	El sistema debe ser capaz de manejar (monitorizar y controlar) elementos diversos, los cuales utilizan en general distintos modelos de información
Mecanismo de notificación	El sistema tendrá la capacidad de enviar notificaciones a la aplicación de gestión cliente para informar de los eventos que acontezcan en la operación de *RuleManager*
Mecanismo de persistencia	El sistema será capaz de recuperarse tras un fallo que suponga la detención abrupta del mismo. Dicho mecanismo será opcional y configurable

Así pues, con estas funcionalidades, el Gestor de Reglas queda definido como una herramienta genérica de monitorización y control, capaz de gestionar escenarios complejos basados en máquinas de estados. Se debe destacar la adaptabilidad que

dicho gestor presenta, ya que ofrece un manejo de reglas bastante flexible (aplicación clásica de reglas, ejecución arbitraria de acciones, etc.), además de ser capaz de manejar información de distintos sistemas, sin ceñirse a manejar objetos en memoria como suele ocurrir con los gestores de reglas de negocio tradicionales.

4.2.4 Lista de requisitos no funcionales

En apartados anteriores, se han desglosado los requisitos funcionales que debe satisfacer *RuleManager*, es decir, las funcionalidades que una vez desarrollado debe ser capaz de ofrecer.

En este apartado se describen los requisitos no funcionales del mismo, es decir, aspectos que el sistema debe cumplir, pero que por sí mismos no aportan nuevas funcionalidades.

Tabla 8. RuleManager - Requisitos no funcionales

Requisito	Descripción
Independencia de plataforma	El sistema podrá ser ejecutado sobre distintas plataformas sin requerir ningún cambio sustancial del mismo
Facilidades de configuración	El sistema deberá ser fácilmente configurable, mediante la utilización de ficheros de propiedades
Aplicación independiente	El sistema deberá poder ser utilizado como una aplicación independiente, es decir, no será necesario desplegarlo dentro de ningún servidor de aplicaciones
Interfaz basada en *Web Services*	El sistema expondrá una interfaz basada en servicios web a través de la cual podrá recibir las reglas que le envíe la aplicación de gestión del cliente. Dicha interfaz también será utilizada para el envío de las notificaciones a dicha aplicación
Autenticación de clientes	El sistema implementará un mecanismo de autenticación que prevea a aplicaciones cliente no legítimas el hacer uso del mismo

4.3 Biblioteca Cliente del Gestor de Reglas

4.3.1 Descripción

De los casos de uso anteriores se extrae la necesidad de contar con aplicaciones cliente que interactúen con el Gestor de Reglas. Dichas aplicaciones serán las encargadas de presentar una interfaz amigable para el usuario gestor (que no tendrá por qué tener una gran experiencia tecnológica) además de gestionar la comunicación

con RuleManager, enviando la descripción de escenarios y reglas, y recibiendo y mostrando las notificaciones oportunas.

Así pues, se plantea la creación de una biblioteca genérica, que actúe como intermediario en la comunicación entre el Gestor de Reglas y estas aplicaciones, abstrayendo a las mismas de los detalles de comunicación involucrados.

4.3.2 Lista de requisitos funcionales

A continuación se presentan los requisitos que la Biblioteca Cliente del Gestor de Reglas deberá satisfacer.

Tabla 9. Biblioteca Cliente del Gestor de Reglas - Requisitos funcionales

Requisito	Descripción
Gestión de la comunicación con *RuleManager*	La biblioteca será capaz de comunicarse con el Gestor de Reglas mediante servicios web
Envío de reglas	La biblioteca, a petición de las aplicaciones que se construyan sobre ella, será capaz de hacer llegar la definición de reglas a *RuleManager*
Recepción de notificaciones	La biblioteca podrá recibir notificaciones que provengan del Gestor de Reglas, informando a las aplicaciones, si éstas así lo desean
Gestión de la autenticación	La biblioteca será capaz de interactuar con el Gestor de Reglas para autenticar a la aplicación de control, siempre que esta proporcione el nombre y la contraseña apropiados

4.3.3 Lista de requisitos no funcionales

En cuanto a los requisitos no funcionales de la Biblioteca Cliente del Gestor de Reglas, se pueden destacar los siguientes.

Tabla 10. Biblioteca Cliente del Gestor de Reglas - Requisitos no funcionales

Requisito	Descripción
Interfaz Java	La biblioteca expondrá métodos en Java para facilitar el desarrollo de aplicaciones de control de *RuleManager*
Integración sencilla	La interfaz expuesta debe permitir un cómodo desarrollo de aplicaciones de control
Aplicación independiente	La biblioteca debe permitir el desarrollo de aplicaciones de control que no requieran ser desplegada en un servidor de aplicaciones

5 Diseño e Implementación

5.1 Introducción

Tras el análisis de los requisitos que debe cumplir el Gestor de Reglas que se desarrolla en este trabajo, las siguientes etapas del modelo de ciclo de vida en cascada son las fases de diseño e implementación.

En la fase de diseño, que se descompone en diseño arquitectónico, y diseño detallado, se definirá la estructura de la solución propuesta para cumplir con los requisitos expuestos en el capítulo anterior, identificando grandes módulos y sus relaciones, tras lo que se estará en condiciones de definir de forma detallada los algoritmos utilizados y la organización del código.

Una vez completada la fase de diseño, se puede comenzar la fase de implementación, en la que se llevará a cabo la codificación de los módulos definidos en la fase de diseño.

5.2 Escenario de referencia

En primer lugar, antes de entrar en detalles de diseño sobre el Gestor de Reglas, se introduce aquí un escenario de referencia que servirá como base en dicho diseño. Este escenario de referencia, se puede ver en la siguiente figura.

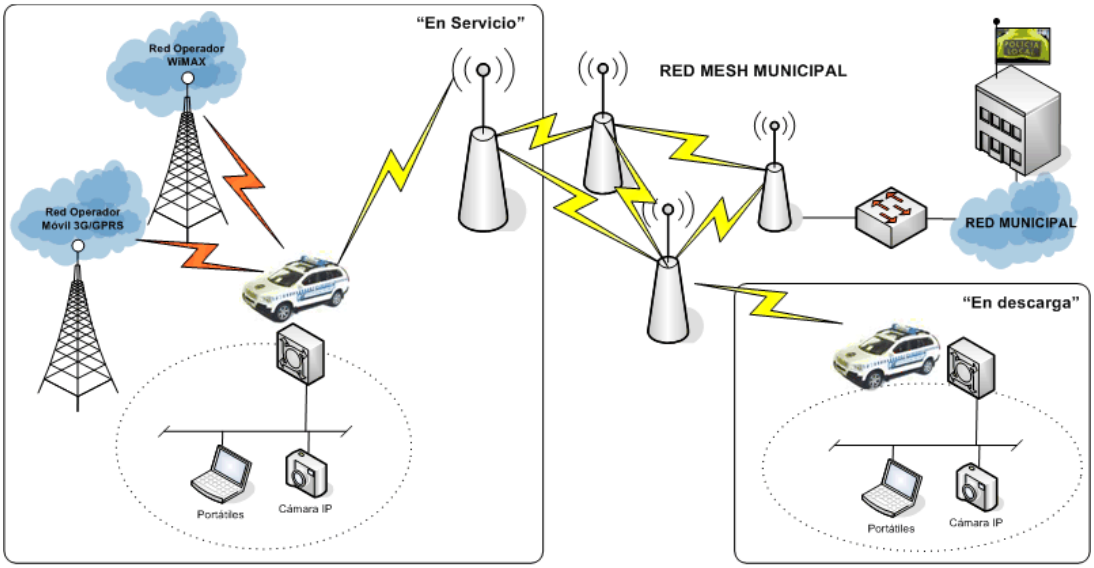

Figura 11. Escenario de referencia

En él se representa un escenario de comunicaciones típico de sectores de seguridad y emergencias, donde se tiene un vehículo equipado con un router móvil en el cual se

encuentra desplegado el Gestor de Reglas. Además, conceptualmente se distinguen dos estados principales: "en servicio" y "en descarga".

En el primero de estos estados, "en servicio", el vehículo se encuentra realizando su operación habitual, moviéndose por distintos lugares, y en general, estando en la zona de cobertura de distintas redes radio. Mientras el sistema de encuentra en este estado, se debe mantener un nivel aceptable de comunicaciones que permita en todo momento estar conectado con el centro de control.

Una vez que el vehículo ha regresado al centro de control, se pasará al estado "en descarga", en el cual los datos que se han ido almacenando a lo largo del tiempo de servicio del vehículo será almacenados en el centro de control a través de la red inalámbrica de alta velocidad del mismo.

Aunque no se representa en la figura anterior, conceptualmente existe otro estado, el estado "apagado", que representa la situación en la que el sistema no está en funcionamiento.

Más formalmente, este escenario se puede modelar mediante el siguiente diagrama de estados, en el que se han incluido una serie de transiciones entre estados.

Figura 12. Escenario de referencia - Diagrama de estados

Como se puede ver, las transiciones entre estado vienen dadas por iniciativa del usuario (arrancar/apagar el sistema, iniciar/detener la descarga intensiva) o bien por fallos eléctricos. En cuanto a las transiciones que se producen en el estado en servicio con sí mismo, representan distintas acciones que se pueden ejecutar en dicho estado: deshabilitar interfaces, cambiar prioridades, balanceo de carga y filtro de tráfico.

5.3 Sintaxis de definición de reglas

Como se ha comentado con anterioridad, la comunicación entre el gestor de reglas y la aplicación cliente se realiza mediante texto marcado en XML, facilitando así el análisis automático por parte del gestor de reglas. Dicha sintaxis se especifica mediante un XML Schema, el cual se incluye en el Apéndice A: XML Schema de definición de reglas.

A continuación se explica brevemente dicha sintaxis, presentando una clasificación de alto nivel de los distintos elementos XML que se han propuesto como lenguaje de especificación de reglas. En primer lugar se debe destacar, que para facilitar la creación de escenarios complejos, se han separado los dos componentes fundamentales de una regla, la condición y la acción. Además, se ha separado también la definición de la regla de la aplicación de la misma.

- Elementos de definición: Estos elementos se utilizan para definir estados, reglas asociadas a dichos estados, y monitores; los cuales, una vez analizados, se almacenarán en memoria a la espera de su aplicación. Las etiquetas XML de estos elementos son: <entry_state>, <entry_action> y <entry_monitor>.
- Elementos de acción: Estos elementos se utilizan para activar o aplicar los elementos anteriormente descritos. Más concretamente, permiten realizar cambios de estados, ejecutar acciones concretas (sin tener en cuenta ninguna condición) o activar monitores. Las etiquetas XML que representan estos elementos son: <execute_state>, <execute_entry_action> y <execute_entry_monitor>.
- Elementos de ejecución de script: En general, todas las acciones que el gestor de reglas realiza sobre la plataforma PMI-IP se aplican mediante la ejecución de scripts alojados en el sistema de ficheros del gestor. Así pues, mediante el elemento <file> se puede hacer referencia a dichos scripts, indicando parámetros, valor esperado de ejecución, y otras configuraciones.

