PROYECTO FIN DE CARRERA

Aplicaciones de telefonía sobre JAIN SLEE

MEMORIA

Capítulo 1
Introducción

1. Introducción

1.1. Agradecimientos

La elaboración de este proyecto no habría sido posible, como es lógico, sin el conocimiento de la existencia de una tecnología tan innovadora como la que se aborda en él. Por ello, quiero agradecer a mi tutor Francisco José Fernández Jiménez no sólo el haber estado a mi disposición para cualquier sugerencia, planteamiento o duda, sino también el haberme mostrado la llave de algo que puede abrirme puertas en el futuro.

No puedo dejar escapar la ocasión para agradecer a todos aquellos profesores que a lo largo de la carrera me han aportado grandes cosas, tanto en mi desarrollo como ingeniero, como en mi crecimiento como persona. Tampoco puedo olvidarme de Ivelin Ivanov, fundador del proyecto Mobicents, que desde un principio ha confiado en mi, teniendo siempre palabras de ánimo y mostrando su disposición a tender la mano.

Mención especial merecen mi familia y amigos. Sin ellos nada de esto hubiera sido posible. Agradecer sobre todo a mis padres a mi hermano y, por supuesto, a Rocío, el haber estado a mi lado en los momentos difíciles y el haber dispuesto siempre de su incondicional apoyo y comprensión.

Por eso este proyecto está dedicado a todos ellos. Muchas gracias.

1.2. Motivación

Resulta obvio que exista una tendencia hacia la convergencia entre las distintas áreas tecnológicas de los operadores. Es en este aspecto donde Java se está configurando como un factor clave para el despliegue de nuevos productos y servicios. Sobre sus bases, JAIN SLEE (*Java APIs for the Advanced Intelligent Network/Service Logic Execution Environment - APIs de Java para la Red/Entorno de Ejecución de Lógicas de Servicio*) ofrece un entorno de ejecución de servicios en tiempo real que cumple los requisitos del mundo de las telecomunicaciones.

Durante los últimos años, el término convergencia ha sido uno de los más usados y abusados en el mundo de las telecomunicaciones. Debido al desordenado crecimiento en los años 90 y al frenético lanzamiento de servicios y productos en pos de captar la mayor cuota de mercado en el menor tiempo posible, algunas de las decisiones tomadas en aquellos momentos están hoy en curso de reconsideración. La meta es rediseñar las estructuras organizativas, técnicas y de sistemas de información de la empresa, en busca de una mayor integración y estandarización interna. Como ejemplos de convergencia, se puede recordar la convergencia prepago-postpago, convergencia fijo-móvil, convergencia en redes todo IP (*Internet Protocol - Protocolo de Internet*), etc. Sin embargo, existe un nuevo factor de convergencia de menor relevancia estratégica, pero que puede, no obstante, dotar al operador de mayores ventajas operativas. Se trata de la

convergencia tecnológica entre las principales áreas técnicas del operador (red, sistemas de información y productos y servicios). Y dentro de esta convergencia, una nueva palanca está actuando con fuerza, que es la creciente utilización de Java (y sus arquitecturas y herramientas asociadas) como elemento clave para el despliegue de nuevos productos y servicios.

La irrupción de Java ha sido uno de los fenómenos más innovadores en las tecnologías de la información durante la última década. Sin embargo, su deslumbrante éxito ha tenido en muchas ocasiones un contrapunto en la mitificación de la tecnología, lo que ha dado lugar a su inadecuada utilización.

Las grandes ventajas, propias de las arquitecturas J2EE (*Java 2 Enterprise Editions - Edición Empresarial del Paquete Java*), que Java ofrecía en el mundo de las tecnologías de la información (portabilidad entre plataformas, menores ciclos de implementación por reducción de tiempo en desarrollo y pruebas, acceso a la mayor comunidad de desarrolladores del mercado, etc.) no son suficientes para los servicios críticos de telecomunicaciones, con requisitos muy diferentes (asíncronos, entorno de ejecución orientado a eventos, tolerancia a fallos, grandes medidas de rendimiento, complejidad de lógicas de servicio, etc.). Debido a estas limitaciones, la utilización del mundo Java en las compañías de telecomunicaciones estaba hasta ahora reducida al campo tradicional de integraciones en Internet y en la Empresa (típicamente orientadas a servicios síncronos y fundamentalmente de petición-respuesta). Sin embargo, la continua mejora de la tecnología y el esfuerzo conjunto de organismos de estandarización (3GPP, ETSI), empresas del sector informático (Sun, Open Cloud, jNetX), operadores de telecomunicaciones e integradores de sistemas están haciendo posible la superación de las limitaciones técnicas existentes gracias a iniciativas como JAIN SLEE.

JAIN SLEE es el estándar Java para entornos de ejecución de lógicas de servicio. SLEE es un concepto maduro que se define como un entorno de ejecución de servicios con baja latencia, alto rendimiento y orientación a eventos de forma asíncrona, que son precisamente las características básicas de los servicios de telecomunicaciones. La novedad reside en la utilización de Java como soporte de este entorno. El ambicioso objetivo de JAIN SLEE es, por lo tanto, ofrecer un entorno de ejecución de servicios en tiempo real con todos los requisitos necesarios para los servicios de telecomunicaciones, contando como tecnología base Java y, por consiguiente, con todas las ventajas que esta tecnología ofrece.

Alrededor del concepto de NGIN (*Next Generation Intelligent Networks - Redes Inteligentes de Nueva Generación*) existen diferentes iniciativas de creciente alcance impulsadas por operadores, fabricantes y organismos internacionales. En el sector de las telecomunicaciones, el criterio de interoperabilidad (a diferentes niveles: redes, dispositivos, servicios, operadores, etc.) es uno de los catalizadores para el éxito del despliegue comercial masivo. En este sentido, existen diferentes organismos cuyo objetivo es conseguir esta normalización. Tomando como ejemplo uno de ellos, la misión del 3GPP es producir especificaciones técnicas aplicables globalmente para

sistemas móviles de tercera generación. Otros organismos con misiones similares son ETSI, OMA, ITU-T, etc.

Estos organismos han venido impulsando diferentes iniciativas con el objetivo común de abrir paso a la siguiente generación de servicios de red con los atributos comentados anteriormente. Sin embargo, como muchas de estas iniciativas, entre las que destacan algunas como OSA/Parlay (*Open Services Architecture - Arquitectura de Servicios Abierta*) y JAIN, se solapan entre sí, es preciso un esfuerzo de unificación para facilitar la utilización y evitar la lógica confusión que las nuevas tecnologías pueden conllevar. La unificación está en marcha y existen trabajos conjuntos para definir la convergencia entre las diferentes iniciativas.

Por tanto, se puede extraer de lo expuesto en este punto que la motivación de este proyecto ha sido la posibilidad tanto de estudiar esta tecnología tan innovadora, como de poner en práctica sus enormes beneficios. Todo esto acompañado de la estimulación de estar trabajando en algo que sin duda revolucionará el mundo de las telecomunicaciones.

1.3. Estado del arte

Está claro que gran parte de la población mundial, tanto expertos como no, está familiarizada o al menos le suena lo que es VoIP (Voz sobre IP). Sin embargo, a la pregunta de cuántos de ellos han escrito alguna vez una aplicación VoIP, la respuesta sería un número realmente bajo. La prueba de esto está en que el 100% de las personas que asistieron a la presentación de Mobicents (Mobicents es un proyecto perteneciente al grupo de desarrollo de tecnología Java *java.net* [1] y la primera y única plataforma VoIP de código abierto certificada por la conformidad de JAIN SLEE 1.0) en JavaPolis el 15 de Diciembre de 2005 estaban familiarizados con VoIP, pero ninguno de ellos había escrito una aplicación VoIP. Esto es algo que no debería sorprender tanto:

1. Hasta hace poco no existía una plataforma de código abierto disponible para que los desarrolladores pudieran hacer libremente modificaciones y pruebas de todo tipo.
2. No existe una verdadera red VoIP pública tal que los desarrolladores puedan desplegar nuevas aplicaciones VoIP y tener a alguien que las pruebe fácilmente. En lugar de eso, lo que hay es un número, por así decirlo, de islas propietarias como Skype, Google Talk y Yahoo! Messenger.

Por tanto, lo que se persigue con la tecnología JSLEE, y en concreto con la plataforma Mobicents, es una nueva generación de servicios de telefonía que cualquiera pueda empezar a escribir en sus portátiles con la misma facilidad que son escritas las aplicaciones web.

Comparando J2EE con JSLEE uno puede darse cuenta de aspectos tan interesantes como:

1. El lapso de tiempo medio de una transacción en J2EE es menor de 2 segundos, mientras que en JSLEE es menor de 2 milisegundos.
2. El volumen de transacción conseguido con JSLEE es 10 veces mayor que con J2EE.
3. El tiempo de funcionamiento en J2EE es de un 99'9% (equivalente a 9 horas al año parado), mientras que en JSLEE es de un 99'999% (equivalente a sólo 5 minutos al año parado).

Esto no hace más que reflejar las innumerables ventajas de JAIN SLEE, como son la reducción de la latencia y el consumo de recursos, el disponer de un entorno de programación abierto que lleva a la reducción del coste en el despliegue de servicios y a la reducción del tiempo de mercado, etc. Aunque, como en toda nueva tecnología, siempre habrá barreras que superar, como, por ejemplo, la necesidad de invertir en ella.

Un ejemplo de que esta tecnología se está comenzando a implantar se puede ver en Vodafone. El 5 de Julio de 2005, Vodafone España anunció en Madrid un servicio de voz para Red Inteligente, que requiere soporte para los protocolos SS7 (*Signaling System 7 - Sistema de Señalización por canal común nº 7*), MAP (*Mobile Application Part - Parte de Aplicación Móvil*), CAP (*Common Alerting Protocol - Protocolo de Alerta Común*) e INAP (*Intelligent Network Application Protocol -Protocolo de Aplicación de Red Inteligente*), y que es integrado dentro de las redes 2G y 3G de Vodafone en España. Por último, decir que este servicio ha sido desplegado con éxito por Vodafone sobre las plataformas jNetX [2] y Open Cloud (la plataforma de Open Cloud es el primer producto desarrollado que obedece al estándar JAIN SLEE 1.0 e implementa todas las características de dicho estándar, incluidas las opcionales).

Finalmente, para ver como el interés por esta tecnología está aumentando de una manera evidente y casi exponencial, se refleja en el siguiente gráfico (Figura 1.1) como ha sido esta progresión en el proyecto Mobicents.

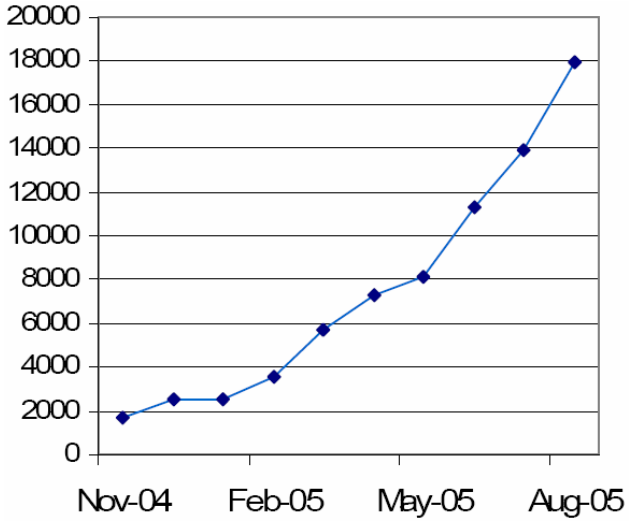


Figura 1.1: Evolución del proyecto Mobicents.

Se observa como el mayor incremento comienza a darse en torno a Julio de 2005. Esta evolución va unida a un mayor registro de desarrolladores, incremento en la calidad y en el número de aportaciones, mayor número de discusiones diarias en el foro del proyecto y un aumento de información en su sección *wiki*.

1.3.1. Visión general

En este apartado se hará una breve introducción a los aspectos tecnológicos que se tratarán en capítulos posteriores. De esta manera se podrá tener una idea general de cómo va a ser el proyecto desde un punto de vista tecnológico.

En dichos capítulos se hará hincapié en el SLEE y en la tecnología JAIN, sin dejar de ver, como es lógico, su confluencia en una única tecnología, la llamada JAIN SLEE. Para ello se tendrá en cuenta la plataforma Rhino perteneciente a Open Cloud y la plataforma Mobicents.

Por supuesto, también se hará mención, en mayor o menor medida, según su importancia, a todo aquello que sirva como base para soportar toda esta tecnología, es decir, lo que a fin de cuentas se llama "requisitos de la tecnología", tanto a nivel hardware, como a nivel software.

Finalmente, se dedicará un breve espacio a los siempre importantes protocolos, haciendo especial hincapié en el protocolo SIP.

La idea no es introducirse demasiado en los detalles técnicos, ya que para eso se dispone de una extensa bibliografía, que servirá, entre otras cosas, para saber qué especificaciones, RFCs, artículos o páginas web son las que hay que consultar en el momento que se estime oportuno.

1.4. Objetivo del proyecto

Para poder entender bien el porqué de este proyecto, se hará mención a una de las conversaciones que se mantuvo con Ivelin Ivanov (fundador del proyecto Mobicents). Después de estar un tiempo conversando, surgió el tema de las fuentes de información, y en concreto se le preguntó cómo se podía acceder a un tipo de información un poco más detallada, que la actualmente existente, de ciertos aspectos de la tecnología JAIN SLEE. Pregunta lógica, en cuanto que un hombre de su posición y que vive el día a día de esta nueva tecnología, quizás tuviese acceso o conocimiento de otras fuentes de información de más difícil localización para usuarios de a pie. Su respuesta, por desgracia, no fue distinta de la esperada: *"We are on the technology bleeding edge my friend. There is no where to learn from, other than the specification. If you write documents they will be the source for other people to learn from"*.

Por tanto, este proyecto tiene como objetivo poder ser fuente de la que puedan aprender otras personas, ya que se está en el difícil comienzo de una nueva tecnología y,

como tal, no existe suficiente documentación al respecto. Además, a todo esto se añade el hecho de la inexistencia casi absoluta de información en español, y debido a que desgraciadamente el inglés resulta ser, a día de hoy, una barrera todavía infranqueable para algunos estudiantes, no cabe duda que este proyecto puede ser de una gran utilidad para nuestro desarrollo en este nuevo campo tecnológico.

1.5. Perspectiva de los capítulos

Después de haber visto el capítulo de introducción, se pasará a resumir los capítulos siguientes en los que está dividido el proyecto:

- **Capítulo 2: Base teórica**

  Se referirá a aquellos aspectos teóricos que se consideren necesarios para la comprensión del proyecto, sin olvidarse de hacer mención a aquellas empresas y compañías que han servido para cumplir con los objetivos.

- **Capítulo 3: Tecnología JAIN SLEE**

  Se dedicará en exclusiva a explicar la tecnología JAIN SLEE, haciendo especial hincapié en su arquitectura y beneficios, e introduciéndose en las dos plataformas utilizadas que cumplen con dicha tecnología.

- **Capítulo 4: Trabajo desarrollado**

  Se explicará como se ha llevado a cabo el proyecto, así como los problemas encontrados y como se han ido solventando para finalmente cumplir con los objetivos planteados.

- **Capítulo 5: Líneas futuras y conclusiones**

  Se verán cuales son las posibilidades que alberga la tecnología JAIN SLEE para el futuro a corto, medio y largo plazo, y se sacarán conclusiones del proyecto realizado.

- **Capítulo 6: Fases del proyecto**

  Se explicará la evolución del proyecto a lo largo del tiempo y se hará uso de un diagrama de Gantt.

Capítulo 2
Base teórica

2. Base teórica

2.1. Introducción

En este capítulo se hará referencia a todos aquellos aspectos que se han considerado de especial importancia para la realización del proyecto. No sólo se tratará la teoría básica aplicada, sino que también se hará mención a aquellas empresas y compañías que han servido para cumplir el objetivo del proyecto.

En los siguientes apartados se presentará una estructura como la que se muestra a continuación:

- Java
  - Plataformas de JAIN SLEE
  - JBoss
  - APIs
    - JMX
    - RMI
- Protocolo SIP
- JAIN
- SLEE
- JSLEE

2.2. Java

Tal y como se ha comentado con anterioridad (1.2), el ambicioso objetivo de JSLEE consiste en ofrecer un entorno de ejecución de servicios en tiempo real con todos los requisitos necesarios para los servicios de telecomunicaciones, utilizando como tecnología base Java, pudiendo así aprovechar todas las ventajas que proporciona la tecnología ofrecida por Sun.

Java es una plataforma virtual de software desarrollada por Sun Microsystems, de tal manera que los programas creados en ella puedan ejecutarse sin cambios en diferentes tipos de arquitecturas y dispositivos computacionales.

La *plataforma Java* consta de las siguientes partes:

- El lenguaje de programación.

- La máquina virtual de Java o JRE (*Java Runtime Environment - Entorno de Ejecución Java*), que permite la portabilidad en ejecución.
- El API Java (biblioteca estándar para el lenguaje).

Por lo tanto, debido a su importancia en el proyecto, se ha considerado imprescindible hacer mención a Sun Microsystems, Incorporated.

Desde su comienzo en 1982, una visión singular -- "*The Network is The Computer*" ("*La Red es el Ordenador*") -- ha impulsado a Sun Microsystems, Inc. hacia su posición de proveedor líder y fuerte en la industria hardware, software y de servicios. Sun puede ser encontrado en más de 170 países, y en su sitio web [3].

Por último, comentar que Sun, Sun Microsystems, el logotipo Sun, JAIN, JMX, J2EE y "*The Network is The Computer*" son marcas registradas de Sun Microsystems, Inc. en los Estados Unidos y otros países.

2.2.1. Plataformas de JAIN SLEE

Las plataformas utilizadas en el proyecto fueron Rhino de Open Cloud y Mobicents, pero al ser comentadas con detalle en el capítulo 3, solamente se hará mención en este apartado a la compañía Open Cloud y al grupo Mobicents:

- Open Cloud [4] fue formada en el año 2000 para crear una tecnología de infraestructura software, basada en Java, que revolucionaría la portabilidad y la interoperabilidad de los servicios *middleware* (agentes software que actúan como un intermediario entre diferentes componentes de aplicación) en las telecomunicaciones, especialmente en 3G IMS (*IP Multimedia Subsystem - Subsistema Multimedia para IP*). La idea es la de entregar al mercado un servidor para la red de telecomunicaciones que sea tolerante a fallos y de alto funcionamiento de aplicación. Esto se consiguió con la publicación de Open Cloud Rhino.

  El 10 de Junio del 2004, Open Cloud anunció que su producto servidor de telecomunicaciones obedecía completamente a la especificación JSLEE 1.0. Por lo tanto, fueron los primeros que pasaron sin error todos los tests al aplicar el TCK (*Technology Compatibility Kit - Kit de Compatibilidad Tecnológica*) a su producto, en este caso Rhino 1.2.

  El 5 de Septiembre de 2005, Open Cloud anunció la disponibilidad general de la plataforma de telecomunicaciones Rhino 1.4. Por tanto, estaba disponible, para poder descargarse, el Kit para Desarrolladores Software de Rhino 1.4, es decir, el SDK. Ésta fue la primera plataforma SLEE que se utilizó y a partir de la cual se comenzó a dar los primeros pasos sobre esta tecnología.

Finalmente, decir que Open Cloud tiene representaciones en Londres, Madrid y Tokio, y tanto Rhino como el propio Open Cloud los tiene como marcas registradas.

- El 21 de Junio de 2005, Mobicents [5] anuncia ser la primera y única *Plataforma VoIP de Código Abierto* certificada por la conformidad de JSLEE 1.0. Está construido sobre la plataforma del Servidor de Aplicación JBoss [6]. Mobicents trae para las aplicaciones de telecomunicaciones un modelo integrante (component model) y un entorno de ejecución robustos. Además, complementa J2EE para habilitar convergencia de voz, video y datos en la próxima generación de aplicaciones inteligentes.

  En el ámbito de las Redes Inteligentes de Próxima Generación de las telecomunicaciones, Mobicents encaja como un motor de núcleo de alto rendimiento para Plataformas de Entrega de Servicios (SDP) y Subsistemas Multimedia para IP (IMS).

  Para terminar, comentar que el servidor Mobicents puede obtenerse libremente y de manera gratuita en su sitio web [5].

2.2.2. JBoss

El servidor de aplicaciones JBoss [6] fue dado a conocer por primera vez en 1999. Por otra parte, el Grupo JBoss fue fundado en año 2001 para proporcionar servicios de soporte técnico experto. Finalmente, El Grupo JBoss fue incorporado en 2004 como JBoss Inc.

JBoss es un servidor de aplicaciones J2EE de código abierto implementado en Java puro. Al estar basado en Java, JBoss puede ser utilizado por cualquier sistema operativo que lo soporte.

JBoss proporciona productos y servicios de apoyo para soportar proyectos de código abierto populares en la arena empresarial, como es el caso de Mobicents. Además, JBoss implementa todo el paquete de servicios J2EE.

Los principales desarrolladores trabajan para una empresa de servicios, JBoss Inc., fundada por Marc Fleury, el creador de la primera versión de JBoss. El proyecto está apoyado por una red mundial de colaboradores y los ingresos de la empresa están basados en un modelo de negocio de servicios.

2.2.3. APIs

Este apartado se dedicará prácticamente en exclusiva a la API JMX (*Java Management Extensions - Extensiones de Gestión Java*), que será la encargada de

definir todo aquello referente a la monitorización y administración del SLEE (donde se estarán ejecutando los servicios). Aunque, por otro lado, también se hablará de la API RMI (*Remote Method Invocation - Invocación Remota de Métodos*), ya que será la que permita invocar métodos de objetos escritos en Java que residen en otro ordenador (remotos) de la misma forma que si los objetos fueran locales.

2.2.3.1. JMX

2.2.3.1.1. Introducción

JMX (*Java Management Extensions - Extensiones de Gestión Java*) es una API estándar del JDK (*Java Development Kit - Kit de Desarrollo Java*) encargada de definir todo aquello referente a la monitorización/administración de aplicaciones basadas en Java.

Anteriormente, los servidores de aplicaciones y las aplicaciones empresariales dependían de las infraestructuras de administración y monitoreo propietarias. Por ejemplo, IBM [7] proveía la especificación PMI (*Performance Monitoring Interface - Interfaz de Supervisión de Rendimiento*) para monitorear WebSphere [8] y sus aplicaciones, mientras que Oracle [9] definió a los DMS (*Dynamic Monitoring Service - Servicio de Monitorización Dinámico*) para exponer las métricas de funcionamiento. Finalmente, tanto IBM como Oracle han adoptado completamente JMX en las últimas versiones de sus productos. JBoss también utiliza JMX.

Básicamente, JMX es a los sistemas de administración lo que JDBC (*Java Database Connectivity - Conectividad de Base de Datos Java*) es a las bases de datos. JDBC le permite a las aplicaciones acceder a bases de datos arbitrarias, mientras que JMX permite a las aplicaciones ser administradas por sistemas de administración arbitrarios. En definitiva, es una capa de aislamiento entre las aplicaciones y los sistemas de administración arbitrarios.

JMX define:

- Una arquitectura.
- Una API.
- Servicios y patrones de diseño para aplicaciones, gestión de red, y monitoreo utilizando el lenguaje de programación Java.

JMX aporta mecanismos para:

- Crear agentes y gestores en el lenguaje Java.

- Implementar aplicaciones de gestión distribuida (se utilizan APIs que implementan protocolos de gestión estandarizados y ampliamente utilizados en la industria).

JMX ha evolucionado en el tiempo. En la especificación actual de J2EE 1.4, soportada por los vendedores de servidores de aplicaciones principales, la versión requerida de soporte JMX es la versión 1.2 que puede ser bajada de JCP.org [10].

2.2.3.1.2. Arquitectura

La arquitectura JMX está construida sobre un modelo de tres capas. Esta flexibilidad permite que cada capa pueda ser utilizada individualmente por los desarrolladores.

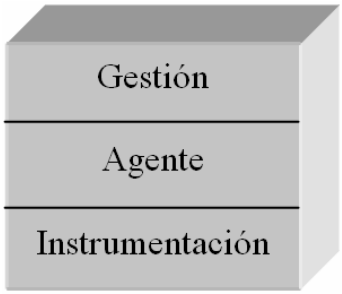


Figura 2.1: Arquitectura de la especificación JMX.

a) **Capa de instrumentación:** Es aquella que define los requerimientos para implementar recursos manejables JMX. Un recurso manejable puede ser cualquier cosa, incluyendo aplicaciones, componentes o dispositivos, y en la arquitectura JMX se implementan mediante MBeans. Los MBeans son objetos Java parecidos a los JavaBeans [11], que en la capa de instrumentación se encargan de representar cada uno de los recursos/aplicaciones. Un modo sencillo de ver los MBeans es pensar que son aquellas aplicaciones que se encargan de monitorizar otras aplicaciones.

b) **Capa de agente:** Se incorporará el MBean agente, aquél encargado de controlar los MBeans de la capa de instrumentación. Cada servidor de aplicaciones suele tener un solo agente, en el caso de JBoss su nombre es MBeanServer.

c) **Capa de gestión:** Aporta los componentes de gestión, los cuales pueden operar como un gestor o como un agente.

Por lo tanto, el nivel de instrumentación facilita la *gestión de la tecnología Java*, mientras que los niveles de agente y de gestión facilitan la *gestión a través de la tecnología Java*, y el conjunto de APIs que permiten la interacción con otros entornos de gestión facilitan la *integración con soluciones de gestión existentes*. En estos tres elementos se reflejan los objetivos de la especificación JMX.

2.2.3.1.3. Comunicación entre capas

La comunicación entre la capa de instrumentación y la capa de agente se produce debido a que el MBeanServer tiene un registro de cada MBean de la capa de instrumentación, cargándose todos los MBeans del sistema al desplegar el servidor.

Se dispondrán de dos tipos de MBeans en la capa de instrumentación, aquellos dedicados a enviar notificaciones (alerts) y aquellos encargados de recibir notificaciones (listeners). Un MBean dependiente de un recurso enviará una notificación si se cumple una condición impuesta por el programador, a la vez que un MBean Listener se encargará de recoger esa petición y realizar la operación correspondiente.

2.2.3.1.4. Conclusión

Los productos basados en la especificación JMX permiten la creación de aplicaciones de gestión usando la tecnología Java. Los elementos a gestionar pueden ser objetos escritos en el lenguaje de programación Java o cualquier otro recurso que tenga como intermediario un objeto escrito en Java.

Los niveles de agente y gestor ofrecen una infraestructura de gestión flexible, dinámica y distribuida, y las aplicaciones construidas en estos niveles son independientes de la plataforma, de los protocolos, de los modelos de información y de las aplicaciones de gestión.

JMX permite la integración con otras tecnologías, haciendo que la información que presenten sus recursos gestionables sea consistente con diferentes modelos de gestión, tales como SNMP (*Simple Network Management Protocol - Protocolo de Gestión de Red Simple*), CIM/WBEM (*Common Information Model/Web Based Entreprise Management - Modelo de Información Común/ Gestión Empresarial Basada en Web*), TMN (*Telecommunication Management Network - Red de Gestión de Telecomunicaciones*), entre otros.

Resumiendo, JMX es la mejor herramienta para la integración del software, ya que permite tener una espina dorsal a partir de la cual el usuario puede integrar todos sus módulos. Aplicado a servidores de aplicaciones, esta integración permite combinar toda la potencia de JMX en materia de estandarización y administración, así como sus propias cualidades como servidor.

2.2.3.2. RMI

RMI (*Remote Method Invocation - Invocación Remota de Métodos*) es el mecanismo ofrecido en Java que permite a un procedimiento (método, clase o aplicación) poder ser invocado remotamente. Una de las ventajas al diseñar un procedimiento con RMI es la interoperabilidad, ya que RMI forma parte de todo JRE, por lo que cualquier plataforma que tenga acceso a un JRE también tendrá acceso a

estos procedimientos, lo que le diferencia del clásico CORBA (*Common Object Request Broker Architecture - Arquitectura de Agente de Petición de Objeto Común*) que requiere de otros criterios.

2.3. Protocolo SIP

SIP (*Session Initiation Protocol - Protocolo de Inicio de Sesión*) [12] habilita a las puertas de enlace de voz sobre IP, clientes finales y otros sistemas de comunicación para interactuar unos con otros. Es un protocolo de señalización que puede iniciar, modificar y finalizar sesiones con muchos participantes. SIP direcciona el sistema de llamada sobre Internet, pero SIP no incluye el mecanismo para el flujo de información entre el llamante y el llamado, es decir, para poder realizar llamadas en Internet se necesita otro protocolo además de SIP. El protocolo SIP inicia solamente la comunicación, mientras que los datos propiamente dichos para la comunicación son elaborados por otros protocolos. Para esta actividad se utiliza SDP (*Session Description Protocol - Protocolo de Descripción de Sesión*) y RTP (*Real-Time Transport Protocol - Protocolo de Transporte de Tiempo Real*). SDP pone de acuerdo los códecs y protocolos de transmisión utilizados por los clientes. La tarea del RTP es, por otra parte, servir de mediador en el flujo de datos multimedia (audio, video, texto, y demás), es decir, codificar los datos, separarlos en paquetes y enviarlos.

Similar al protocolo HTTP, SIP es una aplicación del protocolo cliente/servidor, con peticiones realizadas por clientes y respuestas gestionadas por servidores. SIP habilita a los usuarios a participar en sesiones multimedia (o llamadas) y comunicarse en sesiones unicast y multicast.

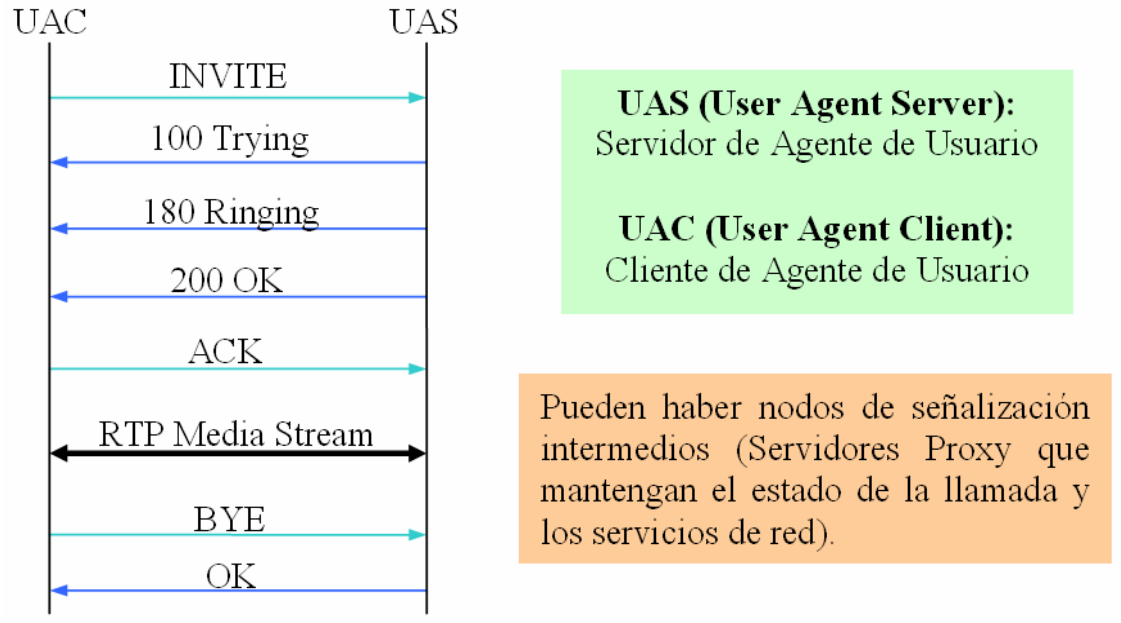


Figura 2.2: Ejemplo SIP de un flujo de llamada.

2.4. JAIN

2.4.1. Introducción

La tecnología JAIN (*Java APIs for the Advanced Intelligent Network - APIs de Java para la Red Inteligente Avanzada*) [13] está cambiando el mercado de las telecomunicaciones desde los sistemas propietarios cerrados, hacia una arquitectura abierta de rápido despliegue de servicios.

JAIN™ es un conjunto de APIs de la red integrada para la plataforma Java™ que proporciona un entorno de trabajo (framework) para construir e integrar soluciones (o servicios) que cruzan de un lado a otro de las redes de paquetes, las redes wireless y la red telefónica pública conmutada. El objetivo de JAIN es proveer portabilidad de servicio, convergencia y acceso seguro (para servicios residentes fuera de la red), para cada una de las redes integradas. JAIN es definido y especificado por la comunidad JAIN y de acuerdo con el JCP (*Java Community Process - Proceso de Comunidad Java*).

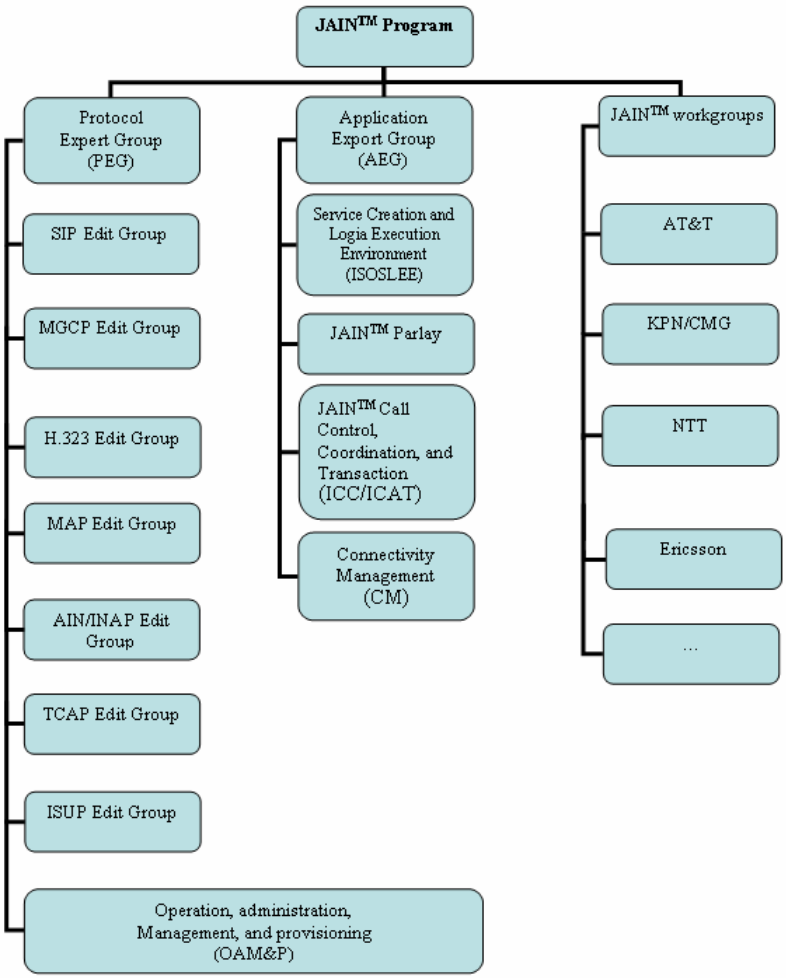


Figura 2.3: Organización de la comunidad JAIN.

La comunidad JAIN la conforman más de 80 compañías, y en ella se especifican más de 25 APIs con el objetivo de lograr la convergencia entre la red IP y la PSTN (*Public Switched Telephone Network - Red Telefónica Pública Conmutada*). JAIN incluye APIs para un alto nivel de desarrollo de aplicaciones, control de llamadas y también APIs de nivel de protocolo para señalización (SIP, SS7, etc.). Hay muchos productos JAIN que están siendo desarrollados en la industria. Además, las APIs de JAIN han sido adoptadas por la mayoría de los vendedores *softswitch* (término genérico para cualquier software pensado para actuar de pasarela entre la red telefónica y algún protocolo de VoIP) y 3GPP OSA/Parlay.

2.4.2. Objetivos

El programa JAIN tiene 3 objetivos:

a) **Portabilidad de servicios:** El hecho de que tanto el desarrollo de la tecnología y las aplicaciones esté restringido a interfaces propietarias provoca un incremento en el coste del desarrollo de aplicaciones, el tiempo de mercado (time to market) y los requisitos de mantenimiento. Lo que aborda JAIN es dar una nueva forma a las interfaces propietarias, tal que se conviertan en interfaces uniformes de Java para el desarrollo de aplicaciones portables. A partir de proporcionar APIs estandarizadas, las aplicaciones podrán arrancar ininterrumpidamente en equipos que obedezcan a diferentes vendedores de JAIN. Para resumir: *"Escribirlo una vez y arrancarlo en cualquier sitio"*.

b) **Convergencia de la red:** Al proporcionar APIs abstraídas, las aplicaciones pueden arrancar y comportarse de la misma manera sin preocuparse si la tecnología de red que hay por debajo es por ejemplo SS7, IP o cualquier otra cosa. Para resumir: *"Cualquier red sirve"*.

c) **Acceso seguro a la red:** Se permite a las aplicaciones que están arrancando fuera del dominio seguro del operador de red, acceder a las capacidades de red de una manera controlada y segura. La oportunidad de mercado para nuevos servicios es enorme cuando se proporciona acceso controlado a la inteligencia y funcionalidad disponible dentro de las redes de telecomunicaciones. De esta manera, miles de nuevas aplicaciones innovadoras serán desarrolladas. Para resumir: *"Accesible por todo el mundo"*.

Las perspectivas para JAIN no podrían ser mejores. Después del último grupo de reuniones mantenidas por la comunidad JAIN, es evidente que la industria está adoptando sin reservas sus estándares y adelantándose a la realización de APIs a través de la estandarización de los elementos en una oferta software (*productization*). Además de todo esto, decir que todos los principales NEPs (*Network Equipment Provider - Proveedor de Equipo de Red*) se han apuntado a la tecnología JAIN y son participantes activos implementando APIs en productos reales. Para ver la lista de miembros de la comunidad JAIN, acudir a la página de miembros de JAIN [14].

Es conveniente puntualizar que son numerosos los grupos de trabajo, como por ejemplo 3GPP, Parlay o ETSI, los que están trabajando en el mismo campo que JAIN. Sin embargo, se debe hacer notar que es JAIN el único grupo que está estandarizando las APIs de Java para las redes integradas.

2.4.3. Arquitectura

A continuación se explicará en que consiste la arquitectura JAIN. No se alcanzará un alto nivel de detalle, ya que siempre se puede hacer uso de las especificaciones [15].

En JAIN, la capa de protocolo está basada en la estandarización de los protocolos específicos [15] (SIP, MGCP, H.323, TCAP, ISUP, INAP/AIN, MAP, etc.), mediante el uso de Java (Tabla 2.1). Al proveer de interfaces de protocolo estandarizadas siguiendo el modelo de objetos Java, tanto las aplicaciones como las pilas de protocolo pueden ser dinámicamente intercambiadas y, al mismo tiempo, proporcionar un alto grado de portabilidad a las aplicaciones en la capa de aplicación, usando pilas de protocolo procedentes de distintos vendedores.

Interfaces de Aplicaciones Java para Comunicaciones	Contenedores de Aplicaciones Java para Comunicaciones
JAIN SIP 1.1	JAIN Service Logic Execution Environment (JSLEE) 1.0
SIP API for J2ME 1.0	SIP Servlets 1.0
JAIN MGCP 1.0	
JAIN MEGACO	
JAIN SDP	
JAIN ENUM	
JAIN TCAP 1.1	
JAIN INAP 1.0	
JAIN JCC 1.1	
JAIN JCAT	
Java Payment API (JPay)	
JAIN Presence	
JAIN Instant Messaging	
JAIN SIMPLE Instant Messaging	
JAIN SIMPLE Presence	
JAIN SIP Lite	
JAIN Service Creation Environment (SCE) - SCML	
JAIN Service Creation Environment (SCE) - Java	
Server API for Mobile Services (SAMS): Messaging	

Tabla 2.1: Especificaciones de las APIs de JAIN.

La capa de aplicación proporciona un sólo modelo de llamada o sesión sobre el que están soportados todos los protocolos en la capa de protocolo. La idea fundamental es proporcionar una sola máquina de estados para sesiones multipartícipe, multimedia y multiprocolo para los componentes de servicio en la capa de aplicación.

En su núcleo, la arquitectura JAIN define una librería de componentes software, un conjunto de herramientas de desarrollo, un entorno de creación de servicios y un entorno de ejecución de lógicas de servicio para construir la siguiente generación de servicios para redes integradas PSTN, de paquetes y wireless.

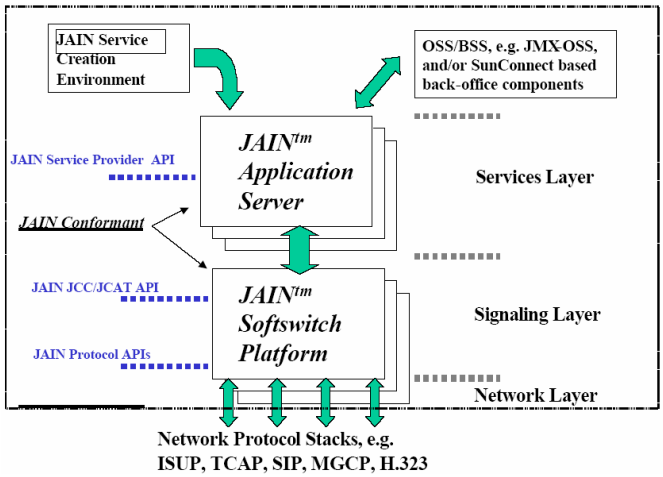


Figura 2.4: Arquitectura JAIN.

Conviene decir que la Tabla 2.1 detalla las actuales especificaciones de las APIs de JAIN, tanto las que están desarrollándose como las que están disponibles.

A continuación se comentará brevemente JAIN SIP, por ser precisamente SIP la base fundamental sobre la que se ha asentado el proyecto desde el punto de vista de los protocolos de red.

2.4.3.1. JAIN SIP

2.4.3.1.1. Motivo de la creación de JAIN SIP

El motivo de su creación podría ser válido para cualquiera de los demás protocolos a los que a día de hoy JAIN proporciona una interfaz estandarizada.

Pues bien, como desarrollador se es libre de implementar un protocolo en cualquier lenguaje, de ahí el definir tu propia interfaz para acceder al comportamiento definido del protocolo mediante, por llamarlo así, un esbozo del estándar del IETF (*Internet Engineering Task Force - Grupo de Trabajo en Ingeniería de Internet*). Lo que ocurre es que mientras la especificación del IETF asegure la interoperabilidad entre

pilas (en esta caso pilas SIP), no se preocupa de la interoperabilidad de las aplicaciones a través de las pilas.

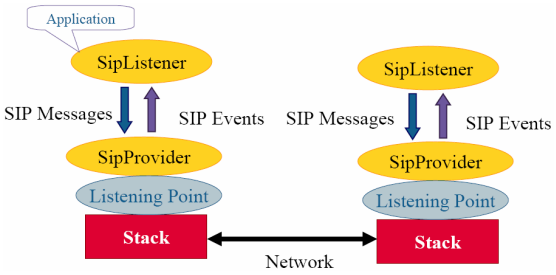


Figura 2.5: Arquitectura de la mensajería JAIN SIP.

Con el nacimiento de JAIN SIP, se satisface esta necesidad con el lenguaje de programación Java. Se asegura que utilizando la especificación JAIN SIP, se va a tener interoperabilidad entre pilas y la interoperabilidad de las aplicaciones a través de dichas pilas, que es a lo que comúnmente se le conoce como portabilidad de las aplicaciones, es decir, que no importe quien ni como haya implementado la pila, siempre y cuando cumpla con la especificación e incorpore una interfaz JAIN SIP.

Decir que la API JAIN SIP dota a la industria de una interfaz estándar dentro de las pilas de protocolo SIP propietarias. Así pues, la API JAIN SIP incluye:

- Interfaces de Servidor Agente/Usuario.
- Interfaces de Servidor *Proxy*.
- Interfaces de Servidor *Redirect*.

Para más información acerca de todo esto, acudir a la RFC correspondiente al protocolo SIP (rfc3261) [12] y a la especificación de JAIN SIP [16].

2.4.4. Interfaces estándares para la señalización de servicios

La iniciativa JAIN define un conjunto de estándares de interfaces para los protocolos de señalización y servicios. Definiendo una interfaz común entre dos o más protocolos las redes convergen. La API JCC (*Java Call Control - Control de Llamada Java*) y el JCAT (*Java Coordination and Transaction - Transacciones y Coordinación Java*) junto con la comunidad JAIN definen cada una de las interfaces de control/sesión de llamada.

El objetivo de las APIs JCC/JCAT es el de proporcionar aplicaciones con un mecanismo con vistas al control de llamada y a las transacciones de procesamiento de llamada. JCC/JCAT incluye las facilidades requeridas para observar, iniciar, contestar, procesar y manipular llamadas, donde una llamada puede incluir sesiones multipartícipe o multimedia sobre la red integrada que haya debajo (PSTN, de paquetes y/o wireless). JCC es un modelo de llamadas unificado que incorpora las características básicas de

JTAPI (*Java Telephony API - API de Telefonía Java*), así como el servicio de control de llamada de Parlay y un entorno de trabajo Java extensible para soportar las aplicaciones de procesamiento de llamada complejo.

2.4.5. Topología de red

La topología de red de JAIN proporciona portadores con la habilidad de desplegar servicios de tercera generación en dispositivos situados dentro o en el filo de la red integrada, incluyendo cualquier dispositivo de usuario final habilitado para la tecnología Java. Además, JAIN también proporciona soporte para todos los protocolos de telefonía necesarios, que son usados entre los diferentes elementos de red en las redes inteligentes, redes inteligentes avanzadas y las redes basadas en IP.

Un aspecto clave de la arquitectura integrante de JAIN es el mover la capa de señalización fuera de los conmutadores propietarios e introducirla en los servidores de control de llamada abiertos, también conocidos como agentes de llamada, controladores *media gateway* o *softswitches*. Cuando se habla de señalización, se refiere a los protocolos usados para establecer y terminar conexiones de comunicación, es decir, se trata de un hilo común entre *switches* y *softswitches* convencionales de telecomunicaciones. La habilidad para adaptar los componentes de señalización entre redes es primordial para el éxito de los proveedores de servicios de red.

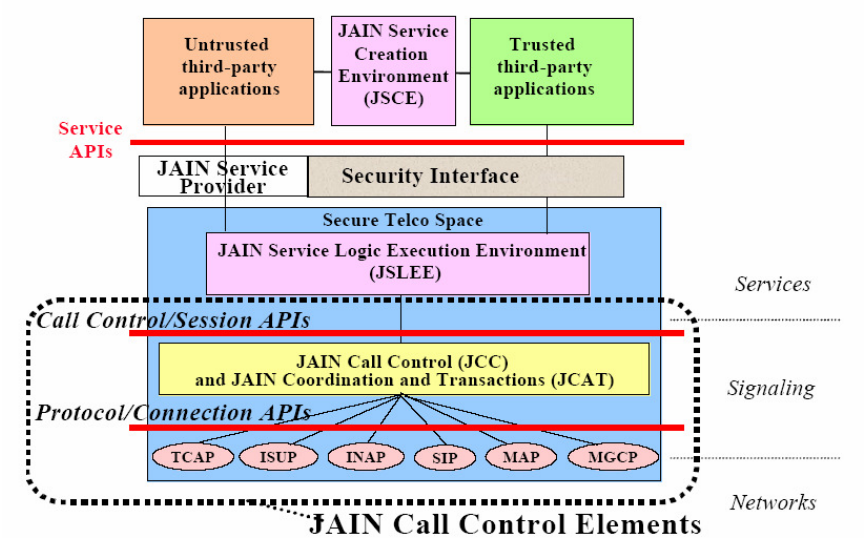


Figura 2.6: APIs de JAIN.

En la Figura 2.6 hay una representación en donde están definidas las APIs de JAIN dentro de una plataforma de comunicaciones. La arquitectura *softswitch* está centrada en mapear las interfaces control/sesión de llamada con las APIs de protocolo subyacentes. Desde que los *softswitches* desempeñan señalización en las redes IP, la mayoría de ellos están equipados con los protocolos subyacentes SIP, MGCP o H.323. Además, varios *softswitches* incluyen también el protocolo SS7 para dirigir interfaces para la red telefónica existente.

La iniciativa JAIN ofrece componentes clave para construir arquitecturas *sofswitch* abiertas y proporcionar portabilidad de servicios a través de una interfaz de Java unificada para desarrollar y desplegar los servicios de próxima generación. Decir también que el armazón JAIN de APIs está dividido en 4 categorías:

- Interfaces de Protocolo o de Conexión.
- Control de Llamada o Interfaces de Sesión.
- Interfaces de Lógicas de Servicios.
- Acceso al Proveedor de Servicios.

2.4.6. Conclusión

JAIN decidió en su momento evolucionar con el tiempo, permitiendo la incorporación de nuevos métodos. Además, hay que destacar que los componentes que obedecen a JAIN no residen necesariamente en un único servidor. Todo está hecho para proporcionar ventajas de escalabilidad, funcionamiento, fiabilidad, gestión, reutilización y flexibilidad.

En definitiva, la tecnología JAIN está cambiando el mercado de las comunicaciones, yendo desde los sistemas propietarios cerrados, hacia una única arquitectura de red donde los servicios puedan ser desarrollados y creados rápidamente.

2.5. SLEE

2.5.1. Introducción

La especificación API del SLEE (*Service Logic Execution Environment - Entorno de Ejecución de Lógicas de Servicio*) define un entorno de trabajo de aplicación para el desarrollo de servicios de telecomunicación portables. Fue especificada en conformidad con el JCP como la JSR 22 [17] (JSLEE v.1.0). Hacer notar que actualmente se está elaborando la JSR 240 [18] (JSLEE v.1.1), la cual está aún en fase de borrador. Finalmente, la aprobación de la JSR 22 [17] se llevó a cabo el 17 de Febrero del 2004. La especificación está liderada por David Ferry de Open Cloud y Swee Lim de Sun Microsystems. Sin olvidarnos de que el grupo experto para esta JSR incluye también compañías como Siemens, IBM, Motorola y NTT Coorporation.

La SLEE RI (*Reference Implementation - Implementación de Referencia*) fue construida por la compañía Open Cloud, la cual también creó la Implementación de Referencia de JSLEE y vende una implementación del SLEE que no es construida encima de la arquitectura EJB (*Enterprise JavaBeans - JavaBeans para la Empresa*) [19]**.** Su producto se hace llamar Open Cloud Rhino y se hizo mención a él en los apartados 1.3 y 2.2.1.

Los 4 elementos básicos del SLEE son los Adaptadores de Recurso, Eventos, Contextos de Actividad y el Entorno de Ejecución, que es el que acoge a los Objetos SBB (*Service Building Block - Bloque de Construcción de Servicios*). La Figura 2.7 muestra la relación existente entre estos elementos.

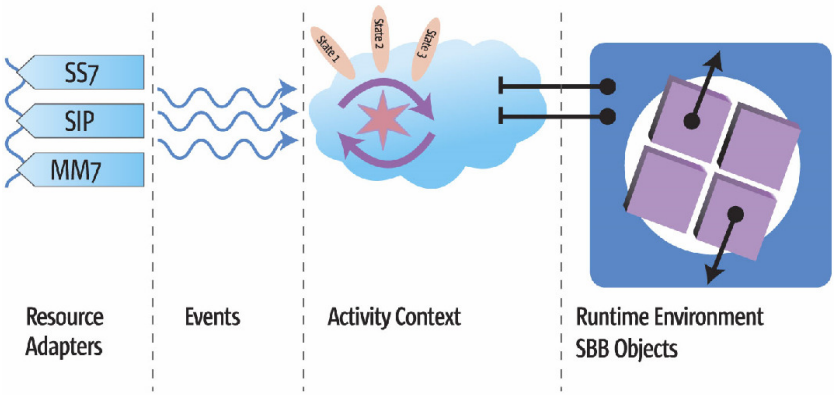


Figura 2.7: Los 4 elementos básicos del SLEE.

2.5.2. Elementos básicos del SLEE

Los adaptadores de recurso son responsables de la comunicación con los protocolos de red externos. Ellos pueden enviar y recibir eventos. Tan pronto reciben un evento generado por la red externa, le muestran este evento al contexto de actividad como objetos de evento (no hacen un consumo o tratamiento del evento). El SBB, localizado dentro del entorno de ejecución del SLEE, tiene interfaces con los contextos de actividad. Los contextos de actividad son usados para repartir estos eventos a los SBBs. Como los adaptadores de recursos pueden comunicarse bidireccionalmente, también pueden emitir eventos a la pila de protocolo origen. Cada uno esos eventos pueden ser disparados por SBBs que se estén ejecutando dentro del SLEE y estén respondiendo a peticiones entrantes.

El contexto de actividad es una entidad lógica dentro del SLEE que recibe y enruta los eventos hacia los componentes SBB. El enrutamiento es desempeñado por el enrutador de eventos, que es parte del SLEE. Los eventos pueden ser duplicados y rutados hacia varios componentes SBB.

A continuación se verá con más detalle los 4 elementos básicos del SLEE:

- **Adaptadores de Recurso:** Los adaptadores de recurso se comunican con los sistemas externos al SLEE. Por ejemplo, dispositivos de red, pilas de protocolo, directorios o bases de datos. De acuerdo con la arquitectura SLEE, un adaptador de recurso es una implementación específica de un tipo de adaptador de recurso (resource adaptor type). Para entenderlo mejor, decir que el RA Type es la interfaz, mientras que el RA (*Resource Adaptor - Adaptador de Recurso*) es la implementación de esa interfaz. Por lo tanto, para los temas de compatibilidad, una de las cosas más

importantes a tener en cuenta será el RA Type. Decir también, que una instancia del propio adaptador de recurso dentro del SLEE se denomina entidad del adaptador de recurso.

El RA Type declara todos los tipos de evento que pueden ser disparados y todas las actividades que el adaptador introduce. Cuando un adaptador de recurso pasa un evento al SLEE, debe proporcionar el objeto de evento (event object), el tipo de evento (event type) y una actividad (activity). La especificación no establece como se pasa esta información al SLEE. Esta API está por encima del implementador de la especificación. Debido a esta ausencia de claridad, una nueva JSR, que debería definir claramente un Armazón de Adaptador de Recurso, fue introducida en Marzo del 2004. Aunque a día de hoy todavía está en el informe borrador.