5.3.1 Elementos de definición

<entry_state>

El objetivo de este elemento XML es la definición de estados por parte de la aplicación de usuario. El concepto de estado definido en este contexto se basa en un conjunto de reglas, las cuales solo se podrán ejecutar si el estado al que pertenecen es el estado actual del sistema en el momento de ejecución.

<entry_action>

La etiqueta <entry_action> define una acción perteneciente a un determinado estado. Dicha acción se puede componer, a su vez, de elementos <file> que permitan la ejecución de ciertos scripts, o de elementos <execute_entry_action> con los que de forma indirecta ejecutar otra acción definida dentro del mismo estado.

<entry_monitor>

El propósito de este elemento es definir las características de los monitores que la aplicación de usuario desea aplicar al sistema. De forma análoga a la etiqueta <entry_action>, en el interior de <entry_monitor> se pueden encontrar elementos <file> y <execute_entry_monitor>, que referencien las acciones que el monitor debe llevar a cabo en el momento de su ejecución.

5.3.2 Elementos de ejecución

En general, los elementos de ejecución se utilizan para provocar la ejecución o aplicación del elemento de definición al que referencian. En los elementos de ejecución se pueden incluir dos etiquetas, <success> y <failure>, que contienen un conjunto de elementos de ejecución que son invocados en función del resultado de la ejecución del elemento al que referencian. Si el resultado de la ejecución coincide con el valor esperado, se ejecutarán las acciones incluidas dentro del elemento <success>; en caso contrario lo harán las incluidas en el elemento <failure>.

Así pues, mediante estas dos etiquetas se implementa la regla como tal, asociado a una determinada condición, una acción en función de si el resultado de la condición se ha cumplido (<success>) o no (<failure>).

Es importante matizar que el flujo de ejecución de reglas o monitores no se inicia hasta que la aplicación cliente envíe el correspondiente elemento de ejecución de forma aislada, es decir, no incluido como hijo de un elemento de definición.

<execute _state>

Este elemento permite seleccionar el estado actual del sistema, el cual debe haber sido previamente definido a través del elemento <entry_state> asociado. Una vez realizada la transición, se podrán aplicar las reglas de ese estado almacenadas con anterioridad.

<execute_ entry_action>

El elemento <execute_entry_action> activa la ejecución de una determinada acción, analizando el resultado de dicha ejecución, y en función del mismo, aplicando las acciones pertinentes.

<execute_entry_monitor>

El elemento <execute_entry_monitor> es análogo al anterior, activando en este caso reglas de monitorización.

5.3.3 Elementos de ejecución de scripts

<file>

Mediante el elemento <file> se hace referencia un script que realizará alguna acción sobre la plataforma sobre la que opera el Gestor de Reglas (acciones de configuración, monitorización, etc.), indicando los argumentos que se le deben proporcionar al mismo, y el valor esperado de su ejecución. Además, en el caso de elementos <file> incluidos en elementos <entry_monitor> se puede especificar el ámbito de ejecución del script de monitorización y la temporización del mismo.

5.4 Gestor de Reglas

5.4.1 Diseño

Como se ha comentado anteriormente, *RuleManager* debe ser capaz de gestionar escenarios de comunicación genéricos basados en reglas. A continuación se describen las tareas de diseño más destacadas que se han llevado a cabo sobre *RuleManager*.

Arquitectura

A continuación se muestra la arquitectura diseñada para el Gestor de Reglas:

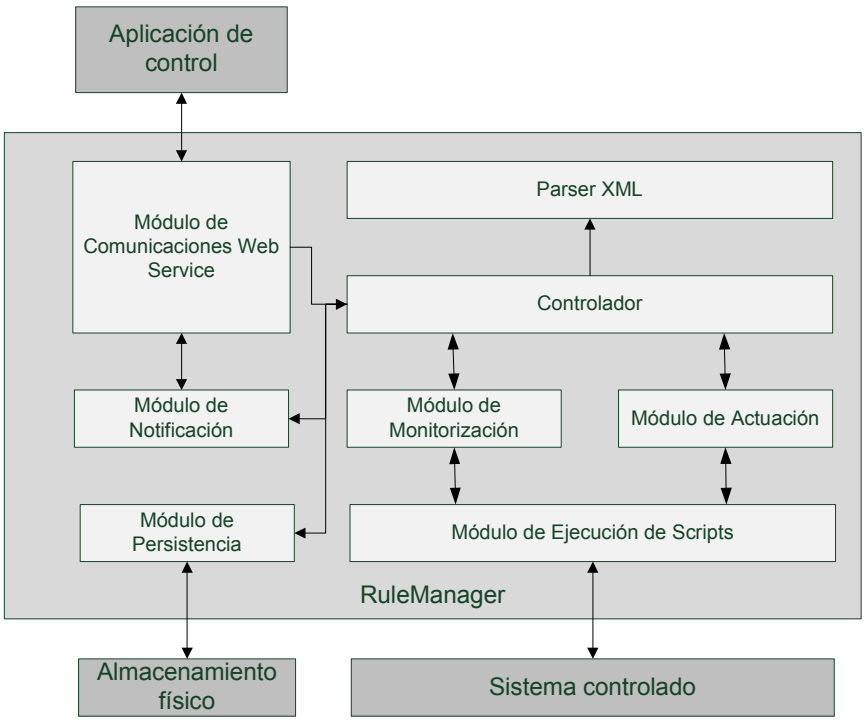

Figura 13. RuleManager - Arquitectura

En esta figura se muestran los principales módulos de los que se compone el Gestor de Reglas de este trabajo, además de presentar las relaciones existentes entre dicho gestor y los elementos externos sobre los que actúa, por un lado con el sistema sobre el cual se aplican las reglas (sistema controlado) y por otro lado con la aplicación de control que permite al usuario final tener el control sobre ciertas acciones que se desarrollan en el sistema.

Como se puede apreciar en la figura, el módulo central del Gestor de Reglas es el "Controlador", el cual se encarga de dirigir el flujo de ejecución entre los distintos módulos. Este módulo controlador, se encarga de procesar las peticiones que recibe a través del "Módulo de Comunicaciones Web Service", utilizando un módulo analizador de XML auxiliar para generar la estructura de objetos que modele las reglas que el usuario inyecta.

Se debe recordar en este punto que la definición de reglas se realiza siguiendo una determinada sintaxis XML, para lo cual se define un *XML Schema*. Dicho esquema se presenta en el Apéndice A: XML Schema de definición de reglas.

Una vez que se han analizado y verificado las reglas introducidas en el gestor, se pasa el testigo a los módulos de monitorización y actuación, los cuales se encargan de realizar el seguimiento de los parámetros indicados del sistema controlado y actuar sobre ellos, respectivamente.

Como resultado de estas operaciones se producirán eventos susceptibles de ser notificados a la aplicación de control del cliente, para lo cual se utilizará el módulo de notificaciones. Dicho módulo, se apoya en el módulo de comunicaciones para informar al cliente de eventos relevantes mediante la utilización de un método *web services* que expone la biblioteca de cliente.

Definición de la interfaz Web Service

En este apartado se describe la interfaz ofrecida por el Gestor de Reglas para implementar las funcionalidades indicadas más arriba.

Gestión de sesión

Para la creación y eliminación de sesiones, *RuleManager* ofrece las siguientes operaciones:

- *getIdSession(String XML)*, acepta un fragmento XML de inicio de sesión en el que se incluyen los parámetros necesarios (nombre y *password*).
- *deleteIdSession(String XML)*, realiza el cierre de sesión, invalidando el id de sesión obtenido con la operación *getIdSession*.

Envío de reglas

Para el envío de reglas de actuación y de monitorización, *RuleManager* ofrece la siguiente operación:

- *sendXML(String XML)*

Notación formal de la interfaz Web Service

- *getIdSession(String getIdSessionXML)*
  - Mensaje de petición

Nombre de parte	Tipo	Opcional	Descripción
getIdSessionXML	String	No	Fragmento XML de petición de inicio de sesión. Deberá incluir un usuario y contraseña válidos

  - Mensaje de respuesta

Nombre de parte	Tipo	Opcional	Descripción
sessionId	String	No	Cadena de texto que representa el identificador de sesión asociado a los datos enviados con getIdSession

  - Errores asociados
    - *IncorrectLogin*. Error lanzado cuando los datos de inicio de sesión no son válidos.
    - *XMLException*. Error lanzado cuando el texto marcado en XML que se envía no es válido.

- *deleteIdSession(String deleteIdSessionXML)*
  - Mensaje de petición

Nombre de parte	Tipo	Opcional	Descripción
deleteIdSessionXML	String	No	Fragmento XML de petición de cierre de sesión. Deberá incluir el identificador de la sesión que se desea cerrar.

  - Mensaje de respuesta

Nombre de parte	Tipo	Opcional	Descripción
-	-	-	-

- Errores asociados
  - *IncorrectSessionID*. Error lanzado cuando el identificador de sesión que se indica en el texto XML no es correcto.
  - *XMLException*. Error lanzado cuando el texto marcado en XML que se envía no es válido.

- *sendXML(String ruleXML)*
  - Mensaje de petición

Nombre de parte	Tipo	Opcional	Descripción
ruleXML	String	No	Fragmento XML en el que se describen reglas de actuación, monitorización, cambio de estados, etc.

  - Mensaje de respuesta

Nombre de parte	Tipo	Opcional	Descripción
-	-	-	-

  - Errores asociados
    - *IncorrectSessionID*. Error lanzado cuando el identificador de sesión que se indica en el texto XML no es correcto.
    - *XMLException*. Error lanzado cuando el texto marcado en XML que se envía no es válido.

Diagrama de secuencia

Para ilustrar lo expuesto anteriormente, se presenta a continuación un diagrama de secuencia con el intercambio de mensajes que se debe llevar a cabo en una secuencia de operaciones típica en *RuleManager*.