- **Eventos:** Los objetos de eventos se encargan de llevar la información desde una entidad que está dentro del SLEE hacia otra. Tan sólo las entidades SBB pueden tanto producir como consumir eventos, mientras que otras entidades, como son los adaptadores de recurso, el SLEE en sí mismo y las facilidades SLEE, pueden únicamente producir eventos. Cada evento está representado por un objeto de evento (subclase de *java.lang.Object*) y un tipo de evento. El tipo de evento determina como enrutará el SLEE el evento.

  Por cada evento que un SBB dispara, el desarrollador necesita especificar un método de disparo de evento abstracto. Este método es implementado por el SLEE. Las entidades SBB reciben eventos procedentes de contextos de actividad unidos a ellas. En el caso de tratarse de un evento inicial, el SLEE crea primero un objeto SBB y luego enruta el evento hacia el SBB.

- **Actividad y Contexto de Actividad:** Una actividad es una entidad lógica del SLEE que representa a un flujo relacionado de eventos. La representación de Java de esta entidad lógica es el objeto de actividad, que es creado bien por entidades de adaptadores de recurso, o bien por facilidades SLEE.

  Un contexto de actividad representa a la actividad subyacente dentro del SLEE, y también puede almacenar atributos, tal que las entidades SBB los puedan utilizar. Los objetos SBB pueden acceder al contexto de actividad, para acceder a esos atributos, a través del objeto de interfaz de dicho contexto de actividad.

  Un SBB puede usar una interfaz de contexto de actividad genérica, o bien extender esta interfaz y definir atributos adicionales que quiera compartir con otros objetos SBB. Aquí se tiene otra libertad para el desarrollador.

Los objetos de actividad son generados, por regla general, por los eventos de red. Los adaptadores de recurso escuchan a estos eventos y crean los objetos de actividad apropiados. Estos objetos son puestos en el contexto de actividad del SLEE. El SLEE es ahora responsable de entregar a los objetos SBB los eventos generados. Por contra, un objeto SBB puede acceder a la interfaz de contexto de actividad para conseguir acceder al objeto de actividad actual, pudiendo disparar eventos en este objeto, que serán entregados de vuelta a los adaptadores de recurso y a la red subyacente.

Para conseguir el acceso a un objeto de actividad, el desarrollador de SBB accede, como de costumbre, a la interfaz de contexto de actividad, que está automáticamente disponible dentro de cada método manejador de eventos de la clase abstracta SBB.

- **Entorno de Ejecución y Clase Abstracta SBB:** De acuerdo con la especificación, el entorno de ejecución SLEE debe hacer algunas APIs disponibles para los componentes SBB en ejecución. Decir que tan sólo se especifica un mínimo entorno de ejecución, dejándose al implementador el hecho de poder proporcionar funcionalidad adicional. La clase abstracta SBB es parte del componente SBB (que también incluye las interfaces locales y el descriptor de despliegue de SBB) y contiene la lógica de procesamiento de eventos, que tiene que ser añadida por el desarrollador de SBB. El desarrollador de SBB implementa una clase abstracta para cada componente SBB. Por otra parte, es el entorno de ejecución el responsable de crear las instancias SBB asociadas a partir de estas clases abstractas, además de implementar ciertos métodos abstractos y de crear instancias que procesen eventos entrantes.

  Cada clase abstracta SBB implementa la interfaz *javax.slee.Sbb* y debe ser definida de manera pública y abstracta. Los métodos concretos contienen la lógica de aplicación del componente, mientras que los métodos abstractos se ocupan de los eventos de disparo, CMP (*Container Managed Persistence - Persistencia Gestionada por Contenedor*), gestión de la *child relationship* (relación padre-hijo entre los SBBs) y el acceder al contexto de actividad específico. Los métodos abstractos son implementados por el SLEE, el cual usa introspección y datos a partir de los descriptores de despliegue para crear este código específico (apartado 3.1 de la especificación [17]).

2.6. JSLEE

La tecnología JAIN SLEE [20] ha sido el eje en torno al cual ha girado el proyecto. Por este motivo se ha utilizado también para darle nombre al mismo (“Aplicaciones de telefonía sobre JAIN SLEE”). En definitiva, se considera esta

tecnología de tanta importancia que se ha decidido dedicarle un capítulo, en el que se encontraran los aspectos más importantes referentes a esta tecnología (capítulo 3).

La principal ventaja de JAIN SLEE es la estandarización del desarrollo y ejecución de servicios, de modo que se cubran los diferentes niveles y, muy especialmente, la capa de acceso a los elementos de telecomunicaciones. JAIN SLEE ofrece un alto nivel de abstracción para el acceso a las capacidades de red, simplificando en gran medida la complejidad existente y, al mismo tiempo, ofreciendo Java como lenguaje de implementación. Paralelamente, JAIN SLEE incorpora los conceptos tradicionales de la arquitectura Java (reusabilidad de componentes, facilidades de gestión y concurrencia, distribución, etc.). Sobre esta base, el desarrollo y despliegue de servicios se realizaría en un tiempo menor y por un mayor número de profesionales, con las consiguientes ventajas para el operador y los clientes finales (mayor número de servicios con un ciclo de lanzamiento más corto y dinámico).

En cuanto a los servicios que se pueden construir y desplegar sobre JAIN SLEE, se pueden citar como ejemplos los servicios tradicionales de inteligencia de red como VPN (*Virtual Private Network - Red Virtual Privada*), traducción de números o *least cost routing*, servicios de localización, servicios de mensajería, como cobro en tiempo real de MMS (*Multimedia Messaging System - Sistema de Mensajería Multimedia*) y servicios basados en señalización SIP, así como cualquier combinación de los anteriores.

JAIN SLEE, como paradigma de los entornos de nueva generación, rompe con la tradicional distinción entre servicios de inteligencia de red y servicios IP. El concepto de Adaptadores de Recurso permite la integración entre diferentes tecnologías en servicios innovadores y de nueva generación, así como el despliegue de servicios tradicionales basados en señalización, como la gestión de enrutamiento de llamadas. No existe un único campo de servicios adecuado para JAIN SLEE, por el contrario, una de las principales ventajas es su gran alcance, el cual es además ampliable mediante la incorporación de adaptadores de nuevos protocolos. Otra de las ventajas es su definición abierta, que elimina las barreras tradicionales de integraciones propietarias.

Los primeros servicios comerciales basados en JAIN SLEE están ya en producción en algunos de los operadores más importantes del mundo. Sin embargo, a pesar de lo prometedor del concepto, el camino por recorrer es todavía extenso. Baste decir que la especificación de JAIN SLEE fue aprobada en febrero de 2004 y las plataformas comerciales que cumplen la especificación son todavía muy pocas. En este sentido, uno de los peligros existentes es que los trabajos de especificación queden por detrás de las implementaciones reales. La prioridad fundamental es ofrecer una portabilidad real entre plataformas comerciales, aspecto que se cubrirá en la siguiente versión de la especificación (JSR 240 [18]).

Capítulo 3
Tecnología JAIN SLEE

3. Tecnología JAIN SLEE

3.1. Introducción

A la conclusión de este capítulo se pretende conseguir tener una visión general de la tecnología JSLEE, de la que, sin duda alguna, ya se saben bastantes cosas gracias a que en el capítulo 2 se hace referencia tanto al SLEE como a la tecnología JAIN.

Para alcanzar esa meta se plantean los siguientes objetivos a cumplir:

- Entender el significado del término SLEE y como se relaciona con JSLEE.
- Entender los objetivos y propósitos de JSLEE.
- Entender la arquitectura JSLEE y su modelo de programación a un alto nivel.
- Apreciar algunas de las consideraciones de robustez de JSLEE.
- Entender como JSLEE se relaciona con otras tecnologías.
- Conocer el estado del estándar JSLEE.

Puede que algunos conceptos, tanto de JAIN como del SLEE, resulten familiares del capítulo 2, pero se considera indispensable el hecho de repetir esos conceptos para facilitar y afianzar el aprendizaje de una tecnología que goza de un alto nivel de complejidad.

Finalmente, aunque será en capítulos posteriores donde se entrará en detalle de las dos plataformas utilizadas en el proyecto (funcionamiento, aplicaciones utilizadas sobre ellas, facilidades, etc.), es decir, Rhino de Open Cloud y Mobicents, que mejor momento que el final de este capítulo para hablar de los aspectos técnicos referentes a estas plataformas, viendo así como se les incorpora la tecnología JSLEE.

3.2. Significado del SLEE

Como se ha venido diciendo hasta ahora, SLEE es un Entorno de Ejecución de Lógicas de Servicio y no es ni mucho menos una idea nueva, aunque eso no significa que esté ya totalmente desarrollada, ya que aún queda un largo camino.

Las ventajas de usar el SLEE son que los servicios pueden ser desarrollados rápidamente extendiendo y usando el núcleo común del entorno de ejecución. Además de esto, los servicios son más robustos, ya que el núcleo común está altamente probado y desplegado.

Sin embargo, a pesar de tratarse de una gran idea, está el inconveniente de la incompatibilidad entre los SLEEs existentes actualmente:

- Los servicios y las características no son portables entre los distintos SLEEs.
- Los servicios y las características ubicados en diferentes SLEEs no pueden interactuar entre sí debido al hecho de que el SLEE, a día de hoy, no soporta interoperabilidad.

3.3. Relación con JAIN™

Las APIS de JAIN para redes integradas traen portabilidad de servicios, convergencia y seguridad en el acceso a la red de datos y telefonía. Proveen de un nuevo nivel de abstracción e interfaces de Java asociadas para la creación de servicios sobre la PSTN, la red de paquetes y las redes inalámbricas. La tecnología JAIN también habilita la integración de Internet y los protocolos de redes inteligentes. Además, el hecho de permitir tener con las aplicaciones Java un acceso seguro a los recursos dentro de la red, da la oportunidad de repartir miles de servicios mejores que los actualmente disponibles. Así, la tecnología JAIN está cambiando el mercado de las comunicaciones desde muchos sistemas propietarios cerrados hacia una única arquitectura de red donde los servicios pueden ser rápidamente creados y desplegados.

El Entorno de Ejecución de las Lógicas de Servicio de JAIN es una parte integrante del conjunto de las APIS de JAIN. Lo realmente importante es la lógica y el entorno de ejecución en el que las aplicaciones de comunicación son desplegadas para utilizar los distintos recursos de red, definidos por las otras APIS de JAIN. Esencialmente, la especificación de JAIN SLEE define interfaces y requisitos para las aplicaciones de comunicación dentro de una plataforma de comunicaciones, utilizando otros estándares de comunicación tanto JAIN como no JAIN.

3.4. Desafíos en el despliegue de servicios

3.4.1. Introducción

En este punto se comentarán los desafíos que se encuentran al querer desplegar nuevos servicios, tanto en las redes actuales como en las futuras redes y se verá como ayuda JSLEE a solventar estos problemas. Sin duda, esto servirá para entender la utilidad y la necesidad de la aparición de JSLEE en nuestras vidas.

El negocio de los móviles de tercera generación se fundamenta en la promesa de nuevas redes con valor añadido, y esencialmente en el despliegue de nuevos servicios a bajo coste. Sin embargo, parece que los usuarios no terminan de creer suficientemente en este beneficio adicional como para hacer la migración.

Ya en Septiembre de 2002 se publicó un artículo [21] en el que se decía: "*Solo el 10% de los usuarios de móviles en Europa utilizarán UMTS en el 2007,…*" (Por Michelle de Lussanet). Parece que no iba mal encaminado.

El mayor desafío es, por tanto, encontrar una forma de desarrollar y vender el valor añadido de los servicios de forma progresiva. Entonces, la gran pregunta es cómo iniciar el crecimiento de ganancias debido a los nuevos servicios, usando las redes actuales o las nuevas de forma que se satisfagan de la mejor manera posible las necesidades. Esto plantea diferentes cuestiones discutibles:

- **Ofrecer servicios sobre soluciones End-to-End propietarias:** Existe cierta convergencia en el nivel de transporte en el núcleo de red. Sin embargo, la convergencia en el servicio y en los niveles OSS/BSS (*Operations Support Systems/Base Station Subsystem - Sistemas de Soporte de Operaciones/ Subsistema de Estación Base*) está limitada. La convergencia sólo existe si el proveedor del servicio es el mismo que el dueño de la red fija y de la red móvil. Incluso en casos donde los operadores están consolidados, existe dificultad en el despliegue rápido de los servicios debido al número de redes.
- **Protección de la inversión en las redes actuales:** Se ha hecho una gran inversión en las actuales redes de telecomunicaciones, por lo que es lógico intentar aprovecharlas al máximo.
- **Protección de la inversión en el desarrollo de nuevos servicios:** Para mantener el crecimiento en los ingresos y en los beneficios, es preferible que los nuevos servicios sean desarrollados para el uso de las redes actuales, pero es necesario asegurarse la migración a las nuevas redes.
- **Diferenciación en la oferta de servicios:** El nuevo crecimiento de ingresos y la retención de clientes vendrá determinado por los nuevos servicios que diferenciarán a los proveedores.
- **Ofrecer servicios rápidamente en respuesta al mercado:** Un competidor debe ser el primero en implementar los servicios; y una rápida respuesta por parte de los demás competidores es crítica para mantener la libre competencia.
- **Ofrecer servicios personalizados dirigidos tanto a un colectivo determinado, como a un usuario particular:** Los proveedores de servicios necesitan encontrar una forma eficiente de ofrecer servicios personalizados.
- **Desarrollar servicios nuevos que dirijan la eficiencia de funcionamiento:** Cuando un cliente es capaz de acceder a su propio perfil de cuenta y gestionarlo, se siente con control y esto deriva en mayor satisfacción por parte del cliente.
- **Desarrollar servicios nuevos e innovadores usando terceras partes:** Para lograr innovación, los operadores y los proveedores de servicio

deben encontrar formas de permitir a terceras partes el poder desarrollar servicios en sus propias redes.

- **Desarrollar servicios web, de datos y de voz que converjan:** Algunos opinan que sería mejor invertir el dinero no solamente en los servicios de tercera generación, sino también en estimular y enriquecer la mensajería y los servicios con contenido.

3.4.2. Soluciones propuestas por JSLEE

A continuación se verá, a modo de tabla (Tabla 3.1), como JSLEE ayuda a hacer frente a los desafíos anteriormente comentados en el punto 3.4.1.

Desafío	Ayuda de JSLEE a hacer frente al desafío
Ofrecer servicios sobre soluciones End-to-End propietarias	- Permite la elección de una plataforma de servicios estándar mientras se fomenta la competencia. - Abre las limitaciones impuestas por un único proveedor de equipos. - Permite servicios portables innovadores. - Permite un alto funcionamiento y baja latencia.
Protección de la inversión en las redes actuales	- Permite la creación de nuevos servicios que abarquen redes dispares existentes. - Genera nuevos ingresos y modelos de negocio para que los servicios creados puedan ser migrados a las nuevas redes.
Protección de la inversión en el desarrollo de nuevos servicios	- Abstrae la lógica de los servicios y minimiza la dependencia en las redes subyacentes.
Diferenciación en la oferta de servicios	- Crear necesidades al cliente de servicios a través de un acceso a una larga y diversa comunidad de desarrolladores, y permitir la modificación y el desarrollo más rápido de los servicios.
Ofrecer servicios rápidamente en respuesta al mercado	- Abre el desarrollo de nuevos servicios a un amplio sector de desarrolladores. - Permite la compra de servicios “off-the-shelf” (disponibles tanto para una compañía como para el público en general) a terceras partes.

Ofrecer servicios personalizados dirigidos tanto a un colectivo determinado, como a un usuario particular	- Deja la organización y el control individual a cargo de sus perfiles de servicio (ej.: a través de un servicio web). - Anima a las terceras partes al desarrollo de nuevos servicios.
Desarrollar servicios nuevos que dirijan la eficiencia de funcionamiento	- Otorga al consumidor la posibilidad de gestionar por sí mismo los servicios de red. - Automatiza servicios que por norma general el proveedor de servicios ofrecía manualmente. - Reduce la dependencia del consumidor en los servicios de soporte manual (servicio técnico).
Desarrollar servicios nuevos e innovadores usando terceras partes	- Proporciona estándares abiertos basados en entornos de ejecución para OSA/Parlay y JSPA. - Abre las redes para que terceras partes puedan desarrollar y ofrecer nuevos servicios. - Minimiza el coste y el riesgo de los proveedores de servicios al desarrollar nuevos servicios no probados.
Desarrollar servicios web, de datos y de voz que converjan	- JAIN SLEE da recomendaciones para la integración de JSLEE con J2EE.

Tabla 3.1: JSLEE haciendo frente a los desafíos de los proveedores de servicios.

3.5. Estándar basado en un entorno de ejecución de lógicas de servicio

Es evidente que se está hablando de JAIN SLEE. La especificación JSLEE define un estándar y el entorno de ejecución de servicios, y especifica como pueden ser creados, gestionados y ejecutados los servicios portables.

La especificación JSLEE ha sido diseñada para soportar servicios de telecomunicaciones. JSLEE proporciona un modelo de programación estándar que puede ser usado por la gran comunidad de desarrolladores Java. Este modelo de programación ha sido diseñado para simplificar el trabajo a los desarrolladores de servicios, eliminar los errores comunes de los programadores y asegurarse de que los servicios son robustos y que pueden crearse rápidamente.

Se sabe que JAIN es tecnología Java, por lo que la larga familia de APIs de Java estándar puede ser aprovechada en las aplicaciones de señalización.

La Figura 3.1 esboza las distintas capas que son especificadas o referenciadas dentro de la especificación JAIN SLEE.

El modelo integrante (component model) especifica como se crea la lógica de servicio, para posteriormente empaquetarla y que pueda recibir eventos externos que darán lugar a la ejecución de los servicios. Por otro lado está la capa de gestión (management layer), que especifica el mecanismo mediante el cual un administrador gestiona un SLEE (incluyendo la suscripción, servicio de instalación, etc.) y permite a los desarrolladores de servicios definir los datos necesitados para un servicio concreto. Este entorno de ejecución SLEE permite colaborar a múltiples componentes de diferentes autores.

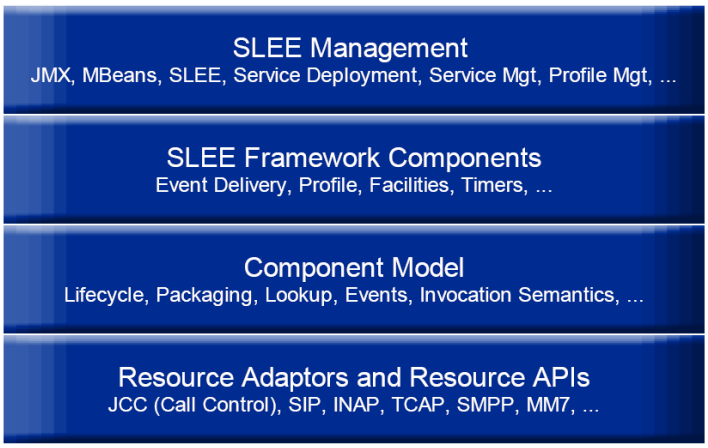


Figura 3.1: Capas de la arquitectura JSLEE.

Los adaptadores de recurso son responsables de la comunicación con un recurso en particular que es externo al modelo de programación de JAIN SLEE, de la comunicación con la lógica de enrutamiento de eventos en la implementación del SLEE y de dotar al programador de aplicaciones de una API adecuada. La arquitectura del adaptador de recurso no está definida en la especificación 1.0 de JSLEE (JSR 22 [17]), pero ya para la versión 1.1 (JSR 240 [18]), que está aún en la fase de borrador, se pretende tener una arquitectura estándar para los adaptadores de recurso.

3.6. Beneficios de JSLEE como entorno de ejecución

Las soluciones JAIN SLEE tienen las siguientes ventajas:

- **Portabilidad de servicios[1]:** JSLEE soporta la filosofía de "*Write Once, Run Anywhere*" ("Escríbelo una vez y ejecútalo donde quieras") del lenguaje de programación Java. Los componentes de las aplicaciones pueden ser desarrollados y luego desplegados en plataformas, que obedezcan a JAIN SLEE, de distintos vendedores y sin tener que recompilar o modificar el código fuente.

[1] Esto a día de hoy todavía no es posible, pero el estándar está concebido para proporcionarlo. Si bien, la revisión de la especificación (JSR 240 [18]) será un paso importante para alcanzar esta meta.

- **Robustez:** El modelo de programación de JAIN SLEE elimina los errores comunes del programador. Esto es debido al uso de un lenguaje fuertemente tipificado y a la adopción de un modelo donde el SLEE es consciente de todo estado relacionado con una llamada o una sesión asignada por la aplicación. La gestión de este estado llega a ser responsabilidad absoluta del SLEE, previniendo errores del nivel de aplicación relacionados con la gestión del estado.
- **Fiabilidad:** JAIN SLEE impone el uso de un modelo de programación transaccional. Con este modelo las aplicaciones tienen semánticas bien definidas bajo condiciones de fallo. Además, este modelo acepta tanto eventos síncronos como asíncronos.
- **Modelo de eventos dinámico y flexible:** Las aplicaciones de señalización hacen un gran uso de las invocaciones asíncronas. JAIN SLEE proporciona un fuerte soporte para las aplicaciones asíncronas con un modelo de eventos dinámico y flexible. Serán los componentes de la aplicación los que reciban eventos de los canales de eventos que se establecen durante el arranque. Los recursos de red, como las pilas de protocolo, crean representaciones de llamadas y emiten eventos generados por las llamadas al SLEE.
- **Arquitectura de componentes orientados a objetos:** JAIN SLEE es la arquitectura de componentes estándar para la creación de aplicaciones de comunicaciones orientadas a objetos distribuidos en el lenguaje Java.
- **Desarrollo de aplicaciones sencillo:** Una implementación JSLEE es responsable de las características del nivel de sistemas, como son la duplicación de memoria, las reanudaciones de los procesos, el gestionar las transacciones y el agrupar la infraestructura. Los desarrolladores de aplicaciones no tienen que entender las transacciones de bajo nivel, ni los detalles de gestión del estado, ni otras APIs de bajo nivel complejas. El código de aplicación basado en JSLEE es sólo responsable de la lógica de aplicación. Por tanto, las aplicaciones basadas en JSLEE han reducido la complejidad, son más sencillas de desarrollar, requieren menos tiempo de desarrollo y presentan un incremento de fiabilidad.
- **Independencia de red:** El modelo de programación JSLEE es independiente de cualquier protocolo de red en particular, API o topología de red. Esto se soporta a través de la arquitectura del adaptador de recurso. Muchas tecnologías pueden ser integradas dentro de un SLEE. Por lo tanto, JSLEE puede ser usado para hacer frente a los problemas de los negocios que implican múltiples redes.
- **Soporta aplicaciones complejas:** Los componentes de las aplicaciones JSLEE pueden tener estado, pueden estar compuestos por otros componentes, pueden crear y destruir otros componentes de aplicaciones, pueden invocar otros componentes de aplicaciones tanto síncronamente como asíncronamente y, por supuesto, pueden invocar adaptadores de

recurso. Estas características permiten el desarrollo modular de aplicaciones complejas.

- **Soporta integración con los sistemas de gestión existentes:** La especificación JSLEE define una API de gestión que permite a un SLEE ser controlado por un sistema de gestión externo, es decir, compatibilidad con JMX, pudiendo esta API ser utilizada como instrumento que puede ser controlado por un servidor JMX.
- **Soporta rich voice[2] y servicios web:** JAIN SLEE permite la interoperabilidad con J2EE, por eso admite estándares basados en la convergencia de voz, y en soluciones de servicios web y de datos.
- **Estándar de la industria:** JAIN SLEE se especifica a través del JCP [22], que permite a múltiples compañías e individualmente colaborar en el desarrollo de las especificaciones de la tecnología Java. Como JSLEE es un estándar, tiene el potencial de atraer a más desarrolladores que cualquier otro entorno de ejecución propietario.

3.7. Integrar JSLEE en el entorno de las telecomunicaciones

Debido a que JSLEE es un motor de procesamiento de eventos basado en un estándar abierto, flexible y de primera calidad, al que se le puede dar uso en numerosos entornos de comunicación, se tiene que:

- Un JSLEE puede ser utilizado en una plataforma de servicios en el sentido tradicional. Sin embargo, es también flexible para poder soportar redes SS7, 3G o NGN.
- Un JSLEE proporciona conexión a sistemas de soporte operativo y de negocios (OSS/BSS). Los servicios se pueden crear para dar eficiencia de funcionamiento al operador, además de interesantes servicios de auto-gestión por parte del consumidor.
- Un JSLEE proporciona un entorno de ejecución abierto externamente, conectado a un entorno seguro del operador a través de una puerta de enlace OSA/Parlay o JSPA (*JAIN Service Provider APIs - APIs del Proveedor de Servicio JAIN*).
- Un JSLEE es capaz de ínter-operar con J2EE, tal que los servicios enriquecidos, basados en la combinación de control de llamada y servicios web, pueden ser desplegados.
- Considerando las capacidades de integración vistas arriba y dependiendo de la perspectiva y de los requisitos, un JSLEE puede ser visto o tratado como un *middleware* o EAI (*Enterprise Application Integration - Integración de Aplicaciones Empresariales*) para las telecomunicaciones.

[2] Rich voice se puede traducir por voz enriquecida, y lo que viene a significar es el simultanear servicios de voz y de datos. Por ejemplo, permitiendo a un usuario móvil hablar mientras ve un documento compartido en la pantalla.

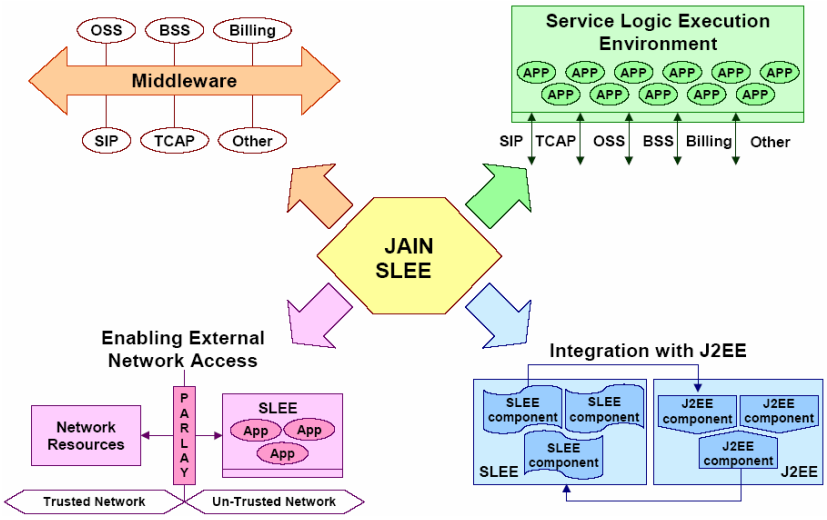


Figura 3.2: JAIN SLEE en un entorno de telecomunicaciones.

3.8. Conceptos fundamentales de la especificación

3.8.1. Introducción

A continuación se hará un breve repaso de los conceptos fundamentales de la especificación JAIN SLEE. Para más información, se puede acudir a la propia especificación [17].

3.8.2. Servicio

Un *servicio* en la terminología JAIN SLEE es una unidad sustituible del campo gestionado. El administrador del sistema de un JAIN SLEE controla el ciclo de vida (incluyendo el despliegue, el deshacer el despliegue y la ampliación o mejora online) de un servicio. La gestión del ciclo de vida se logra a través del uso de las interfaces de gestión estándar proporcionadas por un JAIN SLEE preparado. Un servicio incluye información que describe el servicio, por ejemplo, su nombre, su vendedor y su versión, además de cualquier código del programa que sea parte del servicio. Este código puede incluir clases de Java, Profiles (Perfiles) y SBBs.

3.8.3. Perfil

Un *perfil* de JAIN SLEE contiene datos aprovisionados. Estos datos tienen un esquema, que es un conjunto de propiedades o atributos. Por ejemplo, un número de teléfono y una dirección de correo electrónico de alguien pueden ser considerados dos atributos de un perfil.

JAIN SLEE también define el concepto de la especificación del perfil, la cual incluye interfaces de gestión para modificarlo o actualizarlo, definiendo el esquema de un perfil y la lógica que debe ejecutar para asegurar que éste sea válido. Decir también que los SBBs arrancando en el interior del JSLEE deben acceder a los perfiles como parte de su lógica de aplicación.

3.8.4. Bloque de construcción de servicios

Un *bloque de construcción de servicios* (SBB) es un componente software que envía y recibe eventos, y ejecuta lógica computacional basada en la recepción de eventos y en su estado habitual. Los SBBs son componentes capaces de mantener el estado de un proceso o transacción y, como tal, pueden recordar los resultados de cálculos previos tal que esos resultados puedan ser aplicados en cálculos posteriores. Un SBB puede estar compuesto por otros SBBs, de este modo se permite que aplicaciones más complejas puedan ser creadas a partir de una colección de SBBs.

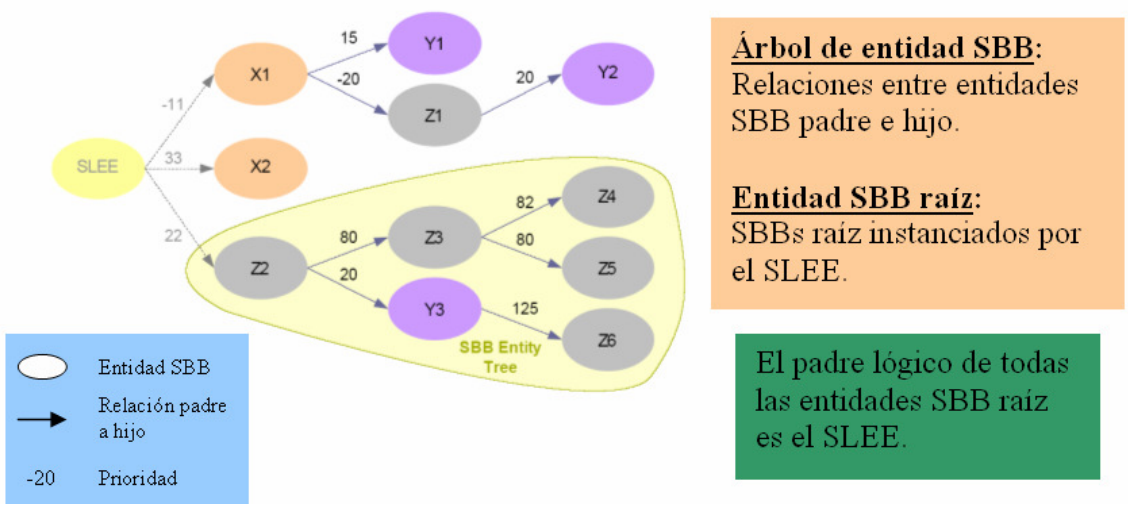


Figura 3.3: Árboles de entidad SBB.

Los SBBs interpretan una lógica basada en la recepción de eventos. Estos eventos se utilizan para representar hechos de importancia que pueden ocurrir en puntos arbitrarios en el tiempo. Por ejemplo, la acción de un sistema externo delegando al SLEE la función de establecer una llamada, puede ocurrir en cualquier momento y, por lo tanto, es fácilmente modelada como un evento. La definición de un SBB incluye información que describe al SBB (por ejemplo, el nombre, el vendedor y la versión del SBB), los eventos que recibe el SBB y el código del programa que proporciona la lógica del SBB. Dicho código para un SBB consta de clases de Java.

3.8.5. Evento

Un *evento* representa una acción que puede requerir un procesamiento de la aplicación. El evento lleva información que describe la acción, como es la fuente del evento. Un evento puede tener su origen en diferentes fuentes, por ejemplo, un recurso

externo como puede ser la pila de un protocolo de comunicaciones, el mismo SLEE o componentes de aplicación dentro del mismo SLEE. Los eventos son una abstracción para modelar acciones que no están relacionadas con ningún hilo de ejecución particular.

Un ejemplo sería una llamada entre dos partes (Rocío y Hugo). Rocío puede llamar a Hugo en cualquier momento. La acción de Rocío marcando el número de teléfono de Hugo puede ser modelada con un evento, y la acción de contestación por parte de Hugo a la llamada puede ser modelada con otro evento.

Dentro del modelo de eventos se puede destacar especialmente la Actividad y el Contexto de Actividad:

- **Actividad:** Una actividad representa una abstracción para un flujo relacionado de eventos. Estos eventos representan acciones de importancia que han ocurrido en la entidad representada por la actividad.

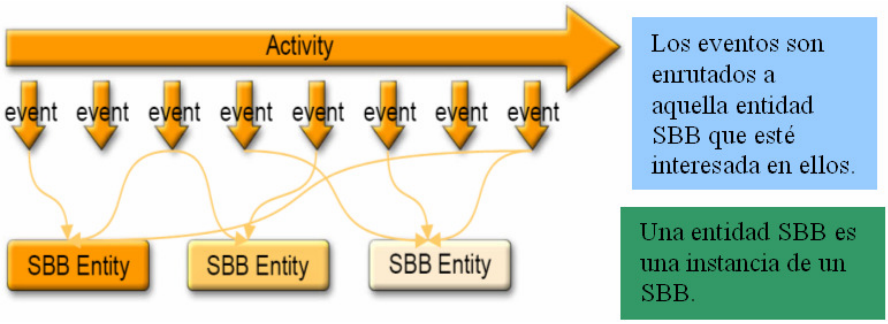


Figura 3.4: Actividad.

- **Contexto de Actividad:** El SLEE utiliza un contexto de actividad para representar y encapsular un objeto de actividad (objeto Java que encapsula una actividad) subyacente dentro del SLEE. Un contexto de actividad es una entidad lógica dentro del SLEE. Además, existe una relación uno a uno entre un objeto de actividad y un contexto de actividad.

Figura 3.5: Contexto de Actividad.

La importancia de los contextos de actividad y el hecho de que una entidad SBB solamente pueda recibir eventos disparados en contextos de actividad a los que esté unido, radica en que si, por ejemplo, un SBB dispara un evento dirigido a otro SBB, lo hará a través de un contexto de actividad que estará unido al SBB destino y, por tanto, se evita que ese evento llegue a todos los SBBs que no están implicados, pero que están a la "escucha" del mismo tipo de evento. Se tiene, de esta forma, un mayor control sobre los eventos.

3.8.6. Recursos y adaptadores de recurso

Los *recursos* representan a entidades externas que interactúan con otros sistemas fuera del SLEE, como son los elementos de red (HLR, MSC, etc.), las pilas de protocolo, directorios y bases de datos. Un *adaptador de recurso* adapta los requisitos y las interfaces particulares de un recurso a los requisitos e interfaces del JAIN SLEE.

JSLEE es un servidor de aplicación que está basado en componentes, los llamados Bloques de Construcción de Servicios (SBB). El servidor de aplicación incluye un modelo de aplicación que es independiente a cualquier protocolo de red o fuente de eventos utilizada para disparar la ejecución de la lógica de tareas en clave. Los eventos en el modelo de aplicación JSLEE son POJOs (*Plain Old Java Objetct - Antiguo Objeto Java Plano*) y necesitan ser creados en algún sitio. El Adaptador de Recurso (RA) en JSLEE tiende un puente entre el modelo de aplicación y la estructura de eventos subyacente. La fuente de eventos puede ser cualquier cosa emitiendo eventos, implementada en cualquier lenguaje y entorno. Ejemplos de fuentes de eventos son una pila SIP, JCC, TCP/IP o incluso una pila HTTP. El RA acepta la llegada de señales de protocolo o eventos específicos, crea las representaciones Java y los dispara dentro del servidor de aplicación JSLEE.

Finalmente, se verá cual es la estructura de un Adaptador de Recurso. Un JSLEE RA consiste básicamente en un Resource Adaptor Type (Tipo de Adaptador de Recurso) y en un Resource Adaptor (Adaptador de Recurso). El RA Type especifica los eventos emitidos y la interfaz utilizada por los SBBs para acceder a funciones del RA. Normalmente, los RA Types están definidos por una industria que tenga los mismos intereses. Buenos ejemplos de esto, en la industria de las telecomunicaciones, son el Call Control RA Type y el SIP RA Type, es decir, las industrias crean sus RA Types según sus necesidades. Por otro lado se tiene el RA en sí mismo, que lo que hace es implementar exactamente un RA Type. Lo usual es que un RA sea proporcionado bien por un vendedor de recursos, o bien por un vendedor de SLEE para adaptar una implementación de un recurso particular al SLEE, como puede ser, por ejemplo, una pila SIP, siendo el administrador el que se encargará de instalar los RAs en el SLEE[3].

[3] Todo esto está dicho siguiendo la especificación actual (JSR 22 [17]). Sin embargo, ya en el borrador de la nueva especificación (JSR 240 [18]) se pueden apreciar los cambios realizados para conseguir la estandarización de los adaptadores de recurso.

Resumiendo, el RA Type define el tipo de RA, la implementación del RA envuelve una pila de protocolo específica y emite objetos Java como eventos dentro del servidor de aplicación JSLEE, y el Contexto de Actividad se establece para intercambiar la información de estado entre el RA y los SBBs.

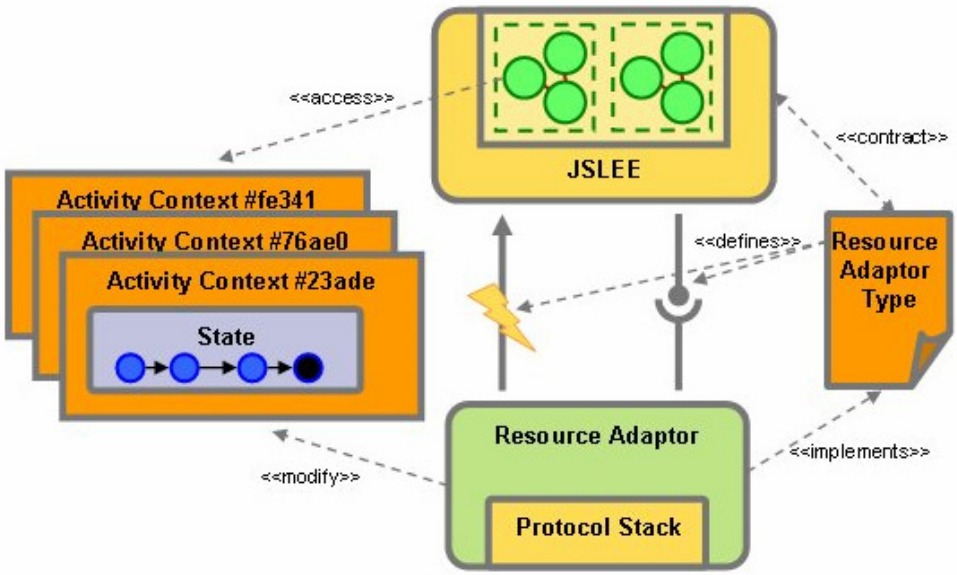


Figura 3.6: RA, RA Type, pila de protocolo, contexto de actividad y JSLEE.

3.9. Plataformas JAIN SLEE

Las plataformas JAIN SLEE utilizadas en el proyecto fueron Rhino de Open Cloud y Mobicents.

3.9.1. Plataforma Rhino

3.9.1.1. Introducción

La plataforma Rhino SLEE consiste en un conjunto formado por herramientas, servidores y ejemplos que soportan la especificación JAIN SLEE 1.0 [17]. Proporciona el camino crítico para construir y desplegar componentes SLEE en una infraestructura de red física crítica, con focalización en la puesta a punto de la tolerancia a fallos y alta disponibilidad en entornos demandados.

Los elementos de la plataforma pueden ser organizados en las siguientes categorías, como se muestra en la Figura 3.7:

- Entorno de Ejecución de Lógicas de Servicio (SLEE).
- Integración.

- Desarrollo de Servicio.

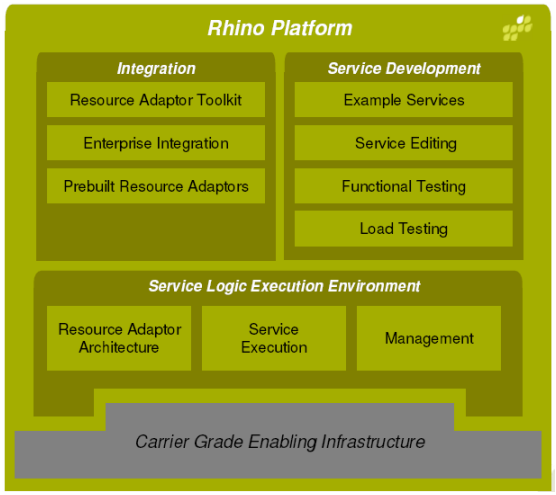


Figura 3.7: Plataforma Rhino.

Algunas características clave del Rhino SDK SLEE son:

- Proporciona un alto rendimiento y una baja latencia en el entorno de ejecución de lógicas de servicio.
- El Rhino SDK SLEE arranca como un servidor de un único nodo, con el que es más sencillo trabajar para el desarrollo.
- Especificación JAIN SLEE 1.0, conforme a la JSR 22 [17].
- Ejemplos de servicios para habilitar un rápido desarrollo de aplicaciones.
- Los Adaptadores de Recurso son proporcionados por defecto para reducir el tiempo de puesta en el mercado.
- El Entorno Federado de Creación de Servicios permite a los desarrolladores construir servicios usando los conjuntos de herramientas existentes.

El Rhino SDK SLEE está pensado para soportar el desarrollo de servicios y pruebas funcionales, pero no es adecuado para pruebas de carga, de fallos o desarrollo en un entorno de producción.

3.9.1.2. Categoría de entorno de ejecución de lógicas de servicio

La categoría de Entorno de Ejecución de Lógicas de Servicio (SLEE) incluye:

- **El servidor Rhino SLEE:** Éste incluye la implementación del modelo SLEE, el procesamiento de eventos del núcleo y el software de gestión de configuración.

- **Algunas herramientas de gestión:** Éstas incluyen aplicaciones cliente que facilitan la gestión del cliente Rhino SLEE, como pueden ser la Consola Web y la Consola de Comandos. Estas herramientas son usadas por administradores de sistemas para desplegar y gestionar servicios y perfiles.
- **Arquitectura del Adaptador de Recurso:** Habilita la integración con servidores críticos de red y otros sistemas externos, y además permite a los Adaptadores de Recurso el poder ser desplegados y configurados independientemente de los servicios ejecutados en el SLEE.

Añadir que el Rhino SDK SLEE utiliza la infraestructura Carrier Grade (Calidad Portadora) de Open Cloud, que asegura que la ejecución de las lógicas de servicio y la función de la gestión online tienen un alto grado de disponibilidad ante posibles fallos.

3.9.1.3. Categoría de integración

La categoría de integración incluye:

- Adaptadores de Recurso por defecto para la integración con sistemas externos comunes, por ejemplo, SIP y JCC.
- Herramientas para un rápido desarrollo de nuevos Adaptadores de Recurso y Componentes de Servicio.
- Integración con los servidores de base de datos.
- Integración de seguridad con los servidores web J2EE y los sistemas de directorio LDAP (*Lightweight Directory Access Protocol - Protocolo Ligero de Acceso a Directorios*).
- Comunicaciones duplex con servidores de aplicación J2EE.

3.9.1.4. Kit de desarrollo software Rhino SLEE

Se ha hecho uso del Open Cloud SDK SLEE (Figura 3.8), que es una solución JAIN SLEE para el desarrollo de servicios e incluye:

- Todo el software en la categoría SLEE.
- Adaptador de Recurso SIP.
- Servicios de demostración de SIP: *Registrar*, *Proxy*, *Find-me-follow-me*.
- Adaptador de Recurso JCC.
- Servicios de demostración de JCC: Call forwarding (desvío de llamada).
- Características de Integración de Empresa.

- Ejemplo de aplicaciones SIP y JCC.

Destacar que la licencia de evaluación distribuida con el Rhino SDK SLEE limita el movimiento máximo de eventos por segundo, y hay que tener en cuenta que algunas veces una llamada puede involucrar más de un único evento.

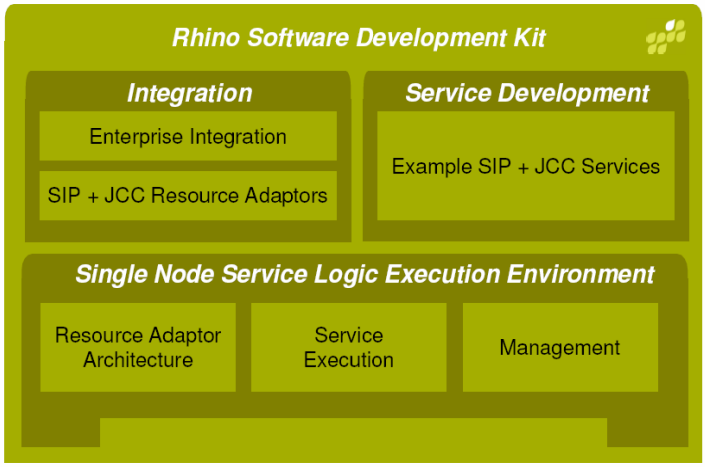


Figura 3.8: El Rhino SDK SLEE de Open Cloud.

3.9.2. Plataforma Mobicents

3.9.2.1. Introducción

La plataforma Mobicents posibilita la composición de Bloques de Construcción de Servicios (SBB) como son el control de llamada, facturación, distribución de usuarios, administración y las características de presencia sensible. La especificación JSLEE permite a las pilas de protocolo comunes, como por ejemplo la pila SIP, ser "enchufadas" como adaptadores de recurso. Además, los SBBs del SLEE tienen muchas similitudes con los EJBs, y la arquitectura estándar extensible realiza con naturalidad la acomodación de puntos de integración con aplicaciones de empresa como son Web, CRM (*Customer Relationship Management - Gestión de las Relaciones con el Cliente*) o puntos terminales SOA (*Service Oriented Architecture - Arquitectura Orientada a Servicios*).

La monitorización y gestión de los componentes Mobicents se hace desde fuera, vía el estándar SLEE, a través de las interfaces de gestión JMX y SNMP. Esto hace de los servidores Mobicents una fácil elección, dentro de las telecomunicaciones, para los OSS (*Operations Support Systems - Sistemas de Soporte de Operaciones*) y los NMS (*Network Management Systems - Sistemas de Gestión de Red*).

Más allá de las telecomunicaciones, Mobicents es aplicable a una amplia gama de situaciones de demanda de un alto volumen y una baja latencia de señalización.

Como ejemplos de esto se puede mencionar el comercio financiero, juegos online, control distribuido, etc.

Finalmente, además de una plataforma VoIP de código abierto sólida, el proyecto Mobicents proporciona el EclipSLEE (JSLEE plugin para Eclipse, Figura 3.9), que es una herramienta de desarrollo que simplifica mucho la creación de nuevas aplicaciones SLEE.

Figura 3.9: EclipSLEE

3.9.2.2. Servidor Mobicents

El Servidor Mobicents incluye un motor de núcleo JAIN SLEE certificado, así como varias extensiones como pueden ser el Adaptador de Recurso SIP, el servicio de registro SIP y el servicio de proxy SIP.

3.9.2.3. Arquitectura Mobicents

Mobicents tiene una arquitectura muy flexible y escalable basada en el estándar que obedece a JAIN SLEE. Los siguientes diagramas ilustran dos puntos de vista diferentes de la arquitectura de Mobicents (Figura 3.10 y Figura 3.11):

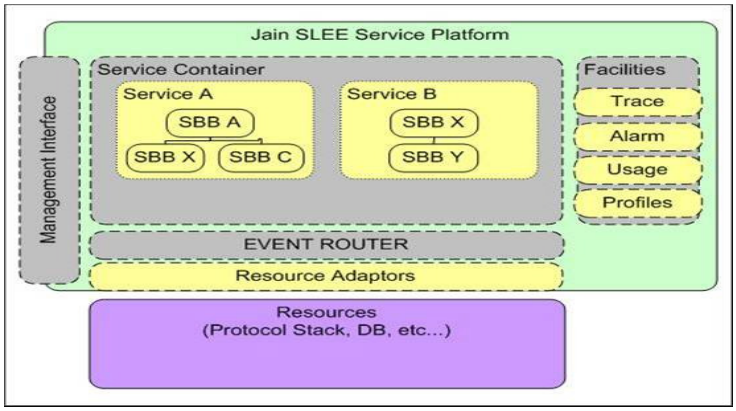


Figura 3.10: Arquitectura JSLEE simplificada 1.

Hasta el momento, Mobicents proporciona los Adaptadores de Recurso SIP, XMPP (*Extensible Messaging and Presence Protocol - Protocolo de Presencia y Mensajería Extensible*) [23], Asterisk [24] y JCA (*J2EE Connector Architecture - Arquitectura del Conector J2EE*) [25], los cuales permiten a las aplicaciones SLEE interactuar con fuentes externas soportando uno de los protocolos implementados.

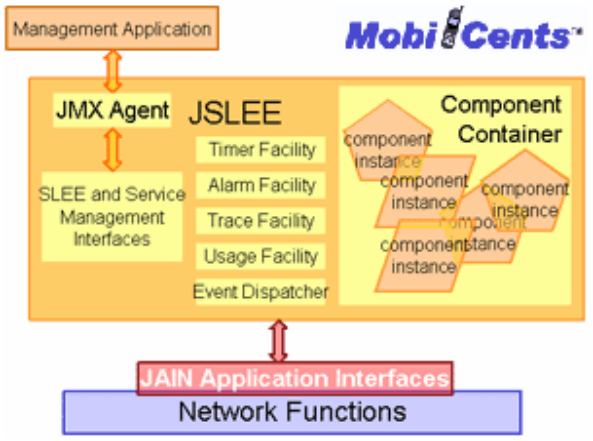


Figura 3.11: Arquitectura JSLEE simplificada 2.

Añadir que Mobicents se encuentra actualmente construido sobre JBoss 3.2.6, y los componentes que usa de él son:

- Microkernel JMX para IoC (*Inversion of Control - Control de Inversión*) y gestión del servicio de núcleo.
- JNDI (*Java Naming Directory Interface - Sistema de Nombrado en Java*) para el registro de servicio SLEE.
- JTA (*Java Transaction API - API de Transacción Java*) para Gestión de Transacción.
- TreeCache para repetición del estado de Alta Disponibilidad.

Destacar que Mobicents no usa EJB ni JMS (*Java Message Service - Servicio de Mensajes Java*) como parte de la arquitectura de núcleo.

3.9.2.4. Beneficios

En este apartado no se van a comentar los beneficios de Mobicents desde el punto de vista de la tecnología JSLEE, sino que se va a tratar aquello que sin duda es la principal ventaja de esta plataforma con respecto a las demás, y que no es otra que su desarrollo en código libre.

Mobicents es un proyecto de java.net [1], es decir, está ubicado en la comunidad de colaboración para la tecnología Java. Por tanto, goza de la colaboración de todos los desarrolladores Java del mundo que quieran incorporarse a este interesante proyecto.

Por otra parte, cualquiera que demuestre tener el conocimiento necesario del código base y de los tests de compatibilidad tecnológica, mediante la aportación de páginas *wiki* (considerado como aportación valiosa) y de parches, tanto para arreglar errores como para realizar mejoras, podrá pasar a formar parte del equipo contribuidor. Todo esto trae consigo una evolución de la tecnología no sólo más rápida, sino también más independiente y por supuesto más imaginativa.

Por suerte, ya se forma parte del llamado “core team”, gracias a la aportación de mejoras como son la actualización de una página *wiki*, parches para el arranque del servidor, aportaciones en el foro del proyecto, además de otra serie de cuestiones en las que se está trabajando y en las que se piensa seguir tras la conclusión de este proyecto. De todas formas, estos aspectos serán vistos con más detalle en el capítulo 4 referente al desarrollo del proyecto.

Capítulo 4
Trabajo desarrollado

4. Trabajo desarrollado.

4.1. Introducción

Una vez vistos los capítulos más teóricos de la memoria, ya se tiene la base suficiente para poder afrontar sin temor alguno los aspectos más prácticos. Como ya se sabe, se ha hecho uso de dos plataformas JSLEE distintas (Rhino y Mobicents). Lo que se hará en los apartados sucesivos es contar cómo se ha ido desarrollando el proyecto a lo largo del tiempo y cómo se han ido cumpliendo los distintos objetivos:

- Red VoIP sobre JSLEE.
- Servicio de telefonía sobre la red VoIP.
- Realizar algún tipo de aportación.

4.2. Organización

Después de tener en cuenta las distintas alternativas posibles para explicar el trabajo desarrollado, se considera que la mejor forma de hacerlo es ir nombrando los distintos objetivos pedidos a lo largo de la elaboración del proyecto, y a partir de ahí comenzar a detallar los pasos que se han seguido para cumplir cada uno de dichos objetivos.

Aunque pueda resultar extraña esta forma de estructurar el capítulo, el tener la posibilidad de consultar cada objetivo y poder ver los distintos pasos a seguir para su culminación con éxito, es sin duda la forma más eficaz y amena de organizar un proyecto de estas características.

Sin más dilación se pasa a ver los objetivos del proyecto.

4.3. Objetivos

Antes de comenzar a describir los objetivos, sería bueno aclarar que cuando se comenzó a desarrollar este proyecto, aunque Mobicents ya había anunciado la existencia de su plataforma, ésta todavía no estaba lo suficientemente desarrollada como para alcanzar los objetivos pedidos, de ahí que al principio sólo se haga mención a la plataforma Rhino de Open Cloud.

1. **Red VoIP sobre JSLEE:** Montaje de una pequeña red formada por dos clientes y un servidor.
2. **Servicio de telefonía sobre la red VoIP:** Tras la consecución del primer objetivo se incorpora un servicio similar al del desvío de llamadas.

3. **Realizar algún tipo de aportación:** Mejora del proyecto Mobicents colaborando en su desarrollo.

4.4. Red VoIP sobre JSLEE

Este primer objetivo consistía en hacer o simular una pequeña red VoIP y utilizar JSLEE para controlarla.

El primer paso era evidente, había que instalar un SLEE. La única plataforma de la que se disponía era la perteneciente a Open Cloud [4] (Rhino), ya que aunque jNetX [2] también proporcionaba una en esos momentos, no disponía ni dispone actualmente de ninguna versión de acceso libre.