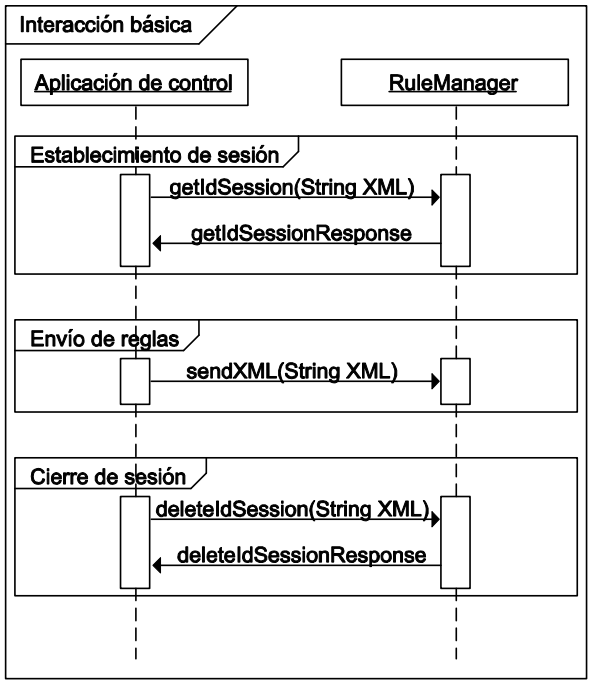

Figura 14. Diagrama de secuencia básico

En este diagrama se muestra cómo las aplicaciones clientes necesitan obtener un identificador de sesión, mediante *getIdSession*, para utilizarlo durante el transcurso de la misma, en el envío de reglas, con *sendXML*. Por último las aplicaciones de control liberan la sesión utilizando el método *deleteIdSession*.

5.4.2 Implementación

Como ya se ha indicado anteriormente, la implementación de este módulo Gestor de Reglas estará basada en tecnología Java. Concretamente se utiliza la tecnología Java SE, el parser XML Apache Xerces-J y el *framework* de desarrollo de servicios web *Apache CXF*.

Clases

La implementación de RuleManager se ha estructurado en varios paquetes, los cuales se van a ir describiendo a continuación.

Paquete *rulemanager*

Es la columna vertebral del gestor de reglas, ya que es el encargado de orquestar los distintos módulos que lo componen, mediante una serie de clases que realizan funciones de alto nivel. Más concretamente se encarga de gestionar las siguientes funciones:

- comunicación con el módulo cliente a través de una interfaz basada en Web Services

- gestión de flujo de información entre la interfaz *web services* y los *parsers* XML
- gestión del mecanismo de trazado interno de sucesos (*logging*)
- gestión de escenarios (entendiendo escenario como cada uno de los distintos diagramas de estados que se pueden modelar mediante el gestor de reglas)

Se compone de las siguientes clases:

- Interfaz *RuleManager*

  Establece los métodos *Web Services* que expone el gestor de reglas:

  - *getIdSession*: inicio de sesión.
  - *sendXML*: envío de documentos XML por parte del cliente.
  - *deleteIdSession*: cierre de sesión.

- Clase *RuleManagerImpl*

  Implementa la interfaz *RuleManager* desarrollando los métodos que la componen. Desde un punto de vista de alto nivel, la clase *RuleManagerImpl* actúa como controlador de las distintas operaciones que la interfaz *RuleManager* expone, invocando a los módulos adecuados en cada caso para el correcto procesamiento de las mismas. En general, coordina las siguientes funciones:

  - Parseo de XML
  - Gestión de sesiones
  - Mantenimiento de reglas en memoria
  - Carga inicial de reglas
  - Mecanismo de persistencia de reglas

- Clase *RuleManagerConstants*

  En esta clase se definen todas las constantes (nombres de elementos XML y de parámetros de configuración) utilizadas en RuleManager.

- Clase *FileLogger*

  Esta clase es la encargada de gestionar los informes de trazas generados por *RuleManager*. Se define con el fin de tener un único *logger* común a todas las clases y permitiendo así almacenar todas las trazas en un fichero de texto común. Se incluyen además métodos de configuración en los que, mediante un fichero de parámetros, el administrador del gestor de reglas puede indicar el nivel de trazado deseado.

- Clase *Scenario*

  La clase *Scenario* modela las distintas máquinas de estados que se pueden definir en el gestor de reglas, las cuales quedan definidas como un conjunto de estados y monitores.

- Clase *State*

  Mediante la clase *State* se modela cada uno de los estados que un escenario puede tener. Más concretamente, la clase *State* mantiene una lista con las distintas acciones (ver clase *EntryAction*) que el estado posee.

Paquete *session*

Este paquete es el encargado de gestionar el inicio, mantenimiento y cierre de sesión por parte de los usuarios.

Se compone de dos clases:

- Clase *SessionParser*

  Mediante este *parser* se analiza el fragmento XML correspondiente al inicio y cierre de sesiones. El modelo de parseo utilizado es SAX, por lo que esta clase extiende a *DefaultHandler*, definida en org.xml.sax.

  Básicamente su funcionamiento se basa en detectar los parámetros necesarios e invocar los métodos adecuados de la clase *SessionManager*, que es la que implementa la lógica de la gestión de sesiones.

- Clase *SessionManager*

  Esta clase implementa la lógica de la creación, mantenimiento y eliminación de sesiones. Proporciona métodos de *login*, donde se compara el nombre y contraseña proporcionados por la aplicación de usuario, devolviendo (en caso de ser correctos) un identificador que la aplicación de cliente debe incluir en las siguientes peticiones al gestor de reglas. Mediante la operación de eliminación de sesión, dicho identificador será invalidado, no pudiéndose usar en las siguientes peticiones.

Paquete *parser*

La misión de este paquete es analizar las reglas y monitores expresados en XML que la aplicación cliente envía, con el doble objetivo de verificar que la estructura del documento XML es conforme al XML Schema base del proyecto, y por otro lado,

detectar cada elemento XML y redirigir el flujo de ejecución hacia las funciones de tratamiento correspondientes.

Lo componen las siguientes clases e interfaces:

- Clase *RuleManagerParser*

  Esta clase es el *DocumentHandler* que se encarga del análisis del XML que las aplicaciones clientes envían, apoyándose en objetos que implementan la interfaz que se verá a continuación, *CallbackAnalyzer*.

- Clase *RuleManagerErrorHandler*

  Maneja los distintos errores que se pueden desprender del análisis del XML recibido, gestionando el lanzamiento de distintas excepciones en función del grado de importancia y procedencia de estos.

- Interfaz *CallbackAnalyzer*

  Interfaz que deben implementar los objetos utilizados en el análisis de los distintos elementos XML que se reciben del cliente. Con el uso de esta interfaz, se consigue tener modularidad en el tratamiento de los distintos elementos, ya que cada objeto que implemente *CallbackAnalyzer* será experto en el análisis de un único tipo de elemento XML.

Paquete *action*

Las clases de este paquete modelan los distintos elementos XML soportados por *RuleManager*, exceptuando aquellos que representan monitores, los cuales se han separado en otro paquete.

- Interfaz *Executable*

  Interfaz que define los métodos que deben implementar los objetos que modelan los elementos XML manejados en *RuleManager*. Define métodos básicos para invocar la ejecución y permitir el paso de resultados entre objetos.

- Clase *EntryAction*

  Los objetos de esta clase modelan el elemento <entry_action>, almacenando las distintas acciones que lo definen, las cuales serán ejecutadas cuando el elemento <execute_entry_action> asociado lo indique.

- Clase *ExecuteEntryAction*

Los objetos de esta clase guardan una referencia al objeto *EntryAction* asociado, el cual una vez ejecutado y, en función de si el resultado de la ejecución es el esperado o no, provocará la ejecución de las instrucciones asociadas a la etiqueta <success> o <failure>, respectivamente.

- Clase *ExecuteState*

Los objetos de esta clase se encargan de establecer como estado actual del sistema un estado definido previamente y, en función de si dicho cambio de estado se realizado correctamente o no, ejecutar las instrucciones asociadas a la etiqueta <success> o <failure>, respectivamente.

- Clase *ExecutionResult*

Mediante los objetos *ExecutionResult*, los *threads* que se utilizan para la ejecución de reglas informan de los resultados de sus procesos. Se debe destacar que los métodos de esta clase son sincronizados con el fin de evitar problemas asociados a la concurrencia, ya que son objetos compartidos entre el objeto que invoca la ejecución (y espera el resultado) y el objeto que realiza la ejecución.

- Clase *FileRef*

Los objetos de esta clase modelan los elementos <file> definidos anteriormente. Así pues, ejecuta el *script* que se indica en el fragmento XML y, en función de si el resultado es el esperado o no, ejecutará las instrucciones pertinentes.

Paquete *monitor*

Las clases de este paquete modelan los distintos elementos XML soportados por *RuleManager* para la creación y ejecución de elementos de monitorización. Se compone de tres clases:

- Clase *MonitorManager*

Mantiene una lista con los distintos monitores que se definen en cada escenario, proporcionando métodos útiles para su creación y gestión.

- Clase *EntryMonitor*

Esta clase modela los elementos XML <entry_monitor> utilizados para la definición de elementos de monitorización, almacenando las distintas acciones que lo componen, y que serán ejecutadas cuando un <execute_entry_monitor> asociado a este <entry_monitor> lo indique.

- Clase *ExecuteEntryMonitor*

  Se encarga de ejecutar el objeto EntryMonitor al que referencia y, en función de si el resultado es el esperado o no, ejecutar las instrucciones asociadas a la etiqueta <success> o <failure>, respectivamente.

Paquete *exceptions*

Se definen tres subclases de la excepción *SAXException* para representar tres tipos posibles de errores:

- Clase *XMLException*

  Error lanzado cuando la estructura del documento XML no es válida.

- Clase *IncorrectSessionIDException*

  Error lanzado cuando el identificador de sesión no corresponde con ninguna sesión activa en ese momento.

- Clase *IncorrectLoginException*

  Error lanzado cuando los datos de inicio de sesión no son válidos (usuario y/o contraseña erróneos).

Paquete *event*

El objetivo de este paquete es implementar un sistema de notificaciones basadas en servicios web entre el gestor de reglas y la aplicación cliente. De esta forma el cliente tiene constancia de la evolución de distintas acciones de interés llevadas a cabo por *RuleManager*.

Lo componen las siguientes clases e interfaces:

- Interfaz *Listener*

  Interfaz que establece los métodos *Web Service* utilizados por el servidor para el envío de las notificaciones al cliente.

- Clase *NotificationManager*

  Gestiona el envío de notificaciones al módulo cliente, ofreciendo métodos de notificación que son consumidos por el resto de objetos de *RuleManager*.

- Clase *NotificationSender*

Esta clase consume el método *Web Service* que los clientes exponen para recibir notificaciones. Es por tanto la clase en la que se apoya *NotificacionManager*.

Paquete *script*

Este paquete se utilizará para administrar la ejecución de los distintos scripts que tendremos definidos en los documentos XML.

Lo componen las siguientes clases:

- Clase *Script*

  Clase auxiliar capaz de ejecutar *scripts* en entornos *Windows* o *Linux*, redirigiendo los *streams* de salida y error, y devolviendo su contenido a los objetos que la utilizan.

- Clase *ScriptExecuter*

  Clase que gestiona la ejecución de *scripts* mediante el uso de la clase anterior. Más concretamente se encarga de la creación y ejecución de objetos *Script*, abstrayendo al resto de elementos de RuleManager del funcionamiento de dicha clase.

Paquete *persistence*

Este paquete implementa la funcionalidad de recuperación ante fallos que *RuleManager* posee. Como se ha comentado anteriormente, esta funcionalidad permite que el gestor de reglas recupere su estado tras un rearranque del mismo, sin requerir intervención por parte de la aplicación cliente.

Lo componen las siguientes clases:

- Clase *PersistenceManager*

  Los métodos de los objetos de esta clase ofrecen las funcionalidades de alto nivel del mecanismo de persistencia, como son el almacenamiento y carga de reglas en disco y el mantenimiento de sesiones de usuario tras la recuperación del gestor.

- Clase *PersistenceUtils*

  Esta clase ofrece funciones auxiliares útiles para la clase anterior, relacionadas con la gestión de los ficheros de persistencia en disco.

5.5 Biblioteca cliente del Gestor de Reglas

5.5.1 Diseño

El objetivo de la biblioteca cliente del Gestor de Reglas es proporcionar una capa de abstracción que facilite la creación de aplicaciones de gestión, ocultando detalles de implementación y comunicación.

Arquitectura

Así pues, la arquitectura global que se propone es la siguiente:

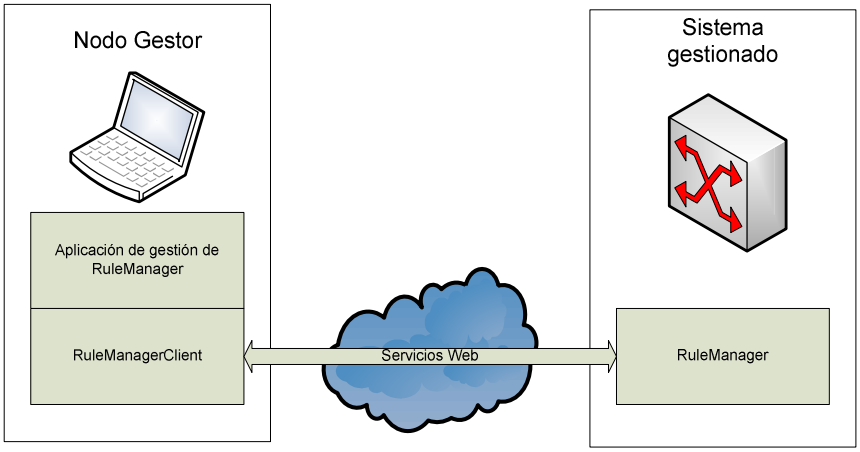

Figura 15. Arquitectura global

Interfaz

La biblioteca *RuleManagerClient*, expondrá una interfaz Java a las aplicaciones de control que les permita interactuar con *RuleManager*. Dicha interfaz contendrá todos los métodos que expone *RuleManager* mediante servicios web, ver Definición de la interfaz Web Service de *RuleManager*.

Además, contiene un conjunto de métodos específicos para la gestión de notificaciones. Para gestionar tales notificaciones, *RuleManagerClient* utiliza un modelo de subscripción, mediante el cual las aplicaciones pueden expresar su deseo de recibir o no tales notificaciones.

- *addListener(RuleManagerEventListener listener)*
- *deleteListener(RuleManagerEventListener listener)*

Un objeto que implemente la interfaz *RuleManagerEventListener*, podrá ser usado para recibir las notificaciones que envíe el gestor. Así pues con estas dos operaciones, se puede gestionar la lista de objetos que recibirán notificación.

La interfaz *RuleManagerEventListener* que deben implementar los objetos que deseen recibir notificaciones del gestor, consta de la siguiente operación:

- *notify(String message)*, esta operación será llamada por *RuleManagerClient* cuando se reciba alguna notificación por parte del gestor de reglas.

Notación formal de la interfaz de notificación

A continuación se detalla formalmente el mecanismo de subscripción de notificaciones.

- *notify(String message)*
  - Mensaje de petición

Nombre de parte	Tipo	Opcional	Descripción
message	String	No	Mensaje que contiene el texto de la notificación

  - Mensaje de respuesta

Nombre de parte	Tipo	Opcional	Descripción
-	-	-	-

- *addListener(RuleManagerEventListener listener)*
  - Mensaje de petición

Nombre de parte	Tipo	Opcional	Descripción
listener	RuleManagerEventListener	No	Objeto que implemente RuleManagerEventListener, interesado en recibir notificaciones

  - Mensaje de respuesta

Nombre de parte	Tipo	Opcional	Descripción
-	-	-	-

- *deleteListener(RuleManagerEventListener listener)*
  - Mensaje de petición

Nombre de parte	Tipo	Opcional	Descripción

listener	RuleManagerEventListener	No	Objeto que implemente RuleManagerEventListener que desee dejar de recibir notificaciones

- Mensaje de respuesta

Nombre de parte	Tipo	Opcional	Descripción
-	-	-	-

5.5.2 Implementación

Las tecnologías y metodologías utilizadas en la implementación de la biblioteca de cliente del Gestor de Reglas son análogas a las utilizadas en la implementación de éste, destacando de nuevo la utilización del *framework Apache CXF* para el desarrollo de enpoints y clientes de servicios web.

Clases

La implementación de *RuleManagerClient* se ha estructurado en varios paquetes, los cuales se van a ir describiendo a continuación.

Paquete *rmclient*

Este paquete contiene la clase que expone los principales métodos que utilizarán las aplicaciones de gestión para llevar a cabo la comunicación con el Gestor de Reglas. Más concretamente se encarga de gestionar las siguientes funciones:

- comunicación con *RuleManager* a través de una interfaz basada en *Web Services* (cliente de *web services*)
  - La biblioteca expone a las aplicaciones de nivel superior los mismo métodos que RuleManager, de forma que para ellas el Gestor de Reglas se confunde con un objeto local
- gestión del mecanismo de notificaciones

Se compone de las siguientes clases:

- Clase *RuleManagerClient*

  Dicha clase implementa la interfaz descrita en el apartado 5.5.1.

Paquete *event*

En este paquete se incluyen las clases necesarias para gestionar el mecanismo de subscripción a las notificaciones del Gestor de Reglas.

Contiene las siguientes clases:

- Interfaz *Listener*

  Interfaz *web service* que expone el método necesario (*notify(String message)*) para recibir las notificaciones del Gestor de Reglas.

- Clase *ListenerImpl*

  Implementación de la interfaz anterior, la cual se encarga de procesar adecuadamente las notificaciones recibidas, haciendo uso de *NotificationManager*.

- Clase *NotificationManager*

  Clase auxiliar que mantiene la lista de objetos que implementan *RuleManagerEventListener*, y que gestiona la notificación a todos ellos cuando se produce la llegada de un nuevo evento desde *RuleManager*.

- Interfaz *RuleManagerEventListener*

  Esta es la interfaz que deben implementar los objetos que deseen recibir notificaciones provenientes de RuleManager a través de la biblioteca cliente

5.6 Aplicaciones de Gestión del Escenario de Referencia

A continuación se describe el desarrollo de dos aplicaciones de gestión del escenario de referencia que se introdujo más arriba. Ambas aplicaciones están pensadas como una herramienta para el operario del sistema, que le permitan actuar y conocer en todo momento el estado del mismo.

Dichas aplicaciones se encargan de mostrar las acciones que se pueden realizar en cada estado (monitorizaciones y acciones de configuración), además de mostrar y procesar las notificaciones que se reciben del Gestor de Reglas.

La primera de estas aplicaciones se ha diseñado como una aplicación de escritorio Java, mientras que la segunda se concibe como una aplicación web, accesible mediante cualquier navegador.

5.6.1 Aplicación Java

La aplicación de control que se desea diseñar debe tener la capacidad para gestionar el escenario de comunicación de la Figura 12. Por tanto, tendrá la capacidad de enviar

las reglas XML necesarias para la definición de dicho escenario, además de permitir al usuario interactuar con el Gestor de Reglas para realizar acciones a su voluntad, como pueden ser activar o desactivar interfaces, aplicar filtros de tráfico, etc.

Entrando en detalles más cercanos al diseño y a la implementación de dicha aplicación, se debe destacar que ésta se apoya en la biblioteca de cliente del Gestor de Reglas, *RuleManagerClient*. Así pues, mediante esta biblioteca, la aplicación de control delega las operaciones de gestión de sesión, envío de reglas y recepción de notificaciones. De esta forma, la aplicación deberá encargarse de gestionar las interacciones del usuario frente a la interfaz gráfica (tratamiento de botones, etc.) y de procesar las notificaciones recibidas, prestando especial atención a cambios de estado del sistema, o eventos de monitorización.

El diagrama de clases de dicha aplicación es el siguiente:

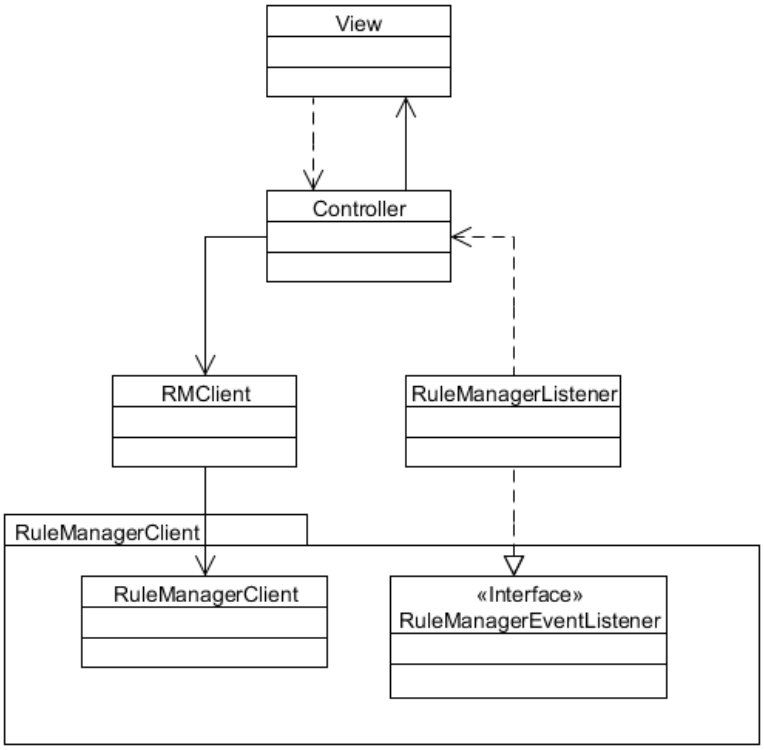

Figura 16. Aplicación de control Java - Diagrama de clases

Dicha aplicación se ha diseñado siguiendo el patrón de diseño MVC (Modelo-Vista-Controlador). Como se puede ver, el elemento central de la aplicación es la clase controlador, la cual se encarga de recoger los eventos generados por el usuario (clicks sobre la interfaz gráfica), y a través de una clase auxiliar (RMClient) utilizar la biblioteca *RuleManagerClient* para realizar llamadas a *RuleManager*. La parte asíncrona de la aplicación sigue el flujo contrario, es decir, mediante RuleManagerListener (que implementa la interfaz RuleManagerEventListener) se reciben notificaciones

provenientes de RuleManager, las cuales se procesan adecuadamente, produciendo los cambios en la vista que sean necesarios.

A continuación se presentan algunas capturas de dicha aplicación:

Figura 17. Aplicación de control Java - Estado en servicio

Figura 18. Aplicación de control Java - Monitorización de energía

5.6.2 Aplicación Web

La aplicación de gestión basada en Web tiene como objetivo el ser capaz de ofrecer las mismas funcionalidades que su homóloga desarrollada en Java, proporcionando las ventajas inherentes de las aplicaciones proporcionadas a través de la Web, como son la independencia de plataforma, coste nulo de despliegue en clientes, ejecución desde terminales con bajas prestaciones, etc.

En general, la mayor parte de operaciones que debe soportar esta interfaz son acciones iniciadas por el usuario de la misma, como por ejemplo activar y desactivar interfaces, cambiar la prioridad de cada interfaz, etc. Este tipo de interacciones, responde claramente a un modelo cliente-servidor, por lo que el protocolo HTTP encaja sin problemas en esta arquitectura.

Sin embargo, un aspecto crucial en el funcionamiento del Gestor de Reglas es la monitorización y la gestión de notificaciones. Así pues, la interfaz web debe ser capaz de reaccionar ante distintos eventos asíncronos que provienen del gestor de reglas. Un claro ejemplo de lo que se está intentando exponer podría ser el siguiente: el usuario comienza a monitorizar el ancho de banda de una interfaz, lo cual se traduce en una petición HTTP, que desemboca en el envío de reglas XML al Gestor de Reglas. Una vez aplicadas estas reglas, se produce una bajada del ancho de banda disponible, por lo que el gestor de reglas procederá a la notificación de tal evento. En este momento, es imprescindible que el usuario final, se percate de esta situación de forma asíncrona, sin tener que realizar ninguna operación específica sobre la aplicación web.

El desarrollo de esta aplicación Web está basado en HTTPServlets de Java, desplegados sobre un servidor de aplicaciones Tomcat 6. El diagrama de clases propuesto es el siguiente:

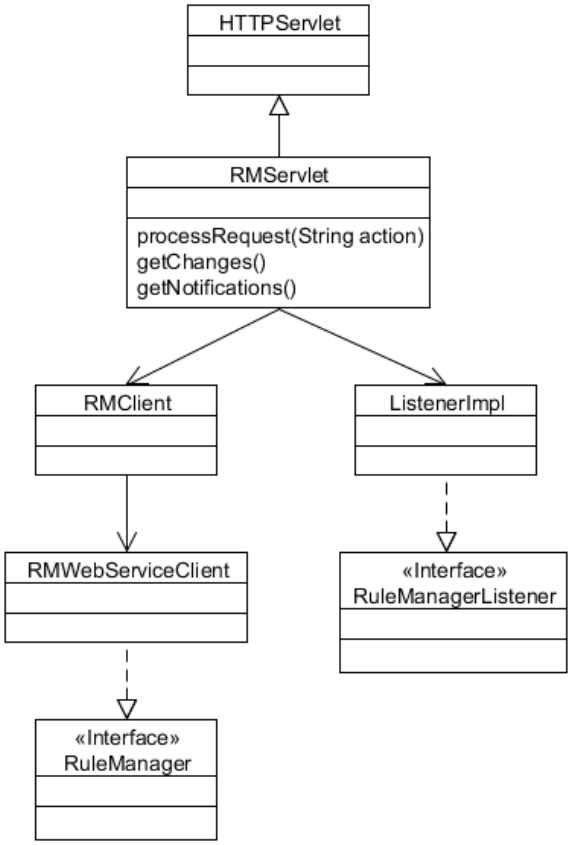

Figura 19. Aplicación de control basada en Web - Diagrama de clases

Como se puede ver en la figura anterior, el elemento central de la interfaz es el HTTPServlet llamado RMServlet, el cual se encarga de procesar las peticiones HTTP que el cliente web invoca. Este HTTPServlet, utiliza un conjunto de clases auxiliares encargadas de gestionar la comunicación con el gestor de reglas. Por un lado, las clases RMClient y RMWebServiceClient, se encargan de consumir los métodos *web service* que el gestor de reglas expone. Por otro lado, la clase ListenerImpl tiene como función la recepción y el procesamiento de las notificaciones que el gestor de reglas emite.

Además, se han definidos distintos modelos de interacción, los cuales se presentan en el siguiente diagrama de secuencia:

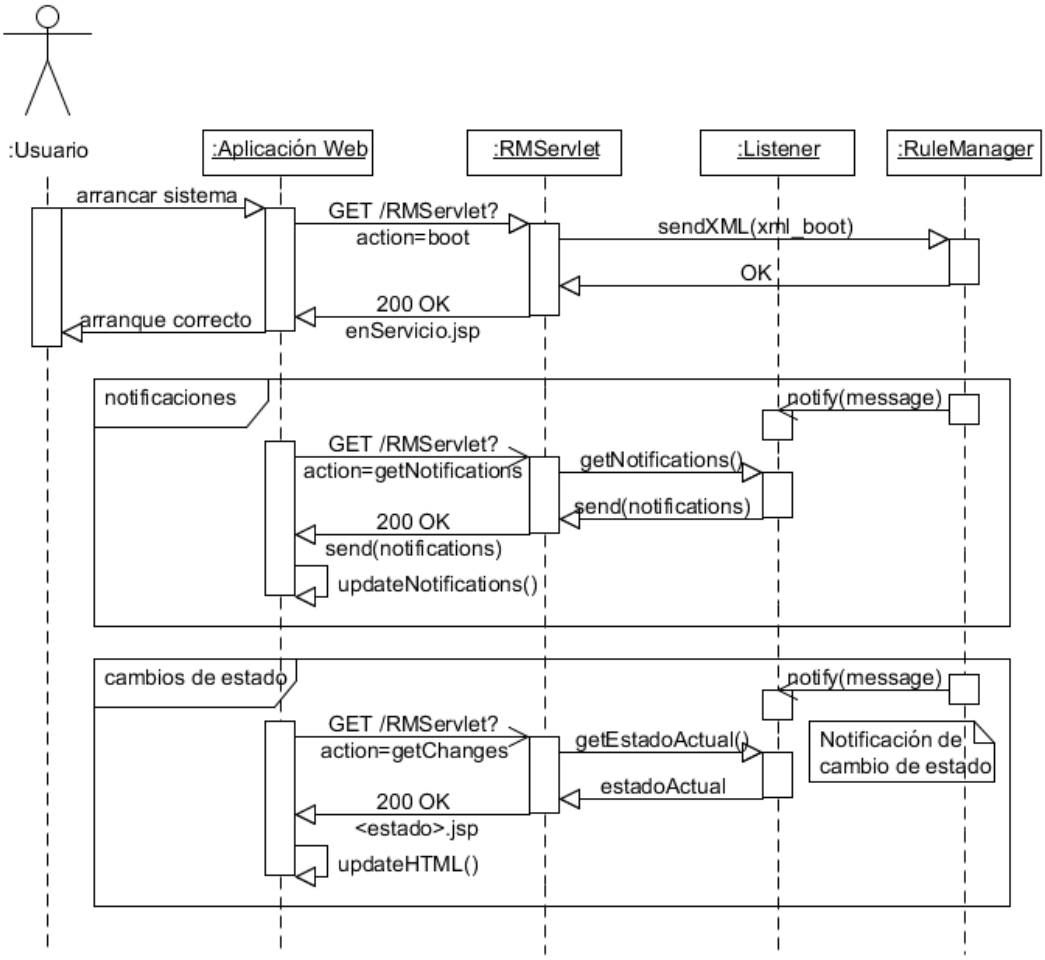

Figura 20. Aplicación de control basada en Web - Diagrama de secuencia

En primer lugar se puede ver una típica interacción en la que el usuario desea llevar a cabo alguna operación, en este caso, arrancar el sistema de comunicaciones gestionado por *RuleManager*. Para ello, el usuario hará *click* sobre un enlace en su navegador, el cual provocará una petición GET que será atendida por el *servlet* RMServlet. Una vez que RMServlet haya identificado el parámetro en el que se indica qué operación se debe realizar (*boot*, en el caso del ejemplo), se realizará la petición pertinente sobre el gestor de reglas. Por último, una vez que la operación se haya completado, la respuesta será enviada de vuelta al navegador del cliente.

Las otras dos interacciones que se presentan, la actualización de notificaciones y cambios de estado, no se originan como consecuencia de una acción llevada a cabo por el usuario, sino que se producen de forma periódica mediante una función *jQuery*, la cual hace una petición GET al servidor con el objetivo de obtener las notificaciones y

cambios de estado que hayan podido producirse. Esta petición será procesada por RMServlet, que devolverá la respuesta al navegador de cliente, el cual, mediante la ejecución de funciones *jQuery*, actualizará el campo de texto de notificaciones, o los fragmentos HTML necesarios para representar el cambio de estado, todo ello de forma transparente para el usuario final.

A continuación se incluyen algunas capturas de dicha aplicación web:

Figura 21. Aplicación de control basada en Web - Estado apagado

Figura 22. Aplicación de control basada en Web - Estado en servicio

6 Pruebas y validación

6.1 Introducción

En el presente capítulo se describe la siguiente fase del ciclo de vida del Gestor de Reglas y de las aplicaciones de control desarrolladas en este trabajo, la fase de pruebas y validación.

Aquí se presentan de forma resumida un conjunto de pruebas realizadas para la validación del Gestor de Reglas y las aplicaciones de control en conjunto, organizadas en torno a pequeños conjuntos de pruebas basados en los casos de uso que se contemplaron en el capítulo 4. Más formalmente, este conjunto de pruebas se puede ver como una serie de pruebas de integración funcionales, las cuales se basan en el escenario de referencia que se propuso en el capítulo 5.2.

Se debe destacar que aparte de la realización de este conjunto de pruebas de validación, se han ido realizando una serie de pruebas unitarias tanto de caja blanca como de caja negra, cuyo objetivo era el comprobar el funcionamiento de cada uno de los módulos en los que se descomponen las distintas aplicaciones. Debido a que estas pruebas no aportan demasiada información no se detallarán en este capítulo.

6.2 Entorno de pruebas

Para la realización de las pruebas que se describen a continuación se han empleado los siguientes elementos:

- *Java Runtime Environment* (JRE) 1.6.0_14
- Servidor de aplicaciones *Tomcat* 6.0.26
- Paquete de reglas XML que modelen el escenario de referencia
- Navegador web *Google Chrome* 5.0.375.99

Además, para la realización de las pruebas unitarias, y para la validación de los distintos módulos de las aplicaciones desarrolladas se han utilizado los siguientes elementos:

- Analizador de protocolo *Wireshark*
- Clases Java basadas en *JUnit*

6.3 Pruebas desarrolladas

El objetivo de las pruebas que se detallan a continuación es el comprobar que todas las funcionalidades del Gestor de Reglas, y de las aplicaciones cliente asociadas se llevan a cabo correctamente, aplicando de forma adecuada las reglas y escenarios que se definan.

Tabla 11. Prueba de gestión de sesión

Orden	Estímulo	Resultado esperado	Resultado
1	Llamada a la función getIdSession(XML_session_1), con nombre y password correctos	Creación del identificador de sesión. Devuelve cadena alfanumérica de 14 caracteres. No se lanzan excepciones	OK
2 (para N=1…)	Llamada a la función sendXML(XML_correcto), con el identificador de sesión obtenido en el paso 1	Se aplican las reglas contenidas en la cadena XML. No se lanzan excepciones	OK
3 (para M=1…)	Llamada a la función sendXML(XML_correcto), con identificador de sesión erróneo	No se aplican las reglas contenidas en la cadena XML. Se lanza IncorrectSessionIDException	OK
4	Llamada a la función deleteIdSession(XML_session_1), con identificador de sesión correcto	Se libera la sesión. No se lanzan excepciones	OK
5 (para P=1…)	Llamada a la función sendXML(XML_correcto), con el identificador de sesión obtenido en el paso 1	No se aplican las reglas contenidas en la cadena XML. Se lanza IncorrectSessionIDException	OK
6	Llamada a la función getIdSession(XML_session_2), con nombre y password incorrectos	No se crea identificador de sesión. Se lanza IncorrectLoginException	OK

Con este conjunto de pruebas se pone de manifiesto el correcto funcionamiento de la gestión de sesiones en el Gestor de Reglas, ya que se cumplen las siguientes condiciones:

- Se crea una sesión (y el identificador asociado) cuando los datos de autenticación son correctos
- Se aplican las reglas que van asociadas al identificador anterior
- No se aplican las reglas que incluyen un identificador de sesión incorrecto
- Se libera la sesión adecuadamente, quedando inutilizado el identificador de la misma
- No se crea una sesión cuando los datos de autenticación son incorrectos

Tabla 12. Pruebas de definición y aplicación de acciones aisladas

Orden	Estímulo	Resultado esperado	Resultado
1	Llamada a la función getIdSession(XML_session_2), con nombre y password correctos	Creación del identificador de sesión. Devuelve cadena alfanumérica de 14 caracteres. No se lanzan excepciones	OK
2 (para N=1…)	Llamada a la función sendXML(XML_Acciones), con el identificador de sesión obtenido en el paso 1. Se envía un conjunto de elementos <entry_action>	Se aplican las reglas contenidas en la cadena XML. No se lanzan excepciones	OK
3 (para M=1…) M≤N	Llamada a la función sendXML(XML_Exec_Actions), con el identificador de sesión obtenido en el paso 1. Se envía un conjunto de elementos <execute_entry_action> asociados a los monitores del paso 2	Se aplican las reglas contenidas en la cadena XML. No se lanzan excepciones	OK

Orden	Estímulo	Resultado esperado	Resultado
4	Llamada a la función sendXML(XML_Exec_Actions), con el identificador de sesión obtenido en el paso 1. Se envía un elemento <execute_entry_action> NO asociado a los monitores del paso 2	No se aplican las reglas contenidas en la cadena XML. Se lanza XMLException	OK
5	Se ejecuta una acción y se cumple la condición del mismo	Se aplican las acciones contenidas en <success>	OK
6	Se ejecuta una acción y no se cumple la condición del mismo	Se aplican las acciones contenidas en <failure>	OK

El objetivo de este conjunto de pruebas ha sido el analizar el correcto funcionamiento del mecanismo de definición y ejecución de acciones en instantes arbitrarios (es decir, sin estar asociadas a ninguna condición).

Tabla 13. Pruebas de definición y aplicación de elementos de monitorización

Orden	Estímulo	Resultado esperado	Resultado
1	Llamada a la función getIdSession(XML_session_3), con nombre y password correctos	Creación del identificador de sesión. Devuelve cadena alfanumérica de 14 caracteres. No se lanzan excepciones	OK
2 (para N=1…)	Llamada a la función sendXML(XML_Monitores), con el identificador de sesión obtenido en el paso 1. Se envía un conjunto de elementos <entry_monitor>	Se aplican las reglas contenidas en la cadena XML. No se lanzan excepciones	OK

Orden	Estímulo	Resultado esperado	Resultado
3 (para M=1…) M≤N	Llamada a la función sendXML(XML_Exec_Monitors), con el identificador de sesión obtenido en el paso 1. Se envía un conjunto de elementos <execute_entry_monitor> asociados a los monitores del paso 2	Se aplican las reglas contenidas en la cadena XML. Se crean threads de ejecución que lanzan los monitores en el intervalo de tiempo indicado. No se lanzan excepciones	OK
4	Llamada a la función sendXML(XML_Exec_Monitors), con el identificador de sesión obtenido en el paso 1. Se envía un elemento <execute_entry_monitor> NO asociado a los monitores del paso 2	No se aplican las reglas contenidas en la cadena XML. Se lanza XMLException	OK
5	Se ejecuta un monitor y se cumple la condición del mismo	Se aplican las acciones contenidas en <success>	OK
6	Se ejecuta un monitor y no se cumple la condición del mismo	Se aplican las acciones contenidas en <failure>	OK

Gracias a este conjunto de pruebas, se ha comprobado el adecuado funcionamiento del mecanismo de monitorización del Gestor de Reglas. Concretamente se han verificado los siguientes puntos:

- Definición de elementos de monitorización
- Gestión de errores en la definición y ejecución de elementos de monitorización
- Aplicación de elementos de monitorización
- Ejecución de acciones en función del resultado de la monitorización
    - Elementos <success>
    - Elementos <failure>

Tabla 14. Pruebas de gestión de estados

Orden	Estímulo	Resultado esperado	Resultado
1	Llamada a la función getIdSession(XML_session_4), con nombre y password correctos	Creación del identificador de sesión. Devuelve cadena alfanumérica de 14 caracteres. No se lanzan excepciones	OK
2 (para N=1…)	Llamada a la función sendXML(XML_States), con el identificador de sesión obtenido en el paso 1. Se envía un conjunto de elementos <entry_state> con elementos <entry_action> asociados	Se aplican las reglas contenidas en la cadena XML. No se lanzan excepciones	OK
3	Llamada a la función sendXML(XML_Exec_state), con el identificador de sesión obtenido en el paso 1. Se indica la transición a uno de los estados definidos en el paso 2	Se establece como estado actual el estado indicado en la cadena XML. No se lanzan excepciones	OK
4 (para M=1…) M≤N	Llamada a la función sendXML(XML_Exec_Actions), con el identificador de sesión obtenido en el paso 1. Se envía un conjunto de elementos <execute_entry_actions> asociados al estado del paso 2	Se aplican las reglas contenidas en la cadena XML. No se lanzan excepciones	OK
5	Llamada a la función sendXML(XML_Exec_Actions), con el identificador de sesión obtenido en el paso 1. Se envía un conjunto de elementos <execute_entry_actions> NO asociados al estado del paso 2	No se aplican las reglas contenidas en la cadena XML. Se lanza XMLException	OK

Orden	Estímulo	Resultado esperado	Resultado
6	Llamada a la función sendXML(XML_Exec_state), con el identificador de sesión obtenido en el paso 1. Se indica la transición a uno de los estados definidos en el paso 2	Se establece como estado actual el estado indicado en la cadena XML. No se lanzan excepciones	OK
7 (para M=1…) M≤N	Llamada a la función sendXML(XML_Exec_Actions), con el identificador de sesión obtenido en el paso 1. Se envía un conjunto de elementos <execute_entry_actions> asociados al estado del paso 6	Se aplican las reglas contenidas en la cadena XML. No se lanzan excepciones	OK
8	Llamada a la función sendXML(XML_Exec_Actions), con el identificador de sesión obtenido en el paso 1. Se envía un conjunto de elementos <execute_entry_actions> NO asociados al estado del paso 6	No se aplican las reglas contenidas en la cadena XML. Se lanza XMLException	OK

En este conjunto de pruebas se han verificado los siguientes aspectos:

- Definición de estados
- Asociación de acciones a estados
- Ejecución de acciones del estado actual
- Ejecución de acciones de un estado diferente al actual
- Transiciones entre estados

Tabla 15. Pruebas del mecanismo de notificaciones a clientes

Orden	Estímulo	Resultado esperado	Resultado
1	Llamada a la función getIdSession(XML_session_5), con nombre y password correctos	Creación del identificador de sesión. Devuelve cadena alfanumérica de 14 caracteres. No se lanzan excepciones	OK
2 (para N=1…)	Llamada a la función sendXML(XML_States), con el identificador de sesión obtenido en el paso 1. Se envía un conjunto de elementos <entry_state> con elementos <entry_action> asociados	Se aplican las reglas contenidas en la cadena XML. No se lanzan excepciones	OK
3	Llamada a la función sendXML(XML_Exec_state), con el identificador de sesión obtenido en el paso 1. Se indica la transición a uno de los estados definidos en el paso 2	Se establece como estado actual el estado indicado en la cadena XML. No se lanzan excepciones	OK
4 (para M=1…) M≤N	Llamada a la función sendXML(XML_Exec_Actions), con el identificador de sesión obtenido en el paso 1. Se envía un conjunto de elementos <execute_entry_actions> asociados al estado del paso 2. Dichos elementos incluyen el parámetro notification=yes	Se aplican las reglas contenidas en la cadena XML. Se envían notificaciones a cliente No se lanzan excepciones	OK
5 (para M=1…) M≤N	La aplicación cliente recibe las notificaciones asociadas a las acciones del paso 4	Se reciben las notificaciones en el endpoint del cliente	OK
6	El cliente procesa las notificaciones	Se muestran las notificaciones en la interfaz del cliente	OK

En este conjunto de pruebas se ha verificado el mecanismo de notificaciones a cliente, el cual permite indicar sobre qué acciones desea la aplicación ser informada.

Tabla 16. Pruebas del mecanismo de persistencia

Orden	Estímulo	Resultado esperado	Resultado
1	Se configura el Gestor de Reglas con la opción persistence=true	Traza indicativa al arranque del Gestor. No se lanzan excepciones	OK
2	Llamada a la función getIdSession(XML_session_6), con nombre y password correctos	Creación del identificador de sesión. Devuelve cadena alfanumérica de 14 caracteres. No se lanzan excepciones	OK
3 (para N=1…)	Llamada a la función sendXML(XML_States), con el identificador de sesión obtenido en el paso 1. Se envían distintos tipos de elementos	Se aplican las reglas contenidas en la cadena XML. Se almacenan en disco las reglas recibidas. No se lanzan excepciones	OK
4	El Gestor de Reglas se cierra de forma abrupta	Las aplicaciones cliente no son capaces de comunicarse con el Gestor de Reglas	OK
5	Se vuelve a iniciar el Gestor de Reglas	Se restaura el estado anterior, volviendo a cargar los elementos que se almacenaron en disco	OK
6 (para M=1…)	La aplicación cliente interactúa con el Gestor de Reglas	La aplicación cliente puede interactuar con el Gestor de Reglas, de la misma forma que en el instante antes del "fallo" del sistema	OK

En este último conjunto de pruebas se ha testeado el funcionamiento del mecanismo de recuperación ante fallos del Gestor de Reglas. Como se puede ver, el Gestor de Reglas es capaz de almacenar de forma persistente su estado, pudiendo restaurarlo tras sufrir un fallo.

7 Conclusiones y líneas de trabajo futuras

7.1 Conclusiones

7.1.1 Entorno y tecnologías de desarrollo

Cada vez más, los sectores profesionales están demandando comunicaciones móviles más avanzadas, que les permitan disfrutar de un acceso a la red de calidad, en base a un conjunto de políticas o reglas de alto nivel. Con el objetivo de cubrir dicha necesidad, en este trabajo se ha desarrollado un Gestor de Reglas que permita gestionar la conectividad ofrecida por un router móvil. En este proceso de desarrollo se han utilizado un conjunto amplio de tecnologías del mundo Java y del mundo de la Web.

En cuanto a la tecnología Java utilizada, se debe destacar en primer lugar la ventaja intrínseca que su uso supone, es decir, la independencia de plataforma, haciendo al Gestor de Reglas una aplicación que puede ser utilizada sobre multitud de dispositivos (aquellos capaces de cargar un JRE). Concretando un poco más, la utilización del framework Apache CXF, ha resultado sorprendentemente cómoda. Dicho framework no era conocido por nosotros, ya que, en otros desarrollos, siempre se optó por utilizar tecnología JAX-WS sobre servidores de aplicaciones (GlassFish – SailFin, Tomcat, etc.). Sin embargo, CXF facilita enormemente un requisito fundamental del Gestor de Reglas: que sea una aplicación independiente y ligera, es decir, que no tenga que ser desplegada sobre un servidor de aplicaciones.

Otro ámbito donde la tecnología Java ha sido crucial es en el análisis y el procesamiento de texto XML. Para ello, como se comentó más arriba, se eligió la tecnología Apache Xerces-J. Xerces-J es un parser XML ampliamente utilizado en la industria, el cual soporta los modos típicos de análisis XML, como son SAX y DOM. Sin embargo, esta no es la característica más destacada para este trabajo. Como se dijo, las reglas que se manejan en RuleManager, vienen definidas mediante una sintaxis XML, especificada en un XSD. Xerces-J ofrece la ventaja de contrastar automáticamente las reglas que el cliente envía frente a este XML Schema Definition, detectando los posibles fallos que existan. Dicha funcionalidad evita engorrosas tareas de comprobación de sintaxis, haciendo que las rutinas de proceso del XML presupongan la corrección del mismo.

En cuanto a las tecnologías Web empleadas, destacar la comodidad del desarrollo basado en HTTPServlets y JSP, las cuales permiten un rápido desarrollo de la parte de controlador y vista, respectivamente, de una aplicación web. Otra tecnología crucial

para el desarrollo de la aplicación de control basada en Web ha sido jQuery. jQuery es una librería que facilita enormemente la utilización de Javascript. Nadie duda de la potencia de Javascript en el lado de cliente, sin embargo, su utilización en aplicaciones complejas puede resultar un tanto laboriosa. Es aquí donde jQuery aporta grandes ventajas, sobre todo en la manipulación de elementos del DOM HTML y en la gestión de las llamadas asíncronas al servidor (AJAX).

7.1.2 Desarrollo del trabajo

El Gestor de Reglas (RuleManager), la biblioteca de cliente (RuleManagerClient) y las aplicaciones de control diseñadas han cumplido los requisitos que se les pedían. Así pues, dichas aplicaciones están en condiciones de ser desplegadas sobre un escenario real, por ejemplo una red móvil basada en un router Cisco 2811 con un módulo AXP.

Como se ha podido ver a lo largo del trabajo, el Gestor de Reglas que se ha desarrollado es capaz de analizar un conjunto de reglas, expresadas mediante texto XML, y aplicarlas sobre una determinada plataforma, llevando a cabo operaciones de monitorización y control. Se debe destacar, que dicho módulo no se ha ceñido a la gestión de un tipo determinado de plataforma, sino que en su diseño se ha intentado dar cabida a múltiples de ellas (mediante la utilización de scripts genéricos como interfaz hacia las plataformas gestionadas).

La sintaxis XML que se ha definido se ha basado en la definición de máquinas de estados que permitan asociar conjuntos distintos de reglas en función del estado del sistema. Además, se ha intentado que dicha sintaxis XML sea lo suficientemente flexible como para definir escenarios de control complejos.

En cuanto a la biblioteca de cliente, se ha comprobado las ventajas que aporta en la tarea de desarrollar aplicaciones de cliente, resolviendo el problema de la comunicación basada en servicios web con el Gestor de Reglas. El hecho de que dicha comunicación se realice mediante servicios web también ha resultado ser ventajosa, ya que permite el desarrollo de aplicaciones de control tanto de tipo aplicación de escritorio como de tipo aplicación web.

En esta línea se han desarrollado dos aplicaciones de control, con el objetivo de poner de manifiesto la utilidad de la biblioteca de cliente desarrollada. La primera de ellas, una aplicación de escritorio desarrollada en Java; y la segunda, una aplicación web. Se considera especialmente relevante esta última, por las ventajas inherentes de toda aplicación web, como son la independencia de plataforma y los costes nulos de despliegue en el terminal de cliente.

Además, se considera que los módulos que se han desarrollado en este trabajo, son lo suficientemente flexibles y reutilizables como para ser aplicados en multitud de ámbitos en los que se necesite llevar a cabo tareas de monitorización y control.

Por último, se debe destacar que este trabajo ha formado parte del proyecto PMI-IP (Prototipo de Movilidad e Interoperabilidad IP) dentro de la Línea Instrumental de Proyectos de I+D+i, encuadrada en el Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica 2008-2011, donde se ha puesto de manifiesto su utilidad y aplicación práctica en un entorno real.

7.2 Líneas futuras de trabajo

A continuación se exponen algunas posibles líneas de trabajo futuro con las que ofrecer una continuidad a los desarrollos de este trabajo fin de máster.

7.2.1 Gestión de la seguridad

En el Gestor de Reglas de este trabajo se ha implementado un mecanismo de autenticación que evita a aplicaciones de control no autorizadas el interactuar con RuleManager. Sin embargo, existen otros aspectos asociados a la gestión de la seguridad que no están completamente resueltos en este trabajo.

Así pues, se propone la utilización de *Web Services - Security* con el objetivo de proporcionar un mayor nivel de seguridad en las comunicaciones entre el Gestor de Reglas y las aplicaciones clientes.

WS-S [28] es una extensión al protocolo SOAP para ofrecer mecanismos de seguridad en entornos de servicios web. Más concretamente, ofrece la capacidad de firmar mensajes SOAP, para asegurar la integridad y el no-repudio; cifrado de mensajes SOAP, para asegurar la confidencialidad; y un mecanismo de asociación de tokens de seguridad.

Una alternativa a WS-S, se puede basar en explotar la capacidad que ofrece SOAP de permitir distintos protocolos de transporte. Así pues, se podría pensar en utilizar HTTPS en vez de HTTP como protocolo de envío de mensajes SOAP. Esta solución ofrece seguridad en el envío de mensajes, con una mayor eficiencia que WS-S, sin embargo, presenta el problema de no ser válida cuando existen proxys a nivel de aplicación, ya que en este caso, es posible que existan enlaces de comunicación donde no se utilice HTTPS. Es aquí donde WS-S, con su capacidad de seguridad extremo a extremo, presenta mayores ventajas.

7.2.2 Desarrollo de aplicación de creación de escenarios

El desarrollo de una aplicación de creación de escenarios sería un complemento ideal para el Gestor de Reglas de este trabajo. Dicha aplicación ofrecería una interfaz

gráfica con la que poder diseñar máquinas de estados, asociando acciones y monitores a los mismos.

Una vez diseñados los escenarios, esta aplicación será capaz de generar las reglas XML que modelan los mismos. Así pues, el usuario tendrá un mecanismo que le facilitará la creación de escenarios complejos, evitando el tener que escribir complejas reglas mediante texto marcado en XML.

7.2.3 Desarrollo de aplicación generadora de clientes

Otro posible desarrollo muy interesante, íntimamente relacionado con el anterior, es la creación de una herramienta capaz de generar automáticamente aplicaciones cliente de gestión.

Una vez que se han generado los XML que modelan un determinado escenario, sería interesante poder automatizar la creación de aplicaciones que los gestionen mediante el análisis de estos escenarios.

Estas aplicaciones tienen muchos aspectos comunes, los cuales permiten la automatización de su creación. En general, dichas aplicaciones se componen de distintas vistas, una por cada estado, conteniendo estas los controles necesarios para la gestión de las acciones asociadas a cada uno de ellos. Así pues, se podría automatizar la creación de la base del código, liberando al programador de tareas repetitivas en cada una de las aplicaciones de gestión, orientando los esfuerzos a la personalización de cada una.

Bibliografía

[1] Akyildiz, I.F.; Xudong W.; , "A survey on wireless mesh networks," *Communications Magazine, IEEE* , vol.43, no.9, pp. S23- S30, Sept. 2005

[2] Perkins, C.; Belding-Royer, E.; Das, S; , "Ad hoc On-Demand Distance Vector (AODV) Routing," *IETF RFC 3561*, July 2003

[3] Open-Mesh.net, "B.A.T.M.A.N. (better approach to mobile ad-hoc networking)," [Online]. Available: http://www.open-mesh.net/ [Accesed: Jun 22, 2010]

[4] Tropos Networks, "Metro-Scale Mesh Networking with Tropos MetroMesh Architecture," *A Technology Whitepaper*. July, 2007 [Online]. Available: http://www.tropos.com/pdf/whitepapers/tropos_whitepaper_metro-scale.pdf [Accesed: Jun 22, 2010]

[5] Clausen, T.; Jacquet, P.; "Optimized Link State Routing Protocol (OLSR)," *IETF RFC 3626*, October 2003

[6] IEEE Standard 802.16 Working Group, "IEEE standard for local and metropolitan area networks (revision of IEEE standard 802.16-2001)," 2004

[7] 3GPP. Third Generation Parnership Project (3GPP). [Online]. Available: http://www.3gpp.org [Accesed: Jun 23, 2010]

[8] 3GPP, 3GPP Release 99. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/FeatureList--R99.htm [Accesed: Jun 23, 2010]

[9] IETF. The Internet Engineering Task Force. [Online]. Available: http://www.ietf.org [Accesed: Jun 23, 2010]

[10] Perkins, C.; "IP Mobility Support for IPv4," *IETF RFC 3220*, January 2002.

[11] Johnson, D.; Perkins, C.; Arkko, J.;, "Mobility Support in IPv6," *IETF RFC 3775*, June 2004

[12] Devarapalli, V.; Wakikawa, R.; Petrescu, A.; Thubert, P.; "Network Mobility (NEMO) Basic Support Protocol, " *IETF RFC 3963*, January 2005

[13] W3C, "Web Services Glossary", February 2004

[14] W3C, "Web Services Architecture", February 2004

[15] W3C, "SOAP Version 1.2 Part 0: Primer", Second Edition, April 2007

[16] W3C, "SOAP Version 1.2 Part 1: Messaging Framework", June 2003

[17] W3C, "Web Services Description Language (WSDL) 1.1", March 2001

[18] JSR (Java Specification Requests) 224, "The Java API for XML-Based Web Services (JAX-WS) 2.0, Final Release", Abril 2006

[19] The Apache Software Foundation, "Apache CXF", [Online]. Available: http://cxf.apache.org/ [Accesed: Jun 25, 2010]

[20] JSR (Java Specification Request) 154, "Java Servlet Specification Version 2.5 MR6", July 2007

[21] JSR (Java Specification Request) 245, "JavaServer Pages Specification Version 2.0", November 2003

[22] JSR (Java Specification Request) 94, "Java Rule Engine API", September 2002

[23] Sandia National Laboratories, "Jess, the Rule Engine for the Java Platform", [Online]. Available: http://www.jessrules.com/jess/index.shtml [Accesed: Jun 28, 2010]

[24] Forgy, C.;, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," *Artificial Intelligence*, Volume 19, Issue 1, pp 17-37, September 1982

[25] JBoss, "Drools Expert User Guide", [Online]. Available: http://downloads.jboss.com/drools/docs/5.0.1.26597.FINAL/drools-expert/html/index.html [Accesed: Jun 28, 2010]

[26] The jQuery Project, "jQuery: write less, do more", [Online]. Available: http://jquery.com/ [Accesed: Jun 28, 2010]

[27] Cisco, "Application eXtension Platform", [Online]. Available: http://developer.cisco.com/web/axp/home [Accesed: Jun 28, 2010]

[28] OASIS, "OASIS Web Services Security (WSS) TC", [Online]. Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss [Accesed: Jul 10, 2010]

Apéndice A: XML Schema de definición de reglas

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://xml.netbeans.org/schema/PMI-IP"
targetNamespace="http://xml.netbeans.org/schema/PMI-IP"
elementFormDefault="qualified">
  <!--Tipos Complejos-->
  <xsd:complexType name="app">
    <xsd:choice maxOccurs="unbounded">
      <xsd:element name="execute_entry_action"
                   type="tns:execute_entry_action_type"/>
      <xsd:element name="execute_entry_monitor"
                   type="tns:execute_entry_monitor_type"/>
      <xsd:element name="execute_state"
                   type="tns:execute_state_type"/>
      <xsd:element name="entry_state" type="tns:entry_state_type"/>
      <xsd:element name="entry_monitor"
                   type="tns:entry_monitor_type"/>
    </xsd:choice>
    <xsd:attribute name="idsession" type="xsd:string"/>
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="password" type="xsd:string"/>
    <xsd:attribute name="wsdl" type="xsd:string"/>
    <xsd:attribute name="action" type="xsd:string" use="optional"/>
  </xsd:complexType>
  <xsd:complexType name="file_type">
    <xsd:choice minOccurs="0" maxOccurs="2">
      <xsd:element name="success" type="tns:success_type"/>
      <xsd:element name="failure" type="tns:failure_type"/>
    </xsd:choice>
    <xsd:attribute name="ref" type="xsd:string" use="required"/>
    <xsd:attribute name="parameters" type="xsd:string"/>
    <xsd:attribute name="success_message" type="xsd:string"/>
    <xsd:attribute name="timer" type="xsd:string"/>
    <xsd:attribute name="expected_result" type="xsd:string"
                   use="required"/>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="id" type="xsd:string" use="required"/>
    <xsd:attribute name="notification" type="xsd:string"
                   default="yes"/>
    <xsd:attribute name="error_message" type="xsd:string"/>
    <xsd:attribute name="scope" type="xsd:string"/>
  </xsd:complexType>
  <xsd:complexType name="success_type">
    <xsd:choice maxOccurs="unbounded">
      <xsd:element name="execute_entry_action"
                   type="tns:execute_entry_action_type"/>
      <xsd:element name="execute_entry_monitor"
                   type="tns:execute_entry_monitor_type"/>
      <xsd:element name="execute_state"
                   type="tns:execute_state_type"/>
      <xsd:element name="file" type="tns:file_type"/>
    </xsd:choice>
  </xsd:complexType>
  <xsd:complexType name="failure_type">
    <xsd:choice maxOccurs="unbounded">
      <xsd:element name="execute_entry_action"
```

```
                type="tns:execute_entry_action_type"/>
    <xsd:element name="execute_entry_monitor"
                type="tns:execute_entry_monitor_type"/>
    <xsd:element name="execute_state"
                type="tns:execute_state_type"/>
    <xsd:element name="file" type="tns:file_type"/>
  </xsd:choice>
</xsd:complexType>
<xsd:complexType name="entry_action_type">
  <xsd:choice maxOccurs="unbounded">
    <xsd:element name="file" type="tns:file_type"/>
    <xsd:element name="execute_entry_action"
                type="tns:execute_entry_action_type"/>
  </xsd:choice>
  <xsd:attribute name="id" type="xsd:string" use="required"/>
  <xsd:attribute name="name" type="xsd:string" use="required"/>
  <xsd:attribute name="success_message" type="xsd:string"/>
  <xsd:attribute name="error_message" type="xsd:string"/>
  <xsd:attribute name="notification" type="xsd:string"
                use="optional" default="yes"/>
</xsd:complexType>
<xsd:complexType name="execute_entry_action_type">
  <xsd:choice minOccurs="0" maxOccurs="2">
    <xsd:element name="success" type="tns:success_type"/>
    <xsd:element name="failure" type="tns:failure_type"/>
  </xsd:choice>
  <xsd:attribute name="id" type="xsd:string" use="required"/>
  <xsd:attribute name="name" type="xsd:string" use="required"/>
  <xsd:attribute name="persistence" type="xsd:string"
                use="required"/>
  <xsd:attribute name="success_message" type="xsd:string"/>
  <xsd:attribute name="error_message" type="xsd:string"/>
  <xsd:attribute name="notification" type="xsd:string"
                default="yes"/>
</xsd:complexType>
<xsd:complexType name="execute_state_type">
  <xsd:choice minOccurs="0" maxOccurs="2">
    <xsd:element name="success" type="tns:success_type"/>
    <xsd:element name="failure" type="tns:failure_type"/>
  </xsd:choice>
  <xsd:attribute name="id" type="xsd:string" use="required"/>
  <xsd:attribute name="name" type="xsd:string" use="required"/>
  <xsd:attribute name="persistence" type="xsd:string"
                use="required"/>
  <xsd:attribute name="success_message" type="xsd:string"/>
  <xsd:attribute name="error_message" type="xsd:string"/>
  <xsd:attribute name="notification" type="xsd:string"
                default="yes"/>
</xsd:complexType>
<xsd:complexType name="execute_entry_monitor_type">
  <xsd:choice minOccurs="0" maxOccurs="2">
    <xsd:element name="success" type="tns:success_type"/>
    <xsd:element name="failure" type="tns:failure_type"/>
  </xsd:choice>
  <xsd:attribute name="id" type="xsd:string" use="required"/>
  <xsd:attribute name="name" type="xsd:string" use="required"/>
  <xsd:attribute name="persistence" type="xsd:string"
                use="required"/>
  <xsd:attribute name="success_message" type="xsd:string"/>
  <xsd:attribute name="error_message" type="xsd:string"/>
  <xsd:attribute name="notification" type="xsd:string"
```

```
                default="yes"/>
  </xsd:complexType>
  <xsd:complexType name="entry_state_type">
    <xsd:choice>
      <xsd:element name="entry_action" type="tns:entry_action_type"
                maxOccurs="unbounded"/>
    </xsd:choice>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="id" type="xsd:string" use="required"/>
  </xsd:complexType>
  <xsd:complexType name="entry_monitor_type">
    <xsd:choice maxOccurs="unbounded">
      <xsd:element name="execute_entry_monitor"
                type="tns:execute_entry_monitor_type"/>
      <xsd:element name="file" type="tns:file_type"/>
    </xsd:choice>
    <xsd:attribute name="id" type="xsd:string" use="required"/>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
  </xsd:complexType>
<!--Elemento application-->
  <xsd:element name="application" type="tns:app"/>
</xsd:schema>
```