En definitiva, habrá que cumplir una serie de requisitos generales para la consecución de este objetivo, que se pasan a enumerar en el siguiente punto.

4.4.1. Requisitos

En un principio, para poder llevar acabo este primer objetivo se necesita lo siguiente:

1. Instalar un SLEE, a través del cual se gestionará la red.
2. Un servicio de proxy, otro de registro y otro de localización, para poder gestionar y encaminar las distintas llamadas, acompañado de una base de datos que facilite dicha gestión.
3. Un adaptador de recurso SIP, que se encargue de recibir los eventos SIP y sepa que hacer con ellos, según se ha explicado en los capítulos anteriores.
4. Clientes SIP, que serán los usuarios de la red VoIP.

4.4.2. Instalación del SLEE

Se comienza descargando el Open Cloud Rhino SLEE SDK 1.4.0 de su sitio web [4], aunque finalmente se terminó usando la versión 1.4.1 que salió unos meses más tarde. Este paquete incluye lo mismo. Sin embargo, como toda nueva versión, corrige algunos errores de la versión anterior.

Dicho esto, el paquete RhinoSDK-1.4.1-ga.tar incluye lo siguiente:

- Código binario de un nodo servidor Rhino SLEE.
- Código binario del adaptador de recurso SIP y JCC.

- Código fuente de ejemplos de servicios SIP y JCC.
- Un conector J2EE y un adaptador de recurso para la integración de J2EE con el SLEE.
- Cliente de supervisión de estadísticas de Rhino.
- Un *script* para instalación automática.
- Documentación.
- Licencia de limitación a 500 eventos por segundo en la tasa de transferencia (throughput).

Dentro de los servicios y componentes SIP que están incluidos con el Open Cloud Rhino SLEE, se comentarán los que harán falta para esta aplicación:

- **SIP Resource Adaptor:** El Adaptador de Recurso SIP (SIP RA) proporciona la interfaz entre la pila SIP y el SLEE. La pila SIP es la responsable de enviar y recibir mensajes SIP sobre la red (normalmente UDP/IP). El SIP RA procesa los mensajes que le lleguan de la pila y los mapea a las actividades y eventos, como es requerido según el modelo de programación del SLEE. Además, el SIP RA debe ser instalado en el SLEE antes de que otras aplicaciones SIP puedan ser utilizadas.
- **SIP Registrar Service:** Esto es una implementación de un *SIP Registrar* a partir de la definición que aparece en la sección 10 de la RFC3261 [12]. Este servicio maneja peticiones de registro (SIP REGISTER), que se envían mediante agentes de usuario SIP para registrar la atadura (binding) de una dirección pública de un usuario con la dirección de red física de su agente de usuario. El *Registrar Service* actualiza registros en el *Location Service* que es utilizado por otras aplicaciones SIP. El *Registrar Service* es implementado utilizando un único componente SBB en el SLEE.
- **SIP Stateful Proxy Service:** Este servicio implementa un *proxy* tal y como se descibre en la sección 16 de la RFC3261 [12]. Este *proxy* es el responsable de enrutar peticiones hacia sus destinos correctos, dados por direcciones de contacto que han sido registradas con el *Location Service*, es decir, el *Registrar Service* (*RegistrarSbb.java*) comprueba que tipo de servicio de localización se ha elegido (*getLocationService*), si un *AC Naming* o un *JDBC Location Service*, y según eso hace el registro. El *Proxy Service* es implementado usando un único SBB, y también pasa las peticiones REGISTER al *Registrar SBB* usando una relación hijo[4].
- **AC Naming & JDBC Location Service SBBs:** Estos SBBs proporcionan implementaciones alternativas del *SIP Location Service* que es usado por los servicios *Proxy* y *Registrar*. Por defecto es

[4] Esto es lo que dice en el manual de Rhino. Sin embargo, si se observa como está realmente implementado, se podrá comprobar que es el propio *Registrar SBB* el que está preparado para recibir las peticiones REGISTRAR, mientras que en el *Proxy SBB* está simplemente comentado, siendo además la única relación hijo (child-relation) la existente con el *Location SBB*.

desplegado el *AC Naming Location Service*, el cual usa una facilidad *AC Naming* del SLEE para almacenar la información de localización, es decir, cada registro se almacena usando un Contexto de Actividad (AC) que está ligado a un nombre en la *AC Naming Facility*. Alternativamente se tiene el *JDBC Location Service*, que almacena la información de localización en una base de datos externa.

Antes de comenzar con la instalación del Rhino SDK SLEE habrá que asegurarse de que se cumplen una serie de requsitos previos (9.1).

Tras seguir los pasos de instalación del Rhino, los cuales pueden verse en el Anexo A.1, se puede comenzar con la construcción y el despliegue de los servicios necesarios para alcanzar el primer objetivo.

4.4.3. Construcción y despliegue de los servicios

Para llevarlo a cabo habrá que utilizar la herramienta Ant [28]. Esta herramienta se sirve de un fichero constructor llamado *build.xml*, el cual tiene marcados una serie de objetivos y como realizarlos. Pues bien, con simplemente indicar el objetivo (target) que se quiere alcanzar, se irán ejecutando los distintos comandos marcados en el fichero constructor para lograr dicho objetivo.

Dicho esto, en la ruta *$RHINO_HOME*[5]*/examples/sip* se tiene todo lo que hará falta para desarrollar los servicios SIP, es decir, se dispone del fichero constructor *build.xml*, los ficheros de configuración *build.properties* y *sip.properties* y los servicios *proxy*, *registrar* y *location* que serán los necesarios para el montaje de la red VoIP, además de todo lo concerniente a la pila y al adaptador de recurso SIP, que se encuentra en la ruta *$RHINO_HOME/examples/sip/lib*.

Antes de desplegar los servicios, se repasarán los ficheros de configuración y se harán las modificaciones oportunas.

build.properties

```
# Location of Rhino client install
client.home=../../client

# Select location service implementation.
# If "usejdbclocation" property is true, JDBC location service will be deployed.
# Default is to use Activity Context Naming implementation.
usejdbclocation=true

# Specify SIP RA config properties here, e.g. ListeningPoints=10.0.0.1:5070/udp
sip.ra.properties=ListeningPoints=0.0.0.0:5060/udp;0.0.0.0:5060/tcp

# Default SBB trace level
# Possible values: Off, Severe, Warning, Config, Info, Fine, Finer, Finest.
sbb.tracelevel=Finest
```

Figura 4.1: Fichero de configuración *build.properties*.

[5] Se entiende por *$RHINO_HOME* la ruta donde se tiene instalado el Rhino SDK SLEE. En este caso, sería la siguiente: */home/vhros/PFC/rhino*.

En el fichero *build.properties* (Figura 4.1) se han hecho dos modificaciones:

- La primera ha consistido en poner la variable *"usejdbclocation"* a valor *true*. De esta forma se consigue que se despliegue el servicio de localización JDBC y no el que viene por defecto (mirar el punto 4.4.2 en la parte final que hace referencia a los servicios), y así poder utilizar la base de datos para gestionar los registros. El uso de esta variable se puede ver en la Figura 4.2, donde se muestran las líneas del fichero constructor *build.xml* en las que se decide que servicio va a ser utilizado.

```
<!-- Setup properties for selecting the location service to use, based
     on the "usejdbclocation" boolean property. -->
<condition property="jdbc.location.selected">
  <istrue value="${usejdbclocation}"/>
</condition>
<condition property="locationsbb" value="JDBCLocationSbb">
  <isset property="jdbc.location.selected"/>
</condition>
<condition property="locationsbb" value="ACLocationSbb">
  <not><isset property="jdbc.location.selected"/></not>
</condition>
```

Figura 4.2: Selección del servicio de localización.

- La segunda simplemente ha consistido en indicar el nivel de las trazas del SBB, es decir, a que nivel de detalle se quieren ver las trazas de los SBBs en el shell donde se está arrancando el SLEE. Pues bien, se ha elegido el nivel *Finest* por ser el más detallado.

sip.properties

```
# This file contains properties that will be substituted
# into SBB deployment descriptors when building with Ant.

# Proxy SBB configuration
# Add names that the proxy host is known by. The first name in the list
# will be treated as the Proxy's canonical hostname and will be used in
# Via and Record-Route headers inserted by the proxy.
PROXY_HOSTNAMES=rhinoslee.proxy.sip,localhost,127.0.0.1
# Add domains that the proxy is authoritative for
PROXY_DOMAINS=rhinoslee.proxy.sip
PROXY_SIP_PORT=5060
PROXY_SIPS_PORT=5061
PROXY_LOOP_DETECTION=true
```

Figura 4.3: Fichero de configuración *sip.properties*.

En el fichero *sip.properties* (Figura 4.3) se han hecho otras dos modificaciones, aunque en este caso ambas están relacionadas entre sí:

- La variable *PROXY_HOSTNAMES* alberga la lista de nombres por los que es conocido el *host* del *proxy*. El primer nombre de la lista será el hostname canónico del *proxy*, y será usado en las cabeceras *Via* y *Record-Route* insertadas por el *proxy*.

```
Via: SIP/2.0/UDP rhinoslee.proxy.sip:5060;branch=z9hG4bK0df540632ad8f8999be310e65a0ef6d4,SIP/2.0/UDP 172.16.17.225:5060;rport
;branch=z9hG4bK489017519
From: <sip:jain1@rhinoslee.proxy.sip>;tag=466764781
To: <sip:jain2@rhinoslee.proxy.sip>;tag=1831830625
Call-ID: 916214943@172.16.17.225
CSeq: 20 ACK
Contact: <sip:jain1@172.16.17.225:5060>
Max-Forwards: 4
User-Agent: Linphone-1.1.0/eXosip
Content-Length: 0
```

Figura 4.4: Cabecera *Via* insertada por el *proxy*.

- La variable *PROXY_DOMAINS* alberga la lista de dominios a los que el *proxy* está autorizado, es decir, para cualquier petición recibida para dirigirse a un usuario de alguno de estos dominios, el *proxy* intentará mapear la URI (*Uniform Resource Identifier - Identificador Uniforme de Recursos*) a una dirección de contacto registrada y localizada, usando el SBB correspondiente al servicio de localización. Sin embargo, peticiones para usuarios en otros dominios serán direccionadas de acuerdo a las reglas normales de rutado SIP.

```
2005-12-18 14:06:15.405  Finer  [notificationrecorder.trace]  (36) Sbb[ProxySbb 1.5, Open Cloud] validateRequest
2005-12-18 14:06:15.406  Finer  [notificationrecorder.trace]  (37) Sbb[ProxySbb 1.5, Open Cloud] checkMaxForwards: OK
2005-12-18 14:06:15.459  Finer  [notificationrecorder.trace]  (38) Sbb[ProxySbb 1.5, Open Cloud] loopDetection: OK, no loop d
etected
2005-12-18 14:06:15.460  Finer  [notificationrecorder.trace]  (39) Sbb[ProxySbb 1.5, Open Cloud] determineRequestTargets: sip
:jain2@pfc.esi is outside our domain, forwarding
2005-12-18 14:06:15.464  Fine   [notificationrecorder.trace]  (40) Sbb[ProxySbb 1.5, Open Cloud] forwarding: INVITE to target
: sip:jain2@pfc.esi
2005-12-18 14:06:15.464  Finest [notificationrecorder.trace]  (41) Sbb[ProxySbb 1.5, Open Cloud] forwarding request:
INVITE sip:jain2@pfc.esi SIP/2.0
```

Figura 4.5: Usuario fuera del dominio.

Dicho esto, el hecho de haber puesto *rhinoslee.proxy.sip* en ambas variables, es simplemente porque ese ha sido el nombre que se ha elegido para el *proxy* y como dominio de los clientes SIP. Se podría haber utilizado algún otro dominio (ej.: *pfc.esi*). Sin embargo, hay un inconveniente con la versión del Linphone utilizada, y es que no existe la opción de *outbound proxy*, es decir, que toda llamada (independientemente del dominio) se envíe al *proxy* especificado.

sip.properties

```
# This file contains properties that will be substituted
# into SBB deployment descriptors when building with Ant.

# Proxy SBB configuration
# Add names that the proxy host is known by. The first name in the list
# will be treated as the Proxy's canonical hostname and will be used in
# Via and Record-Route headers inserted by the proxy.
PROXY_HOSTNAMES=rhinoslee.proxy.sip,localhost,127.0.0.1
# Add domains that the proxy is authoritative for
PROXY_DOMAINS=rhinoslee.proxy.sip,pfc.esi
PROXY_SIP_PORT=5060
PROXY_SIPS_PORT=5061
PROXY_LOOP_DETECTION=true
```

Figura 4.6: Nuevo dominio autorizado por el *proxy*.

Imaginar que se añade a la variable *PROXY_DOMAINS* el dominio *pfc.esi* (Figura 4.6), y que el cliente jain2 se registra como *sip:jain2@pfc.esi*.

Pues bien, si desde el cliente jain1 se intentase llamar a jain2, la llamada no tendría éxito, ya que el cliente Linphone intentaría enviar esa llamada a la dirección *pfc.esi*, y esa dirección no existe y tampoco se tiene en el fichero */etc/hosts*. Sin embargo, si existiese la opción de *outbound proxy*, la llamada la dirigiría al *proxy sip:rhinoslee.proxy.sip* y al pertenecer el destino a uno de los dominios a los que está autorizado dicho *proxy*, todo funcionaría normalmente[6].

```
2005-12-19 10:40:13.330  Finer  [notificationrecorder.trace]  (3600) Sbb[ProxySbb 1.5, Open Cloud] validateRequest
2005-12-19 10:40:13.330  Finer  [notificationrecorder.trace]  (3601) Sbb[ProxySbb 1.5, Open Cloud] checkMaxForwards: OK
2005-12-19 10:40:13.330  Finer  [notificationrecorder.trace]  (3602) Sbb[ProxySbb 1.5, Open Cloud] loopDetection: OK, no loop
 detected
2005-12-19 10:40:13.382  Finer  [notificationrecorder.trace]  (3603) Sbb[ProxySbb 1.5, Open Cloud] determineRequestTargets: s
ip:jain2@pfc.esi is local
2005-12-19 10:40:13.399  Finest [notificationrecorder.trace]  (3604) Sbb[JDBCLocationSbb 1.5, Open Cloud] read record for sip
:jain2@pfc.esi/sip:jain2@172.16.17.218:5060
2005-12-19 10:40:13.411  Finer  [notificationrecorder.trace]  (3605) Sbb[ProxySbb 1.5, Open Cloud] determineRequestTargets: t
arget set for sip:jain2@pfc.esi: [sip:jain2@172.16.17.218:5060]
2005-12-19 10:40:13.412  Fine   [notificationrecorder.trace]  (3606) Sbb[ProxySbb 1.5, Open Cloud] forwarding: INVITE to targ
et: sip:jain2@172.16.17.218:5060
2005-12-19 10:40:13.412  Finest [notificationrecorder.trace]  (3607) Sbb[ProxySbb 1.5, Open Cloud] forwarding request:
INVITE sip:jain2@172.16.17.218:5060 SIP/2.0
Via: SIP/2.0/UDP rhinoslee.proxy.sip:5060;branch=z9hG4bK66943be9e06676032c718f8d842789ae,SIP/2.0/UDP 172.16.17.179:5061;rport
;branch=z9hG4bK1907995260
From: <sip:jain1@rhinoslee.proxy.sip>;tag=7583399
To: <sip:jain2@pfc.esi>
```

Figura 4.7: Usuario dentro del dominio.

Así pues, la configuración elegida para el fichero */etc/hosts* es la siguiente:

```
127.0.0.1          localhost.localdomain          localhost
172.16.17.179      rhinoslee.proxy.sip
```

La dirección 172.16.17.179 es la correspondiente al puesto de trabajo donde se hizo la instalación del Rhino SLEE y, por tanto, donde se desplegó el *proxy*. Esta configuración fue previa a la instalación del SLEE, de ahí que las últimas líneas correspondientes al fichero de configuración *$RHINO_HOME/config/config_variables* tengan el siguiente aspecto:

```
WEB_CONSOLE_HOSTNAME=rhinoslee.proxy.sip
RHINO_PASSWORD=password
RHINO_USERNAME=admin
LOCALIPS="[0:0:0:0:0:0:0:1%1] 127.0.0.1 [fe80:0:0:0:20f:b0ff:fe6b:4120%2] 172.16.17.179"
RHINO_WATCHDOG_DUMPTHREADS=/home/vhros/PFC/rhino/dumpthreads.sh
```

Figura 4.8: Fichero de configuración *config_variables*.

Por este motivo, el certificado creado para la consola web tiene constancia de que la máquina que alberga dicha consola es *rhinoslee.proxy.sip* y no el habitual *localhost*. Se podrá utilizar para acceder a la consola (siempre que se tengan los

[6] Para hacer esta prueba, que se ve reflejada en la Figura 4.7, se hizo uso del cliente SJphone [29] que sí incorpora la función de *outbound proxy*. Sin embargo, no tiene la opción de chat. También se podría haber usado el Linphone, pero habría que cambiar el fichero */etc/hosts*: "172.16.17.179 rhinoslee.proxy.sip *pfc.esi*", simulando de esta forma la función de *outbound proxy*.

permisos[7]) cualquier dirección distinta a *https://rhinoslee.proxy.sip:8443/*, siempre que la dirección de la máquina se corresponda con el *host* donde está arrancando el SLEE. Sin embargo, llegará un aviso de certificado, ya que se está intentando acceder a una máquina, por ejemplo, 127.0.0.1 (si se está accediendo desde la misma máquina) que en realidad se llama *rhinoslee.proxy.sip*.

Finalmente, se hará el despliegue de los servicios. Para ello se utilizará la herramienta Ant [28], tal y como se mencionó anteriormente. Así pues, habrá que situarse en la ruta que alberga al *build.xml* correspondiente a los servicios SIP y se ejecutará el Ant indicando el objetivo *deployexamples*, que es el que interesa para el servicio que se quiere desplegar.

```
[vhros@rhinoslee sip]$ ant deployexamples
```

No haría falta escribir *deployexamples*, ya que se puede ver en el fichero constructor (Figura 4.9) como ese es el objetivo marcado por defecto.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Ant script for SIP examples management. -->
<project name="Open Cloud Rhino SLEE SDK - SIP Examples" default="deployexamples">

  <property file="${basedir}/build.properties"/>
  <property file="${basedir}/sip.properties"/>
```

Figura 4.9: Objetivo por defecto del constructor de servicios SIP.

4.4.4. Instalación de los clientes SIP

A continuación se realizará la instalación de los clientes SIP. Se ha elegido para este cometido el cliente Linphone [30]. Linphone es un teléfono web con una interfaz GNOME que permite realizar fácil y eficientemente llamadas de voz por IP, a través de Internet o LANs (*Local Area Network - Red de Área Local*), entre estaciones GNU/Linux y hacia otros *webphones* compatibles con SIP.

Para una instalación correcta del Linphone, habrá que tener una serie de librerías que no vienen por defecto en el Linux Fedora Core 4. Por tanto, se tendrá que hacer una actualización del sistema operativo. En la Tabla 4.1 se puede ver una representación de las distintas dependencias junto con los sitios web de descarga de cada uno de los paquetes.

[7] Para poder acceder a la consola web desde una máquina remota hay que tener permiso. Eso se consigue añadiendo la siguiente línea al fichero de configuración *$RHINO_HOME/config/mlet.conf*, indicando el *host* que se quiere que tenga permiso para acceder de manera remota: *permission java.net.SocketPermission "remotehost", "accept,resolve";*.

Paquete	Necesita	Sitios web
linphone-1.1.0	- ilbc-rfc3951 - libosip2-2.2.0 - jack-audio-conncetion- kit-0.100.0 - libsamplerate-0.1.2 - speex-1.0.5	- www.linphone.org - www.linphone.org - http://jackit.sourceforge.net/ - www.mega-nerd.com/SRC/download.html - www.speex.org/download.html
jack-audio-connection-kit-0.100.0	- libsndfile-1.0.12	- www.mega-nerd.com/libsndfile/#Download
libsndfile-1.0.12	- sqlite-3.2.7	- www.sqlite.org/download.html
sqlite-3.2.7	- tcl8.4.9	- www.escomposlinux.org/lfs-es/blfs-es-6.0/general/tcl.html
libsamplerate-0.1.2	- fftw-3.0.1 - libsndfile-1.0.12	- www.fftw.org - www.mega-nerd.com/libsndfile/#Download

Tabla 4.1: Tabla de dependencias para la instalación del Linphone.

Así pues, primero se realizará la instalación de los distintos paquetes necesarios, para lo cual se recomienda abrir una shell en modo super-usuario, es decir, con todos los permisos, y proceder a la instalación de los diferentes paquetes siguiendo el procedimiento que se indique en los distintos ficheros de instalación o las indicaciones de la propia página web de donde se descargó el paquete. Normalmente, los pasos a seguir para realizar la instalación de los paquetes, los cuales se han ubicado en la ruta del proyecto */home/vhros/PFC/*, son los siguientes:

```
[root@rhinoslee “paquete”]$ ./configure
[root@rhinoslee “paquete”]$ make
[root@rhinoslee “paquete”]$ make install
```

Siguiendo este proceso de instalación, los ficheros de los paquetes serán instalados en las rutas determinadas por defecto. De ahí la necesidad de ser super-usuarios, ya que las rutas por defecto suelen pertenecer al *root*. En caso de querer hacer una instalación más personalizada, habrá que acudir a los ficheros de instalación de los distintos paquetes y seguir las indicaciones correspondientes.

Finalmente, ya se puede efectuar la instalación del cliente Linphone, aunque previamente habrá que tener en cuenta que el Linphone buscará los distintos paquetes que necesite (ficheros .pc) en las rutas marcadas por la variable de entorno *PKG_CONFIG_PATH*[8], que serán todas aquellas que terminen en *pkgconfig*. Para

[8] Esto mismo ocurrirá durante la instalación de los paquetes individuales, como es en el caso del *libsamplerate*, ya que depende del *fftw* y del *libsndfile*. Así que, simplemente tenerlo en consideración e ir actualizando la variable de entorno sobre la marcha.

determinar estas rutas se abrirá una shell en modo super-usuario (para poder buscar en toda la estructura de ficheros) y se ejecutará la siguiente instrucción:

```
[root@rhinoslee ~]$ find / -name 'pkgconfig'
```

El resultado obtenido se uilizará para añadir esa variable de entorno en el *.bash_profile* (fichero de configuración de la shell de usuario).

Se edita el fichero *.bash_profile* en modo usuario:

```
[vhros@rhinoslee ~]$  gedit .bash_profile
```

Se añade la nueva variable de entorno:

```
export PKG_CONFIG_PATH=/usr/lib/pkgconfig:/usr/lib/qt-3.3/lib/pkgconfig:/usr/share/
pkgconfig:/usr/local/lib/pkgconfig
```

Una vez aplicados los cambios efectuados, bien reiniciando el sistema, o bien exportando la variable en la misma shell desde donde se están instalando los paquetes, ya se puede instalar el Linphone:

```
[root@rhinoslee linphone-1.1.0]$ ./configure --with-ilbc=/usr/local
[root@rhinoslee linphone-1.1.0]$ make
[root@rhinoslee linphone-1.1.0]$ make install
```

Siguiendo estos pasos, la instalación debe haberse efectuado correctamente. Sin embargo, siempre es recomendable redireccionar las salidas de errores a algún fichero, tal que se pueda ver en cada momento si se ha producido algún error y poder solucionarlo. Éste ha sido el procedimiento seguido, si bien, se trata de la instalación de una aplicación. Por tanto, puede hacerse al gusto de cada uno. El hecho de haber detallado estos pasos ha sido porque la instalación no es tan simple como ejecutar un *script*, así que se creyó conveniente explicar los posibles inconvenientes que pueden surgir.

4.4.5. Configuración de los clientes Linphone

Para llevar a cabo este cometido habrá que tener una idea clara de cómo va a ser la red que se va a montar (Figura 4.10).

Figura 4.10: Red VoIP montada.

Para el montaje de la red VoIP se hizo uso de 3 puestos de trabajo (1 Servidor y 2 Clientes: *sip:jain1@rhinoslee.proxy.sip* y *sip:jain2@rhinoslee.proxy.sip*).

Así pues, una vez instalado el Linphone en los dos puestos de trabajo destinados a ser clientes SIP, se procede a su configuración. Tan sólo se hará una vez, ya que la única diferencia entre una configuración y otra es el nombre del cliente, jain1 en un caso y jain2 en el otro. Esta configuración será necesaria para que los clientes conozcan el servidor en el que tienen que registrarse, tal que una vez registrados el *proxy* pueda enrutar las llamadas realizadas en la red.

Primero se arranca el cliente Linphone, tal y como se ve en la Figura 4.11.

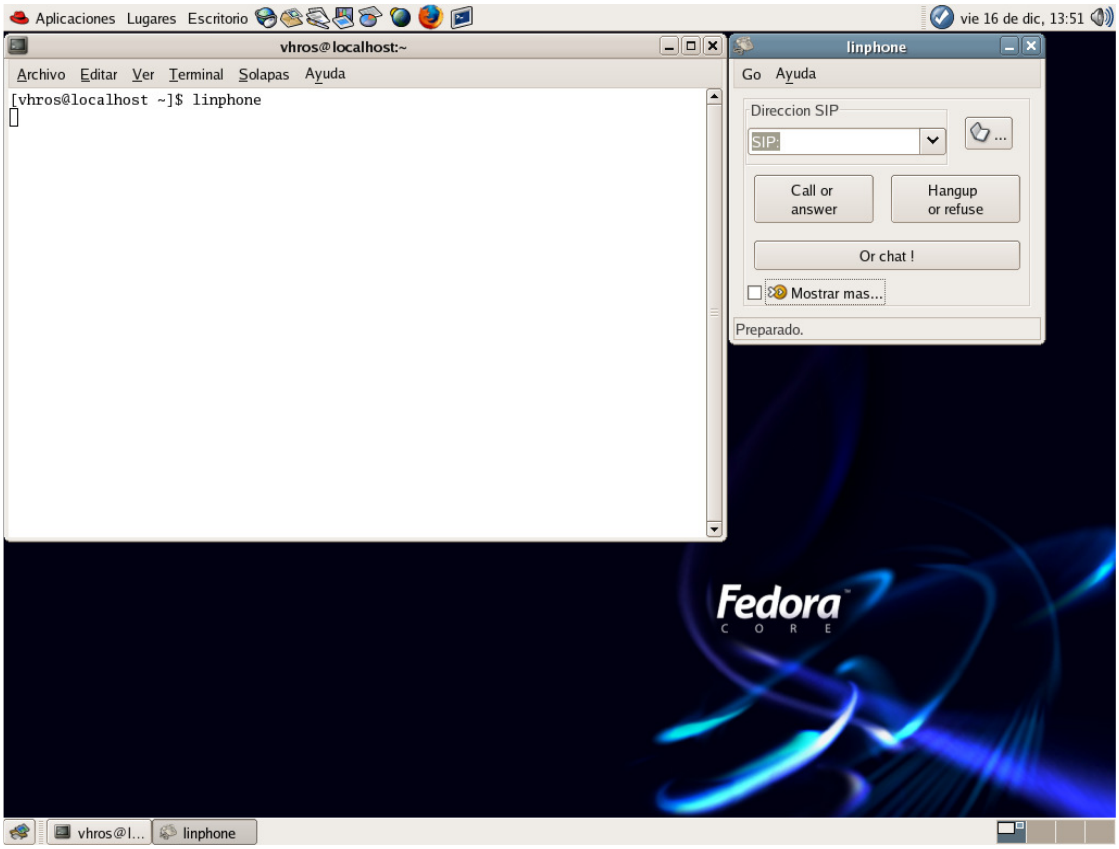


Figura 4.11: Arrancando Linphone.

Una vez arrancado el cliente, se elige la opción *Preferencias* dentro de *Go*. A continuación se selecciona la pestaña *SIP*. Estando ya en esta ventana, se pulsa el botón *Add proxy/registrar* y se introducen los datos (Proxy: *rhinoslee.proxy.sip* | Identidad: *jain1@rhinoslee.proxy.sip*)[9]. Además, se activará la casilla de *Envío de registro* y la de *Publicación de información de presencia*. Esto mismo se puede ver con más detalle en la Figura 4.12.

[9] Matizar que estas direcciones, por tratarse de direcciones SIP, tienen un campo *"SIP:"* al principio. Sin embargo, este campo será motivo de error. Pero eso ya se verá en el punto 4.4.6, donde se comprueba el funcionamiento de la aplicación.

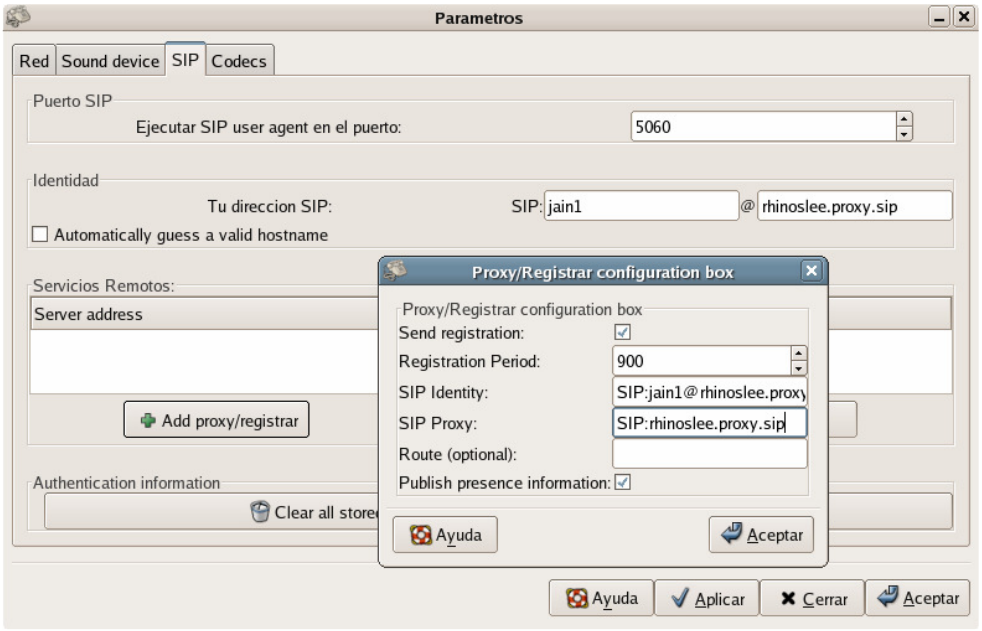


Figura 4.12: Configuración del *Proxy/Registrar*.

Finalmente, se pulsa *Aceptar* y tendría que quedar algo como lo que se observa en la Figura 4.13.

Figura 4.13: Configuración del Linphone.

Una vez aceptada la configuración, se cerrará el cliente Linphone para completar el último paso antes de iniciar el SLEE. Este último paso consiste en actualizar el fichero */etc/hosts* del puesto de trabajo, es decir, hay que indicarle la IP correspondiente al *host rhinoslee.proxy.sip*. Por tanto, se editará el fichero */etc/hosts* en modo super-usuario y se añadirá la siguiente línea:

```
172.16.17.179        rhinoslee.proxy.sip
```

A continuación, desde el otro puesto de trabajo (el servidor), se arranca el SLEE y se despliegan los servicios:

Shell 1

```
[vhros@rhinoslee ~]$ pg_ctl –D $PGDATA –l logfile start
postmaster iniciándose
```

Shell 2

```
[vhros@rhinoslee ~]$ cd $RHINO_HOME
[vhros@rhinoslee rhino]$ ./init-management-db.sh
[vhros@rhinoslee rhino]$ ./start-rhino.sh
```

Shell 3

```
[vhros@rhinoslee ~]$ cd $RHINO_HOME
[vhros@rhinoslee rhino]$ cd examples/sip
[vhros@rhinoslee sip]$ ant deployexamples
```

Antes de poderse registrar hay que tener en cuenta si hay activado algún cortafuegos. De ser así, en el equipo que hace de *proxy* habrá que configurar dicho cortafuegos de tal manera que acepte las peticiones de los clientes.

Finalmente, se arranca el cliente SIP en los otros 2 equipos y se comprueba que el registro es un éxito (Figura 4.14).

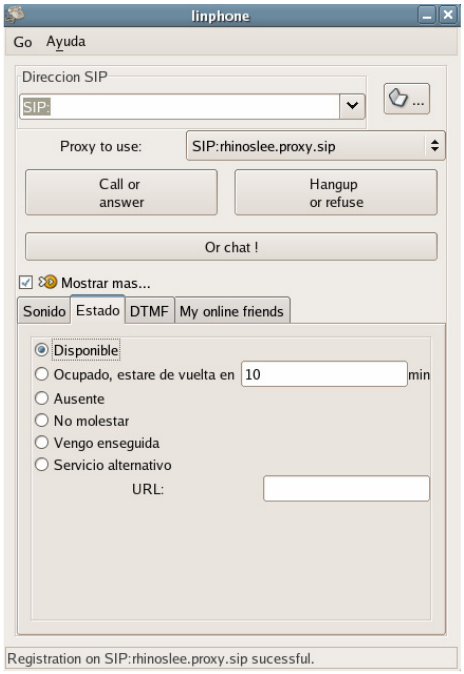


Figura 4.14: Cliente Linphone registrado.

4.4.6. Comprobación de la aplicación

Ya está todo instalado, el servidor en marcha y los 2 clientes arrancados y registrados.

Como se va hacer uso de bases de datos, se recomienda la instalación del *pgAdmin III* [31]. El pgAdmin III es una interfaz sencilla para el diseño y administración de una base de datos PostgreSQL, diseñada para ejecutarse en la mayoría de los Sistemas Operativos. La aplicación corre bajo GNU/Linux, FreeBSD y Windows 2000/XP. El pgAdmin III esta diseñado para responder a las necesidades de todos los usuarios, desde escribir simples consultas SQL a desarrollar bases de datos complejas. La interfaz gráfica soporta todas las características de PostgreSQL y facilita la administración. La aplicación también incluye un creador de consultas, un editor SQL, un editor de código del lado del servidor, etc. Además, se distribuye con un instalador y no requiere ningún *driver* adicional para comunicarse con la base de datos.

Una vez instalado y configurado el pgAdmin III para acceder a las bases de datos, se puede acceder a la tabla de registros, la cual tendrá un aspecto similar al que se muestra en la Figura 4.15.

	oid	sipaddress varchar	contactaddress varchar	expiry int8	qvalue int4	cseq int4	callid varchar
1	25972	SIP:jain1@rhinoslee.proxy.sip	SIP:jain1@172.16.17.225:5060	1135190663869	0	1	314792129@172.16.17.225
2	25971	SIP:jain2@rhinoslee.proxy.sip	SIP:jain2@172.16.17.227:5060	1135190660954	0	1	903802436@172.16.17.227
*							

Figura 4.15: Registro de los clientes SIP.

En los clientes no hace falta tocar el cortafuegos, ya que una vez abiertos los clientes SIP, estos estarán a la escucha de los mensajes SIP que entren en el equipo siempre y cuando no haya una restricción explícita.

Finalmente, ya se puede intentar hacer una llamada (ej.: jain1 llama a jain2).

Una vez comprobado que el diálogo se establece correctamente y con una buena calidad de sonido, se pasa a probar la opción de chat. Cual es la sorpresa cuando se comprueba que solamente se pueden enviar mensajes en un sentido, es decir, desde el cliente que realizó la llamada hacia el que la recibió.

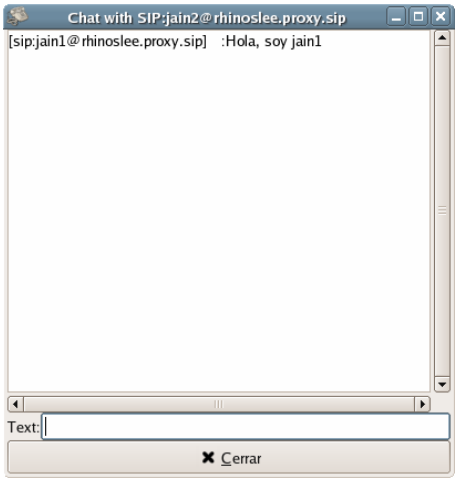


Figura 4.16: Ventana de chat del cliente jain1.

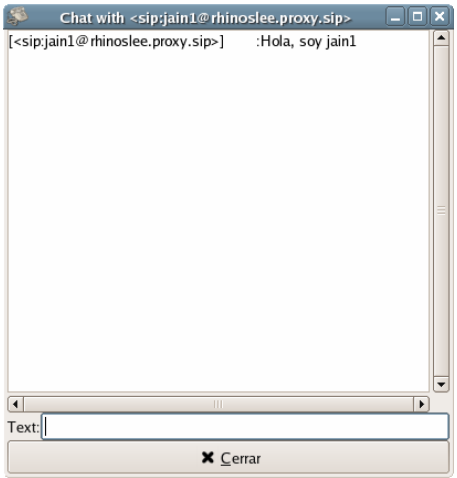


Figura 4.17: Ventana de chat del cliente jain2.

Para solucionar este problema, lo primero será hacer uso de un analizador de paquetes. Se hizo uso del *ethereal* [32].

La captura hecha con *ethereal* a la interfaz *rhinoslee.proxy.sip* durante el envío de mensajes de texto entre los dos clientes SIP se puede observar en la Figura 4.18.

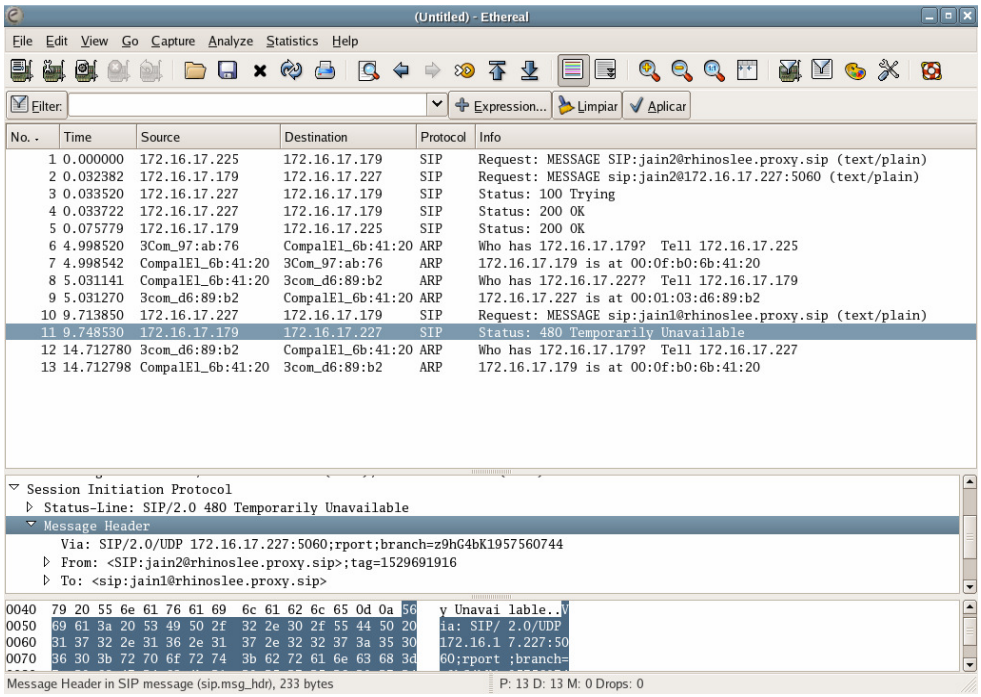


Figura 4.18: Etheral durante el envío de mensajes instantáneos.

Con esto, y prestando atención a las trazas que aparecen en el SLEE (Figura 4.19 y Figura 4.20), se observa que el problema no es que los clientes SIP estén enviando mal los paquetes. Lo que ocurre es que cuando las direcciones pasan por el *proxy*, estas pasan a tener un formato de todo en minúsculas. Recordar que las direcciones SIP tienen un campo al principio *"SIP:"*. Por tanto, las direcciones registradas y que se alojan en la tabla de registro de la base de datos son *SIP:jain1@rhinoslee.proxy.sip* y *SIP:jain2@rhinoslee.proxy.sip* (Figura 4.15). Pues bien, cuando el cliente jain1 llama a jain2 marca la dirección correcta. Esa dirección llega al *proxy*, éste usa el servicio de localización y encuentra la dirección destino de jain2 y le llama. Pero a jain2 la llamada le llega del *proxy*, y como se ha dicho anteriormente, éste convierte todo a minúsculas, así que jain2 piensa que quien le está llamando es *sip:jain1@rhinoslee.proxy.sip*, de modo que cuando le intenta mandar un mensaje de texto, lo que hace es enviárselo a *sip:jain1@rhinoslee.proxy.sip* en lugar de a *SIP:jain1@rhinoslee.proxy.sip*. Esta petición vuelve a pasar por el *proxy*, que utilizará el servicio de localización y evidentemente no encontrará ninguna dirección en la tabla de registro que se corresponda con *sip:jain1@rhinoslee.proxy.sip*, así que considerará que ese usuario no está registrado. Por eso los mensajes solamente llegan en una dirección.

La forma de solucionarlo es muy sencilla, la dificultad estriba en darse cuenta de la causa del problema.

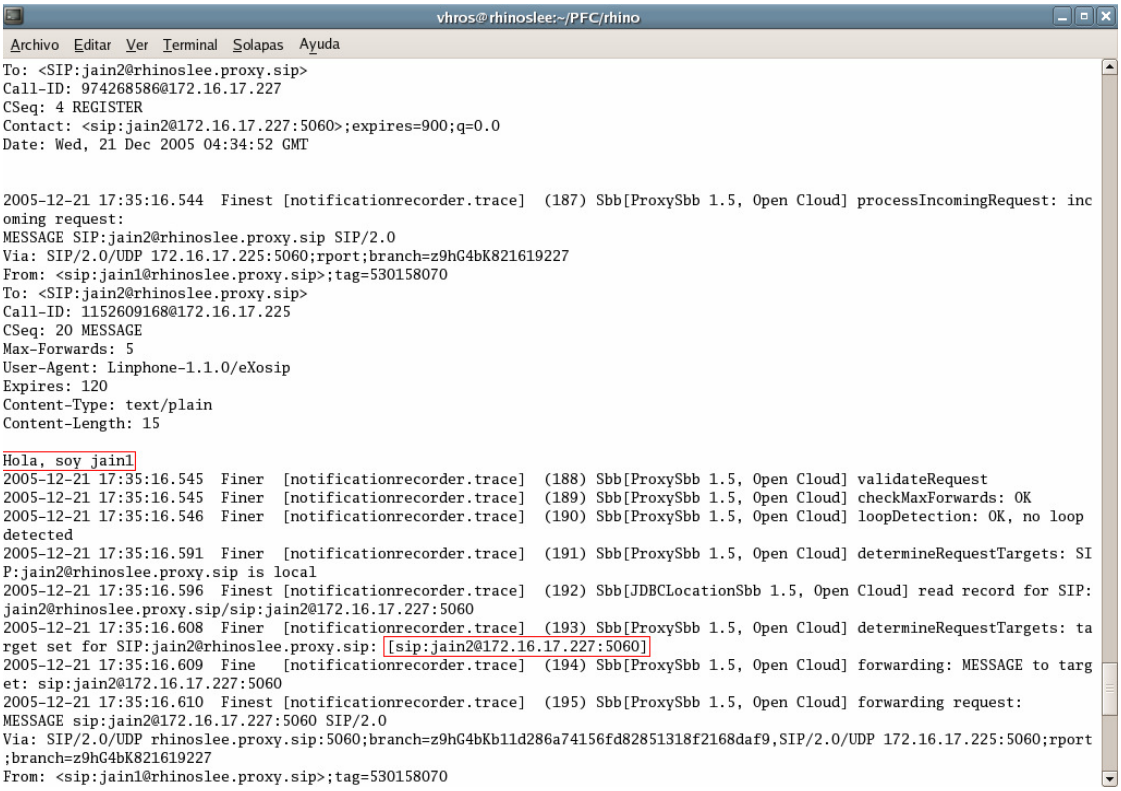


Figura 4.19: Trazas del mensaje enviado por el cliente jain1.

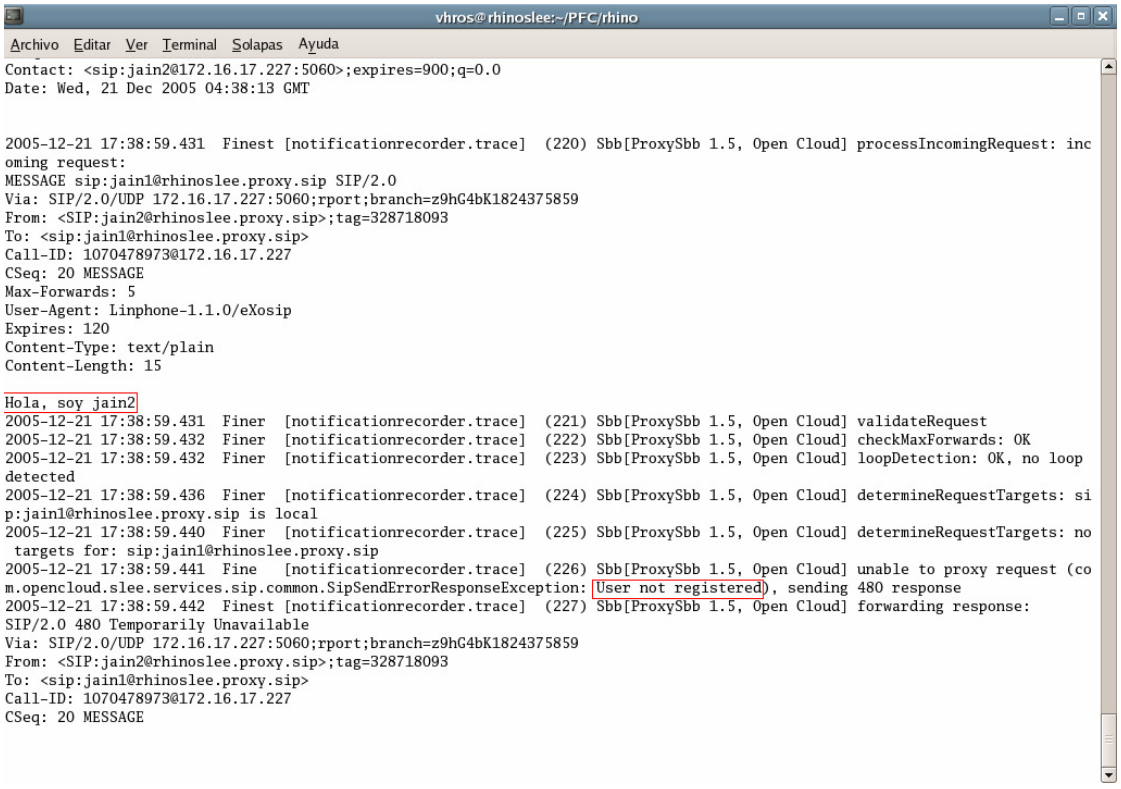


Figura 4.20: Trazas del mensaje enviado por el cliente jain2.

La solución está en registrar a los usuarios usando direcciones SIP con todos los caracteres en minúsculas. Para ello se cambiará la configuración de los clientes Linphone (Figura 4.21).

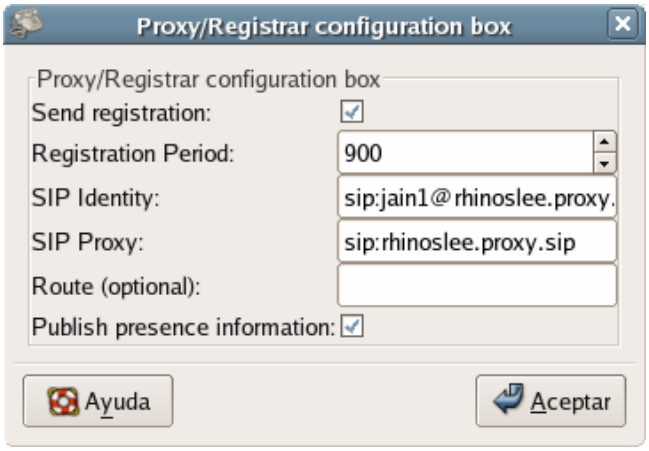


Figura 4.21: Configuración correcta.

	oid	sipaddress varchar	contactaddress varchar	expiry int8	qvalue int4	cseq int4	callid varchar
1	25893	sip:jain1@rhinoslee.proxy.sip	sip:jain1@172.16.17.225:5060	1135187599481	0	3	1004608834@172.16.17.225
2	25892	sip:jain2@rhinoslee.proxy.sip	sip:jain2@172.16.17.227:5060	1135187579727	0	3	199622971@172.16.17.227
*							

Figura 4.22: Nuevo registro de los clientes.

Con esta modificación, ya se puede hacer uso de la aplicación sin ningún problema y mantener conversaciones de voz con el envío de mensajes instantáneos tanto simultáneamente como no.

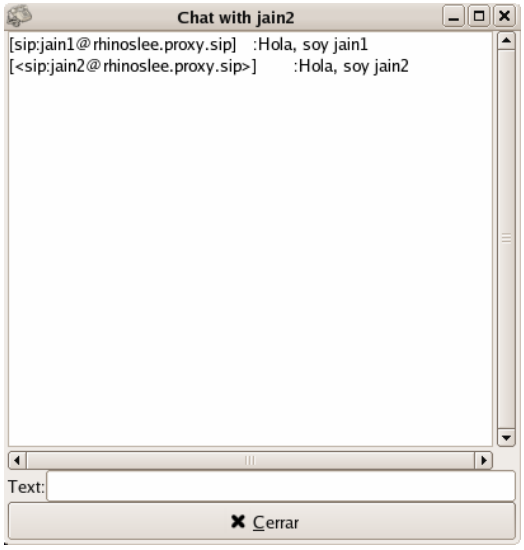


Figura 4.23: Ventana de chat del cliente jain1.

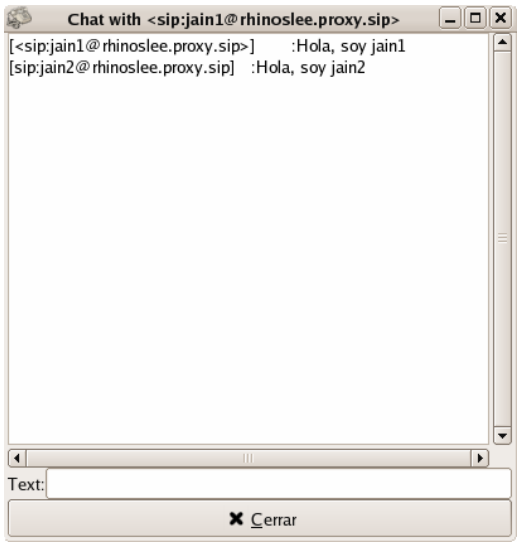


Figura 4.24: Ventana de chat del cliente jain2.

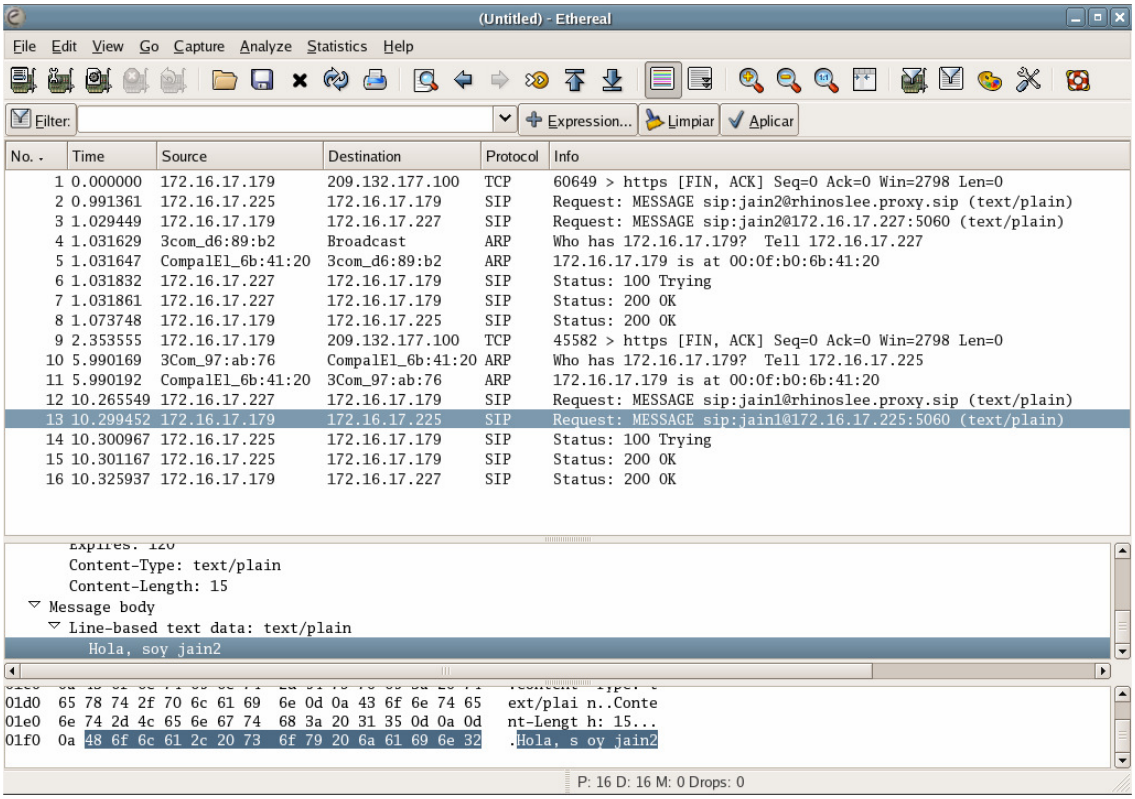


Figura 4.25: Etheral durante el envío de mensajes instantáneos (solucionado el problema).

Esta situación da lugar a plantearse la pregunta de por qué al establecer una conversación de voz no hay ningún inconveniente, mientras que en el uso de la opción de chat influye el formato de las direcciones SIP. Analizando la situación mediante la captura de paquetes con el *ethereal*, se obtiene la respuesta a dicha pregunta.

Se consideran dos aplicaciones distintas dentro del cliente Linphone y dos usuarios, usuario A y usuario B, ya registrados correctamente en el servidor. Por un lado está la aplicación para establecer conversaciones de voz, y por otro la aplicación válida para chatear.

Al establecer una conversación de voz se crea un diálogo entre los dos clientes, esto quiere decir que se establece una sesión SIP entre los extremos finales. Por tanto, cuando el usuario A llama a B, el *proxy* interviene solamente hasta que los dos usuarios inician la conversación. Primero se efectúa un intercambio de mensajes SIP, durante el que A invita a B a una conversación (SIP/SDP. Request: INVITE), posteriormente se envían los tonos de llamada (SIP. Status: 100 Trying y 180 Ringing), y finalmente la aceptación de la llamada por parte de B (SIP. Status: 200 OK). En todo esto habrá tomado parte el *proxy*, que con ayuda del servicio de localización sabrá donde tendrá que enviar el INVITE, mensaje que le llegará a B y en el que ya va la información de la dirección del usuario llamante (ej.: *jain1@172.16.17.225:5060*). Los mensajes posteriores le llegarán a A y en ellos va la información correspondiente al usuario llamado (ej.: *jain2@172.16.17.227:5060*). De esta manera, los clientes SIP tendrán la información necesaria para establecer la conversación sin necesidad de acudir al *proxy*.

Por tanto, una vez aceptada por B la petición de A comienza el flujo RTP, en el cual ya no es necesaria la intervención del *proxy*. Por tanto, no tendrá efecto el hecho de poner el campo *"SIP:"* en mayúsculas, ya que la consulta a la base de datos se hace únicamente para el envío del INVITE (y no para el resto del flujo de mensajes), y para que eso se haga correctamente lo único que hay que hacer es llamar usando la dirección correcta del destino (bien sea en mayúsculas o bien en minúsculas). Para salir de dudas se puede parar el SLEE y comprobar que la conversación no se corta.

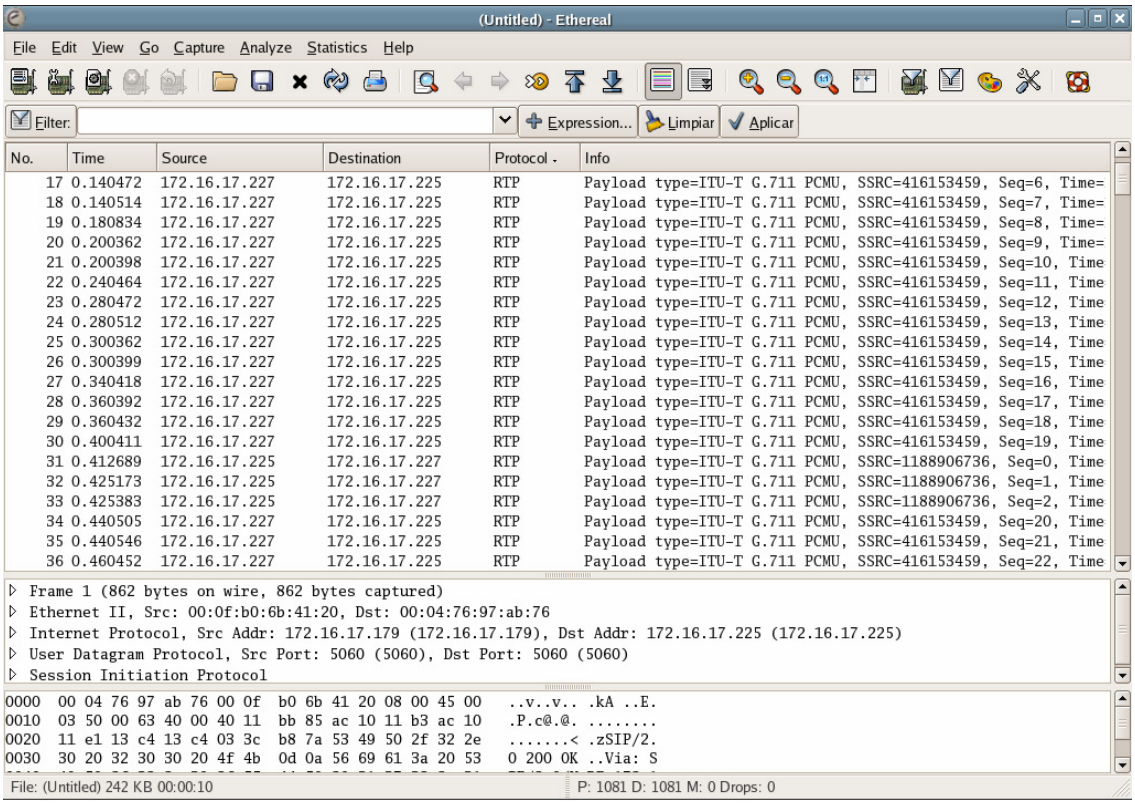


Figura 4.26: Conversación entre jain1 y jain2.

Por otra parte está el chat. La gran diferencia con la utilidad anterior es que en el Linphone, para el envío de mensajes instantáneos, no se establece una sesión SIP entre los dos extremos finales. Aquí lo que se produce es un intercambio de mensajes SIP en los que va información, en texto plano, correspondiente al texto escrito en la ventana del chat. En este caso todos los mensajes pasarán por el *proxy*, y es éste el que se encargará de enrutarlos. De ahí la importancia de que en todo momento la dirección de destino de los mensajes se corresponda con la dirección registrada en la base de datos, ya que para cada mensaje enviado se hará uso del servicio de localización.

Para terminar, se explicará qué habría que hacer si tan sólo se dispone de 2 equipos en lugar de 3 (Figura 4.27). En este caso el equipo que hace de *proxy* y que por tanto tiene el SLEE, tendrá que hacer también de cliente SIP. Para ello, hay que tener presente que no debe existir conflicto entre el puerto UDP del Adaptador de Recurso SIP (5060) y el puerto SIP del cliente Linphone en el mismo equipo.

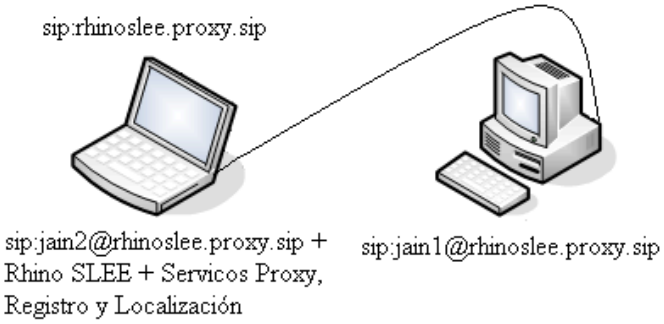


Figura 4.27: Red VoIP alternativa.

Según lo dicho anteriormente, cuando se esté configurando el cliente alojado en el mismo equipo que el SLEE, lo que se hará es cambiar el puerto SIP del Linphone, tal y como se ve en la Figura 4.28. Se cambia el 5060, que viene por defecto, por el 5061 o cualquier otro que se pueda utilizar.

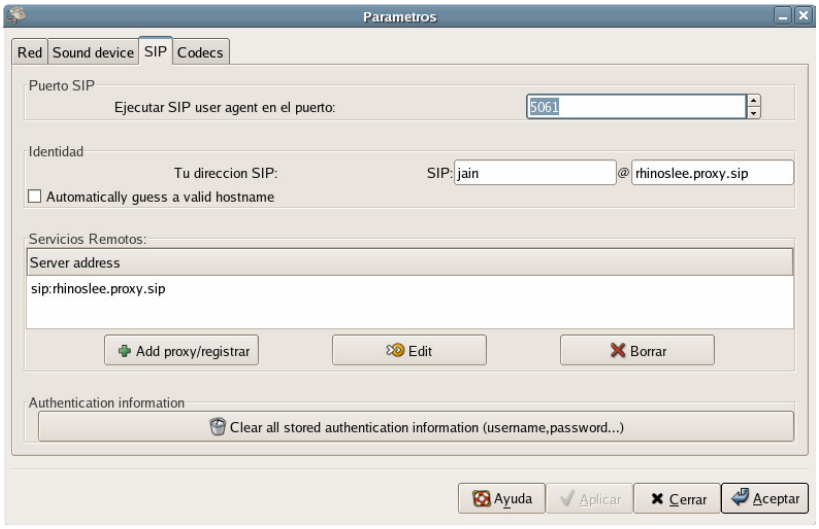


Figura 4.28: Cambio del puerto SIP en el cliente.

Otra opción hubiese sido cambiar en el fichero de configuración *build.properties* (Figura 4.1) la línea que hace referencia a las propiedades del SIP RA, es decir, *sip.ra.properties*[10]. Para evitar el conflicto comentado anteriormente, se cambiaría el puerto UDP del SIP RA (5060) por otro distinto, como puede ser el 5070. De esta forma también se evitaría el conflicto y se podría dejar el puerto SIP del cliente Linphone en 5060. Sin embargo, para llegar ahora al *proxy* se haría a través del puerto 5070, ya que el servicio *proxy* está unido al SIP RA, por lo que habría que indicarlo de alguna manera en la configuración del cliente Linphone. La forma de hacerlo es editar otra vez la dirección del servidor en los clientes SIP y escribir *"sip:rhinoslee.proxy.sip:5070"* en el campo *"SIP Proxy:"* (Figura 4.29).

[10] Destacar que la dirección 0.0.0.0 se corresponde con la reservada para las direcciones desconocidas, por lo que se puede realizar el registro desde la misma máquina donde está el SLEE, usando la dirección de bucle invertido (127.0.0.1) o la propia correspondiente al puesto de trabajo (172.16.17.179).

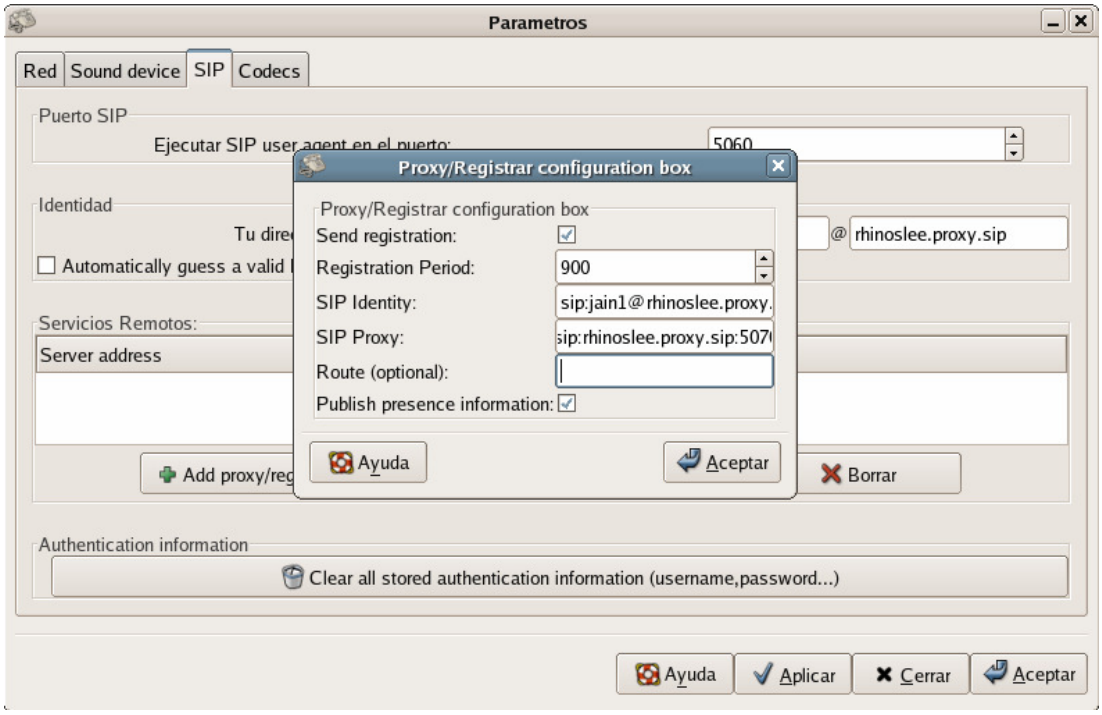


Figura 4.29: Cambio de la configuración del *proxy*.

Aceptando esta configuración, el resultado final sería el de la Figura 4.30.

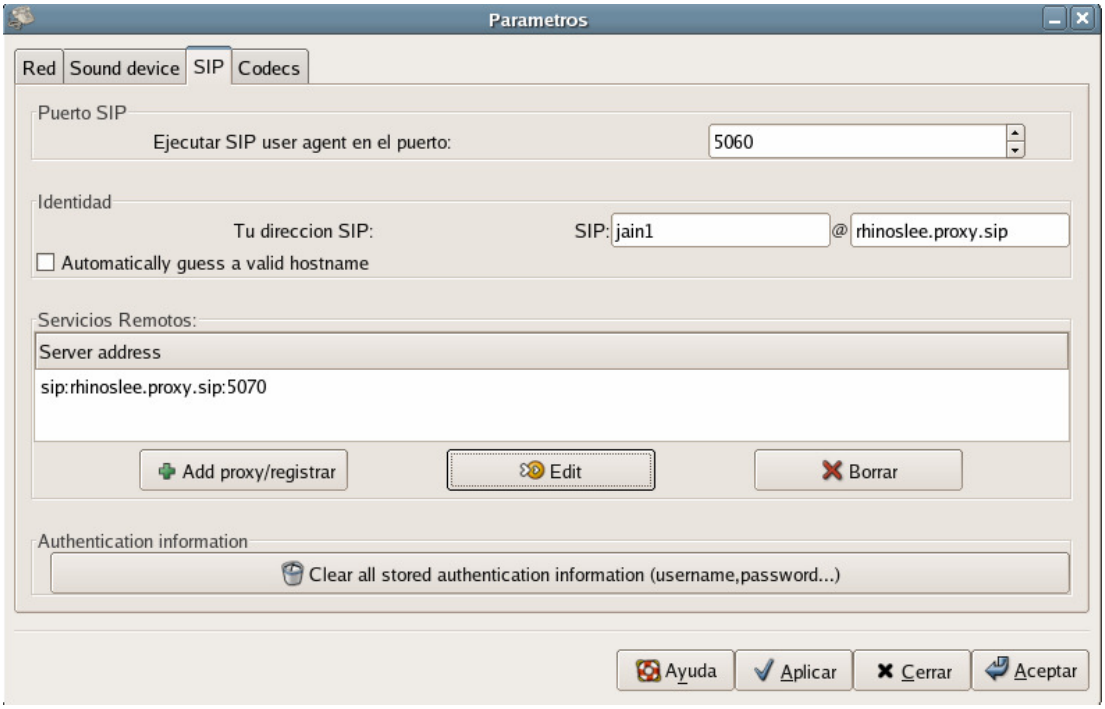


Figura 4.30: Nueva configuración del Linphone.

Se puede plantear la pregunta de por qué antes no hacia falta indicar en el campo *"SIP Proxy:"* de los clientes que el puerto del *proxy* era el 5060. La respuesta está en

que el formato usado para indicar la dirección del *proxy* es *"sip:host:port"* y el Linphone considera por defecto, si no hay nada a continuación del *host*, que el puerto es el 5060.

4.5. Servicio de telefonía sobre la red VoIP

Una vez conseguido el primer objetivo, se planteó el desplegar algún servicio de telefonía sobre la red VoIP montada. Uno de los servicios más típicos es el del desvío de llamada. Se utilizará un servicio similar, llamado FMFM (Find Me Follow Me), que consistirá en permitir a los clientes el poder especificar direcciones SIP alternativas para los casos en los que dichos clientes no estén disponibles.

Para desplegar este servicio, primero se arrancará el SLEE. Una vez hecho esto, se utilizará la herramienta Ant para desplegar el servicio *fmfm*:

```
[vhros@rhinoslee sip]$ ant deployfmfm
```

Este servicio hace uso de lo que se llama *Profile* (Perfil). Por tanto, será aquí donde se haga uso de la consola web (*https://rhinoslee.proxy.sip:8443/*) para gestionar los perfiles de usuario, y por tanto de lo que es la arquitectura JMX, la cual se implementa mediante MBeans.

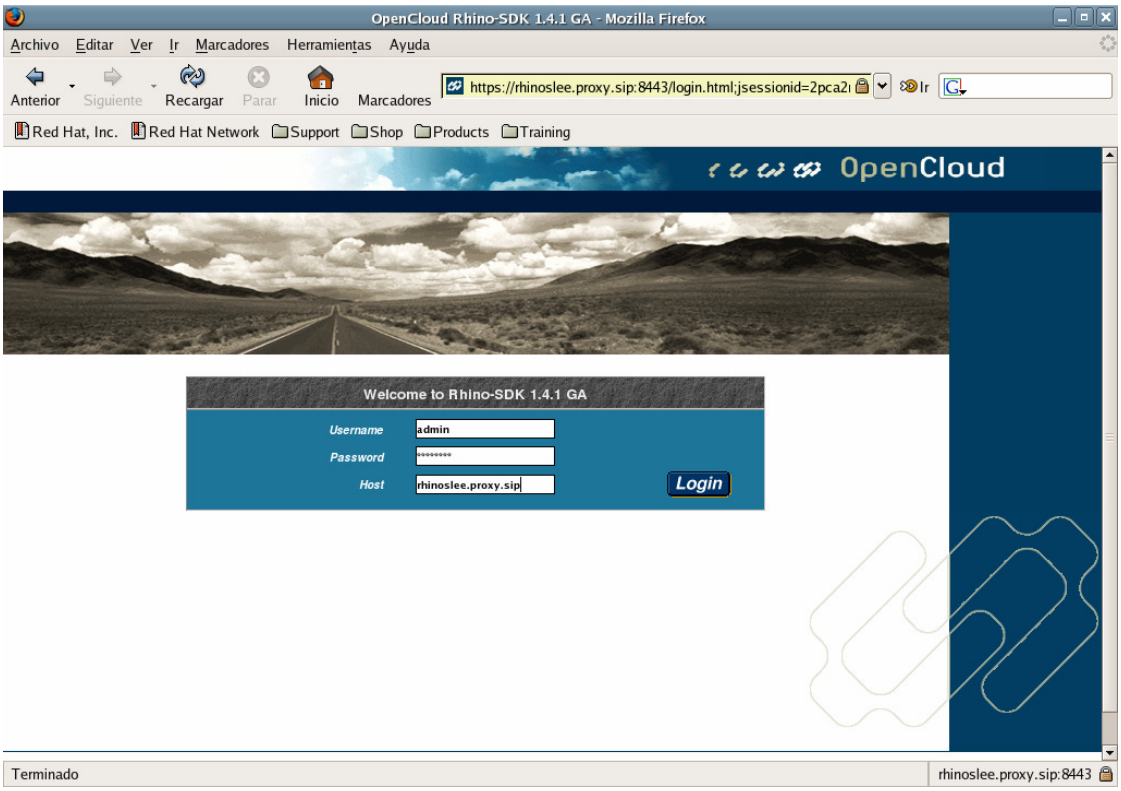


Figura 4.31: Acceso a la consola web.

Por defecto, el nombre de usuario es "*admin*", la clave es "*password*" y el *host* es, en este caso, "*rhinoslee.proxy.sip*". Estos datos están y pueden ser modificados en el fichero de configuración *$RHINO_HOME/config/config_variables* (Figura 4.32).

```
WEB_CONSOLE_HTTP_PORT=
WEB_CONSOLE_HTTPS_PORT=8443
WEB_CONSOLE_KEY_PASS=changeit
WEB_CONSOLE_STORE_PASS=changeit
WEB_CONSOLE_HOSTNAME=rhinoslee.proxy.sip
RHINO_PASSWORD=password
RHINO_USERNAME=admin
LOCALIPS="[0:0:0:0:0:0:0:1%1] 127.0.0.1 [fe80:0:0:0:20f:b0ff:fe6b:4120%2] 172.16.17.179"
RHINO_WATCHDOG_DUMPTHREADS=/home/vhros/PFC/rhino/dumpthreads.sh
```

Figura 4.32: Login y password de acceso a la consola web.

Una vez que se haya accedido a la consola web, aparecerá la siguiente pantalla (Figura 4.33):

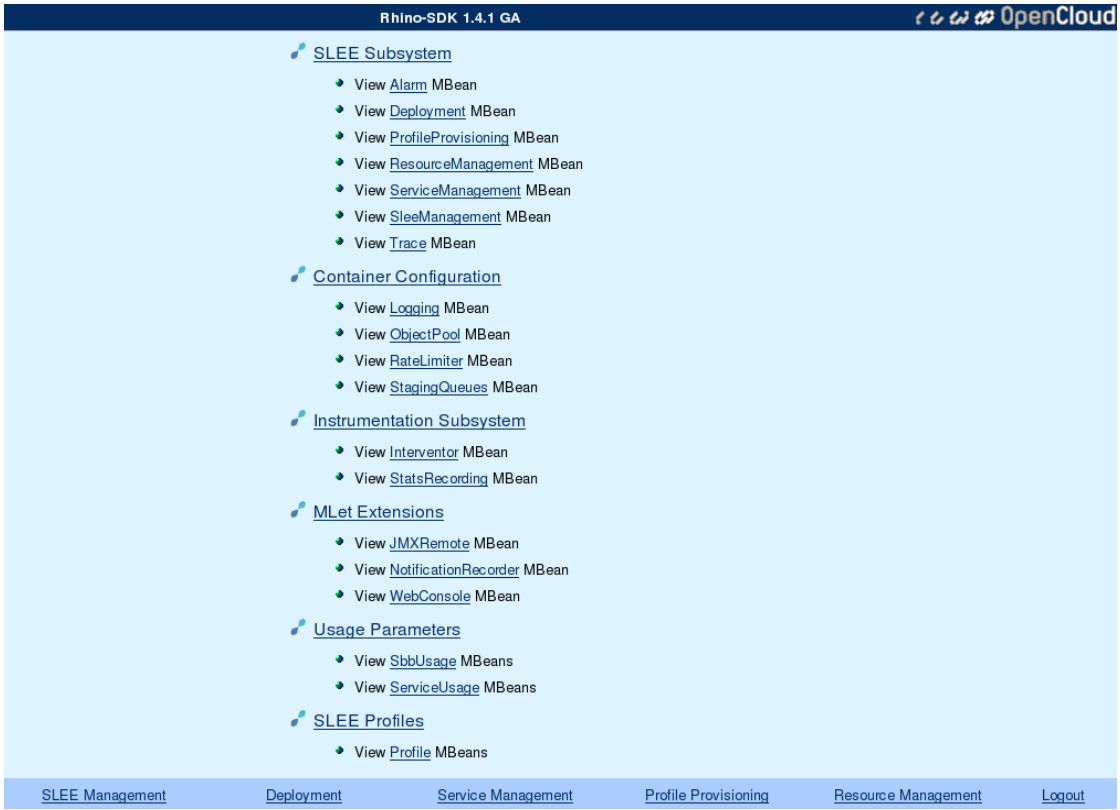


Figura 4.33: Interfaz principal.

Desde la interfaz principal se podrá acceder al resto de interfaces visuales que proporciona la consola web. Lo que interesa en estos momentos son los perfiles, que son gestionados con el Profile Provisioning MBean (MBean de Aprovisionamiento de Perfiles), así que se accederá a ese MBean (Figura 4.34).

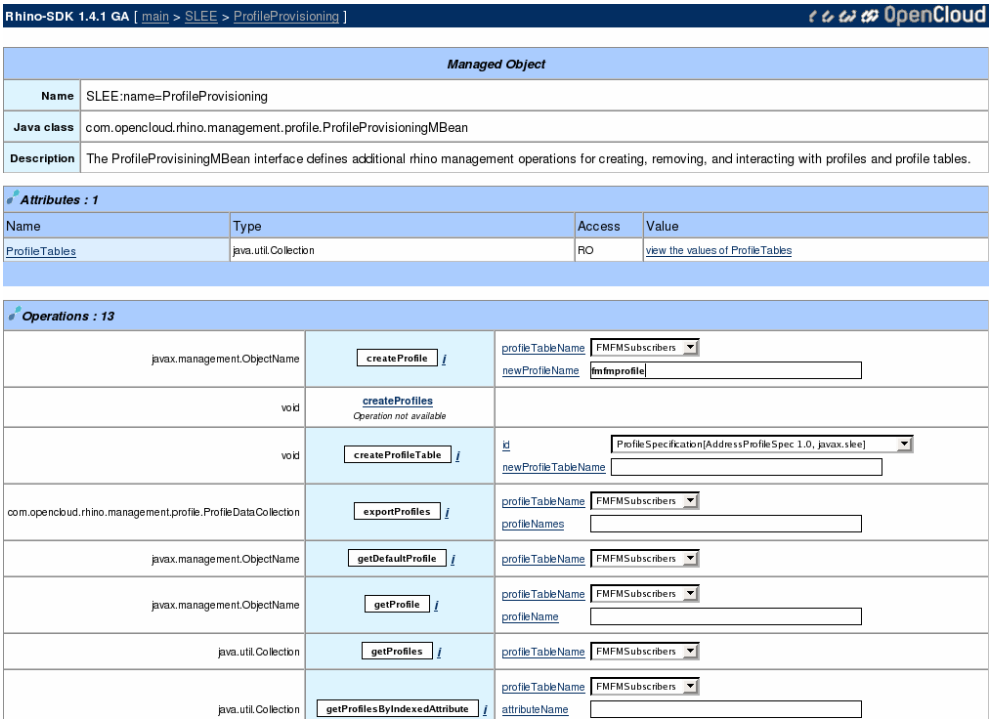


Figura 4.34: Profile Provisioning MBean.

Al haber desplegado previamente el servicio *fmfm*, resulta que ahora se puede observar en la Figura 4.34 como aparece creada una Tabla de Perfiles (Profile Table) llamada *FMFMSubscribers* y que servirá para albergar a los usuarios que dispongan de este servicio, mediante la creación de distintos perfiles. Así pues, se creará un perfil (*fmfmprofile*) de ejemplo que servirá para hacer uso de la aplicación.

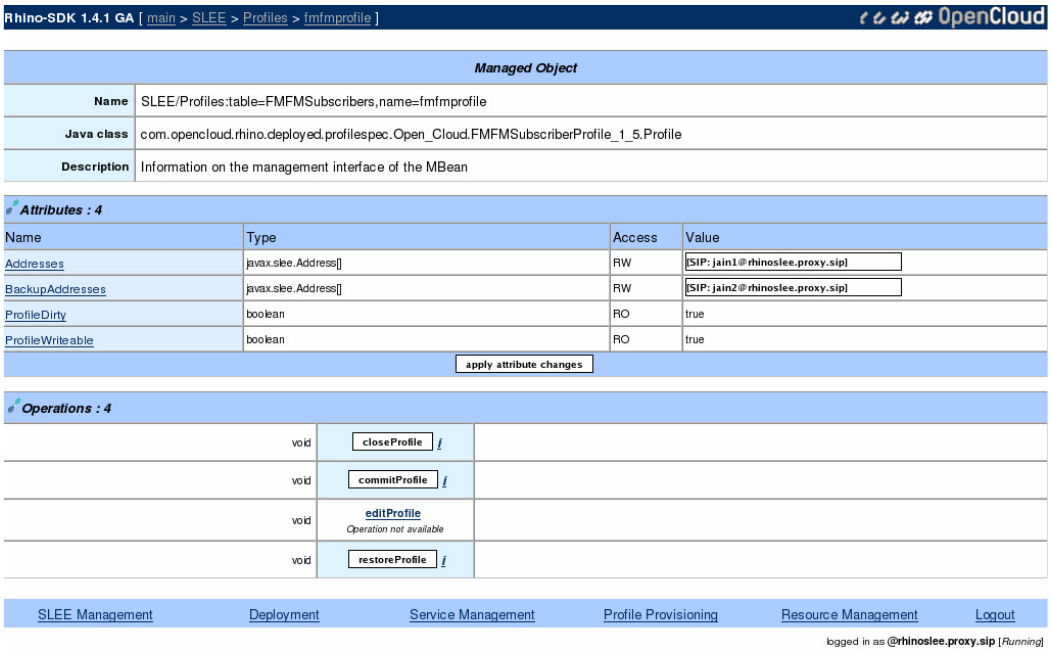


Figura 4.35: Creación de un perfil.

Lo que se hará en un principio, es introducir en el atributo *Addresses* la dirección SIP del usuario que va a hacer uso de este servicio, y en el atributo *BackupAddresses* la dirección SIP a la que el usuario quiere que vaya dirigida la llamada en caso de que él no esté disponible. Una vez hecho esto, se aplican los cambios (Figura 4.35). Pues bien, si se procede a asignar esos atributos (*commitProfile*) ocurre lo siguiente:

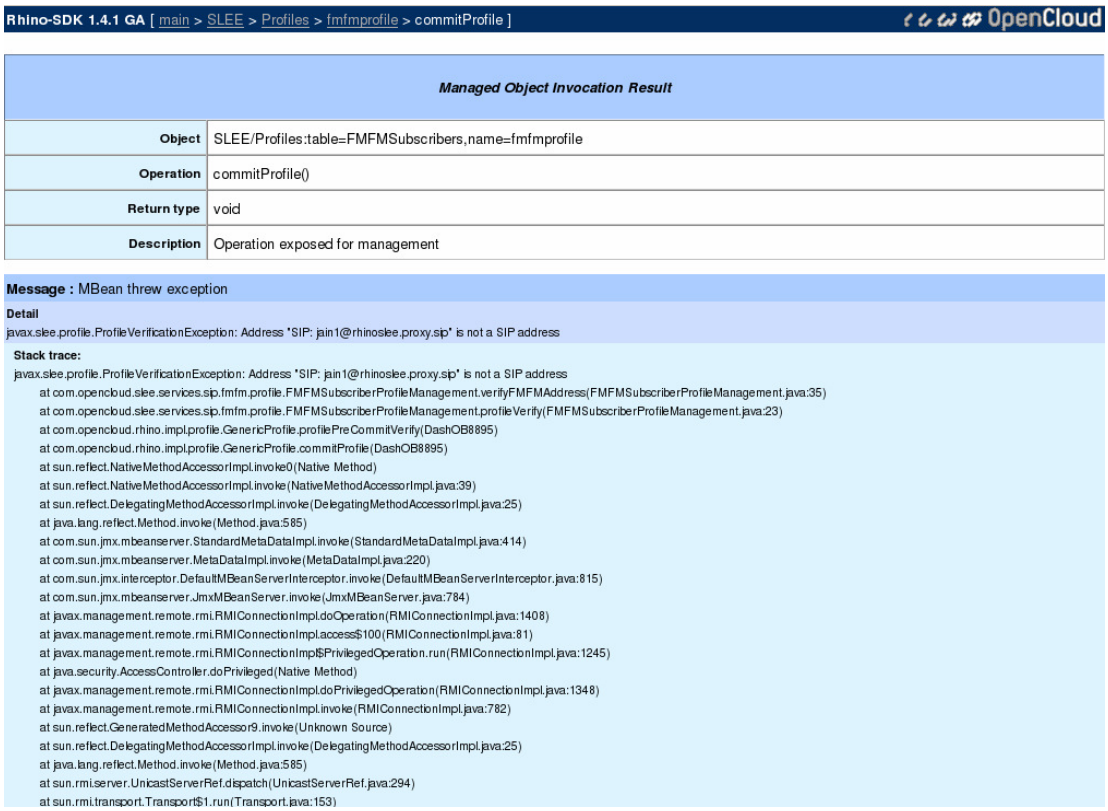


Figura 4.36: Error en el *commitProfile*.

Se ha producido un error que indica que la dirección que se ha puesto en el atributo *Addresses* (*SIP:jain1@rhinoslee.proxy.sip*) no es una dirección SIP. Esto es algo que puede parecer en un principio bastante ilógico, así que se acudirá a la clase de Java *FMFMSubscriberProfileManagement*, que es donde parece ser que se produce este error. Por tanto, habrá que dirigirse a la línea 35 del código fuente, tal y como indica la salida de errores de la Figura 4.36:

```
[vhros@rhinoslee ~]$ cd $RHINO_HOME/examples/sip/fmfm/profile
[vhros@rhinoslee profile]$ gedit FMFMSubscriberProfileManagement.java
```

Efectivamente, es en la línea 35 donde se produce la excepción *"is not a SIP address"*. Se observa que esa excepción se lanza en el proceso de verificación de las direcciones (Figura 4.37), y que el requisito es que la dirección SIP empiece por *"sip:"* o *"sips:"*. Se puede pensar en un principio que la dirección SIP utilizada cumple con ese requisito. Sin embargo, no es así, y es aquí donde reside la importancia de tener un conocimiento previo del JAIN SLEE.

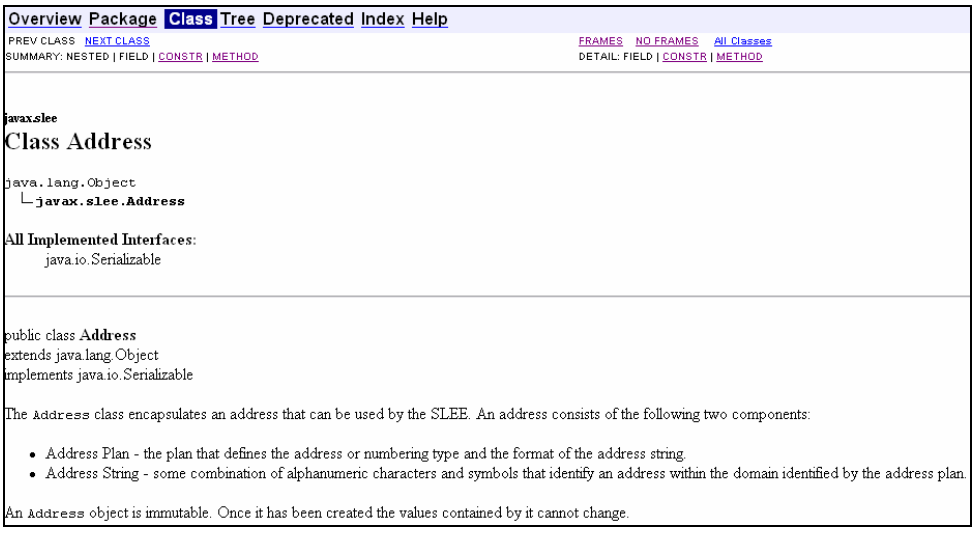
```
    /**
     * Ensure all addresses are SIP addresses
     */
    public void profileVerify() throws ProfileVerificationException {
        Address[] addresses = getAddresses();
        for (int i = 0; i < addresses.length; i++) verifyFMFMAddress(addresses[i]);
        Address[] backupAddresses = getBackupAddresses();
        for (int i = 0; i < backupAddresses.length; i++) verifyFMFMAddress(backupAddresses[i]);
    }

    private void verifyFMFMAddress(Address address) throws ProfileVerificationException {
        // Check address plan
        if (address.getAddressPlan() != AddressPlan.SIP)
            throw new ProfileVerificationException("Address \"" + address + "\" is not a SIP
address");
        // Check URI scheme - must be sip: or sips:
        String uri = address.getAddressString().toLowerCase();
        if (!(uri.startsWith("sip:") || uri.startsWith("sips:")))
            throw new ProfileVerificationException("Address \"" + address + "\" is not a SIP
address");
    }
}
```

Figura 4.37: Clase de Java *FMFMSubscriberProfileManagement.java*

Visitando el sitio web *http://jcp.org/en/jsr*, se puede descargar tanto la JSR 22 [17], como la JSR 240 [18]. En ambas se tiene acceso a la API y a la documentación de todos los paquetes y clases de JAIN SLEE. Al observar la Figura 4.35, se aprecia como los atributos *Addresses* y *BackupAddresses* son del tipo *javax.slee.Address*, y acudiendo a la documentación (Figura 4.38) se concluye que la clase *Address* encapsula una dirección que puede ser usada por un SLEE. Esta dirección consiste en 2 componentes:

- **Address Plan:** Se corresponde con el plan que define la dirección o el tipo de numeración, y el formato de la cadena de dirección (address string). En este caso será *"SIP:"*
- **Address String:** Combinación de caracteres alfanuméricos y símbolos que identifican una dirección dentro de un dominio reconocido por el plan de dirección. En este caso será *"sip:jain#@rhinoslee.proxy.sip"*.


Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS
SUMMARY: NESTED | FIELD | CONSTR | METHOD

FRAMES NO FRAMES All Classes
DETAIL: FIELD | CONSTR | METHOD

javax.slee
Class Address

java.lang.Object
  └ javax.slee.Address

All Implemented Interfaces:
java.io.Serializable

public class Address
extends java.lang.Object
implements java.io.Serializable

The Address class encapsulates an address that can be used by the SLEE. An address consists of the following two components:

- Address Plan - the plan that defines the address or numbering type and the format of the address string.
- Address String - some combination of alphanumeric characters and symbols that identify an address within the domain identified by the address plan.

An Address object is immutable. Once it has been created the values contained by it cannot change.

Figura 4.38: Documentación de la clase *javax.slee.Address*.

Finalmente, se crea correctamente el perfil *fmfmprofile*:

Atributo	Address Plan	Address String
Addresses	SIP:	sip:jain1@rhinoslee.proxy.sip
BackupAddresses	SIP:	sip:jain2@rhinoslee.proxy.sip

se aplican los cambios:

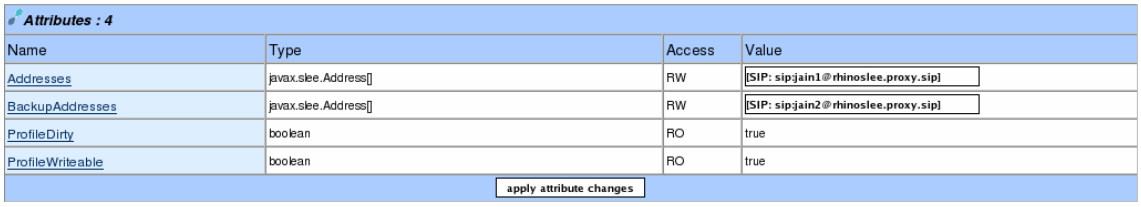


Figura 4.39: Creación correcta del perfil *fmfmprofile*.

y se asignan los atributos (*commitProfile*):

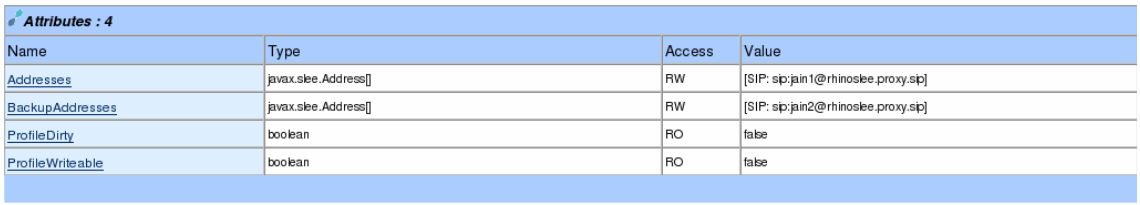


Figura 4.40: Sin errores en el *commitProfile*.

4.5.1. Comprobación del servicio FMFM

Para la comprobación de este servicio, simplemente hay que hacer uso de los clientes SIP. El perfil está configurado de tal manera que si el usuario jain1 recibe una llamada, y no está disponible, dicha llamada se encaminará hacia el usuario jain2. Por tanto, habrá que arrancar el cliente jain2 y dejar apagado el cliente jain1 o bien arrancarlo, pero poniéndolo en estado ocupado, ausente, no molestar o vengo enseguida.

Finalmente, se arranca un nuevo cliente desde el equipo que tiene el SLEE para poder hacer la llamada a jain1 y ver como se redirecciona hacia jain2. Por tanto, para este caso se usará una configuración que resulta ser una mezcla de las vistas en la Figura 4.10 y la Figura 4.27. De todas formas, con la configuración de la Figura 4.27 hubiera sido suficiente, teniendo en cuenta que el cliente jain1 no tiene porqué ser arrancado. El resultado final es un éxito, por lo que se da por concluida esta demostración.

4.5.2. Utilización de una base de datos para el servicio FMFM

Ya se tiene cumplido el segundo objetivo, y a continuación se propone el mejorarlo.

Se ha visto como se ha hecho uso de los perfiles para poder utilizar el servicio *fmfm*. Para ello se ha utilizado la consola web, y se ha podido comprobar como la gestión, a pesar de resultar muy cómoda si se compara con el uso de una simple consola de comandos, puede resultar algo lenta y compleja si se compara con la utilización de una base de datos. Por dicho motivo, se propone el uso de una base de datos en la que se almacene la información necesaria para la utilización del servicio *fmfm*. En concreto se utilizará la base de datos que se crea por defecto durante la instalación del Rhino SLEE.

Para hacer uso de la base de datos, se tendrá que conseguir que la aplicación haga su búsqueda en la base de datos en lugar de en una tabla de perfiles. Es la clase *FindMeFollowMeSbb* la que se encargará de realizarlo, así que se editará el código fuente:

```
[vhros@rhinoslee ~]$ cd $RHINO_HOME/examples/sip/fmfm/
[vhros@rhinoslee profile]$ gedit FindMeFollowMeSbb.java
```

En este fichero (Figura 4.41), se puede observar como se busca en *"fmfmProfileTableName"* cuando se trata de una búsqueda en perfiles. Si se acude al fichero de configuración *sip.properties* (Figura 4.42) se puede ver como ese nombre coincide con una variable a la que se le da el valor de *FMFMSubscribers*, que precisamente concuerda con el nombre de la tabla de perfiles creada cuando se desplegó el servicio *fmfm* en el punto 4.5. De esta forma se comprueba que todo tiene coherencia.

```
public abstract class FindMeFollowMeSbb extends OCSipSbb {

    public void setSbbContext(SbbContext context) {
        super.setSbbContext(context);
        try {
            Context myEnv = (Context) new InitialContext().lookup("java:comp/env");
            boolean useJDBC = ((Boolean)myEnv.lookup("fmfmUseJDBC")).booleanValue();

            if (useJDBC) {
                String dsName = (String) myEnv.lookup("fmfmDataSourceName");
                DataSource ds = (DataSource) myEnv.lookup(dsName);
                lookup = new JDBCSubscriberLookup(ds, getSbbTracer());
            }
            else { // profiles
                String profileTableName = (String) myEnv.lookup("fmfmProfileTableName");
                lookup = new FMFMProfileLookup(profileTableName);
            }
        } catch (NamingException ne) {
            severe("unable to set SBB context", ne);
        }
    }
```

Figura 4.41: Fichero *FindMeFollowMeSbb.java*

Volviendo al tema que aborda este apartado, se observa en la Figura 4.41 que para que se produzca el uso de la API JDBC, la variable *useJDBC* tiene que tener el valor *true* (Figura 4.42). Con JDBC lo que se tiene es una API que permite la ejecución

de operaciones sobre bases de datos desde el lenguaje de programación Java. Pues bien, lo que queda es crear una tabla[11] en la base de datos.

```
# FMFM SBB configuration
# Uses profiles for subscriber data by default, enable JDBC here.
FMFM_USE_JDBC=true
FMFM_PROFILE_TABLE_NAME=FMFMSubscribers
FMFM_DATASOURCE_NAME=jdbc/FMFMSubscribers
```

Figura 4.42: Activación del uso de JDBC.

Para crear la tabla *findme*, lo primero será editar el *script* de inicio de gestión de bases de datos *init-management-db.sh* y añadirle al final las líneas de código dadas en el fichero *$RHINO_HOME/examples/sip/fmfm/jdbc/postgres_findme_db.sql*.

```
[vhros@rhinoslee ~]$ cd $RHINO_HOME
[vhros@rhinoslee rhino]$ gedit init-management-db.sh
```

Lo añadido finalmente al *script* será lo siguiente:

```
create table findme ( -- for SIP Find Me Follow Me Service
  sipaddress varchar(80) not null,
  backup_sipaddress varchar(80) not null,
  seq_no integer,
  primary key (sipaddress, seq_no)
);
COMMENT ON TABLE findme IS 'FMFM Subscribers';
```

Figura 4.43: Modificación del *init-management-db.sh* para crear la tabla *findme*.

Finalmente, se arrancará el *script* habiendo iniciado primero el servidor postgreSQL, de tal forma que la tabla *findme* será creada junto con el resto de tablas indicadas en el *script*.

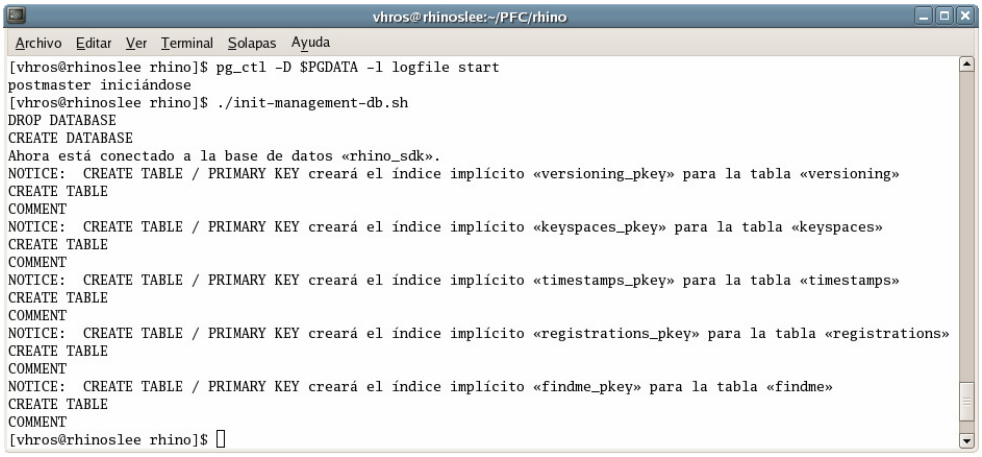


Figura 4.44: Creación de la tabla *findme*.

[11] El nombre de esa tabla será *findme*, ya que como se puede ver en el fichero *JDBCSubscriberLookup.java*, la selección de tuplas se hace de la siguiente forma: *"select * from findme where sipaddress = ? order by seq_no"*.

Ya se puede abrir el gestor de base de datos (Figura 4.45) y comprobar como se ha añadido la tabla *findme* a la base de datos *rhino_sdk*.

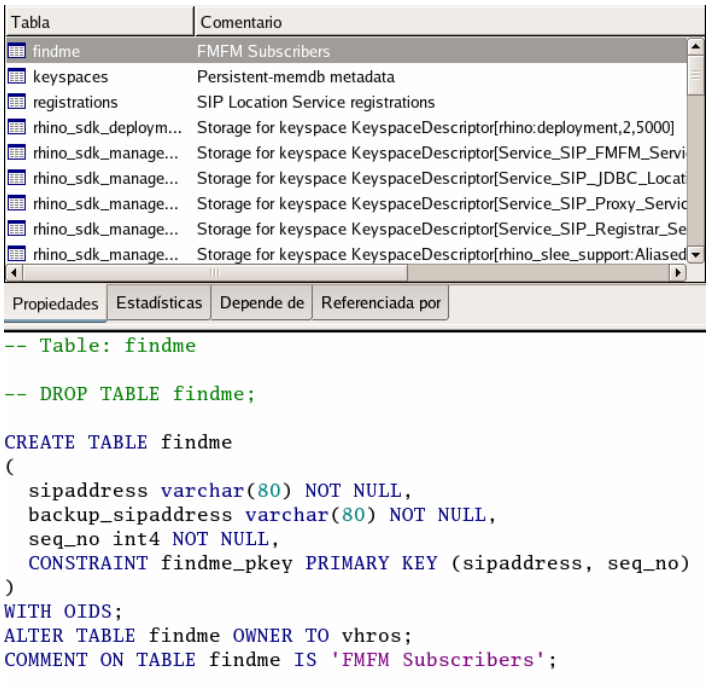
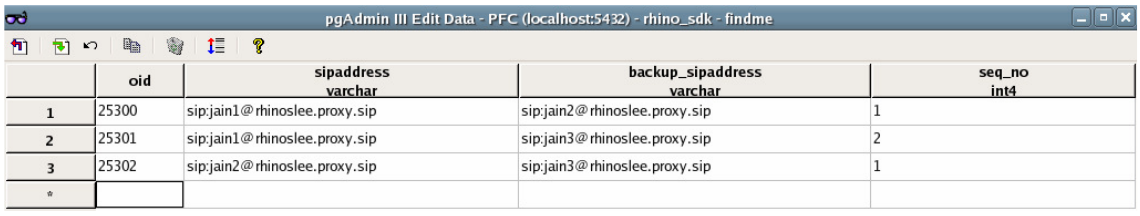


Figura 4.45: Tabla *findme* en la base de datos *rhino_sdk*.

A continuación, se visualiza la tabla y se añaden las tuplas equivalentes a lo que ya se hizo en su momento con los perfiles. A modo de ejemplo, en la Figura 4.46, se observa como se tienen los mismos campos a rellenar que en caso de los perfiles, con la salvedad de que aquí las direcciones ya son consideradas como direcciones SIP, y por tanto no hay que escribir el plan de dirección, simplemente la dirección SIP en sí misma (Address String). Además, se puede apreciar la existencia de un nuevo campo de nombre *seq_no*, que no es más que un entero para indicar la "prioridad[12]" de las direcciones de reserva (backup addresses), es decir, estando el usuario llamado no disponible, en que orden se irán intentando las llamadas a las direcciones de reserva. En el ejemplo (Figura 4.46), si se produce una llamada a jain1 y no está disponible, la llamada se redireccionará a jain2, y de no estar tampoco disponible se llamará a jain3.



	oid	sipaddress varchar	backup_sipaddress varchar	seq_no int4
1	25300	sip:jain1@rhinoslee.proxy.sip	sip:jain2@rhinoslee.proxy.sip	1
2	25301	sip:jain1@rhinoslee.proxy.sip	sip:jain3@rhinoslee.proxy.sip	2
3	25302	sip:jain2@rhinoslee.proxy.sip	sip:jain3@rhinoslee.proxy.sip	1
*				

Figura 4.46: Registro para el uso del servicio *fmfm* (Base de datos *rhino_sdk*).

[12] Lo de poner "prioridad" entre comillas, no es más que debido a que no se trata exactamente de una prioridad, sino que es un número de secuencia tal que la aplicación sepa en que orden usar las direcciones de reserva en caso de haber varias.

4.5.3. Comprobación de la aplicación usando la base de datos.

Esta comprobación se hará tal y como se hizo en el punto 4.5.1. El resultado obtenido es el mismo, pero con un indudable incremento en la facilidad de gestión.

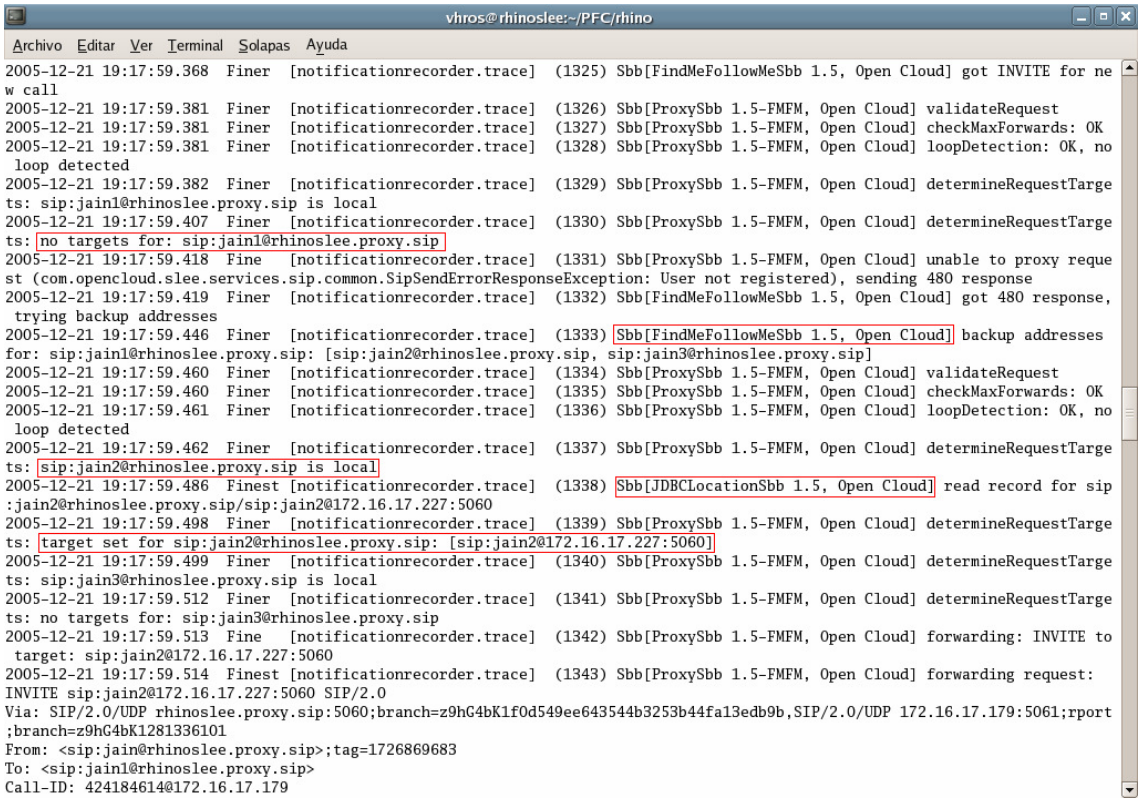


Figura 4.47: Trazas en la utilización del servicio *fmfm*.

En la Figura 4.47 se ven claramente los pasos seguidos. Se produce una llamada a jain1, el *proxy* determina que ese usuario no está disponible, y por tanto salta al servicio *fmfm*, el cual determina las direcciones de reserva de jain1. La de mayor "prioridad" es la correspondiente al cliente jain2, y es entonces cuando se salta al servicio de localización para saber la dirección exacta y poder así encaminar la llamada.

Por último, recordar que el servicio *fmfm* tendrá que haber sido nuevamente desplegado para que se actualicen los cambios referentes al uso de JDBC.

4.5.4. Liberación de la base de datos

Siguiendo con la mejora, se propuso hacer una nueva en cuanto a la libertad de la base de datos utilizada. Con esto se quiere decir que hasta ahora lo que se ha hecho es crear una tabla correspondiente al nuevo servicio incorporado, pero esa tabla se ha añadido a la base de datos *rhino_sdk*, que es la base de datos que usa el SLEE y lo lógico es que sea administrada por el administrador del propio SLEE.

Lo ideal sería que el nuevo servicio incorporado se libere de la base de datos del SLEE, para que de esta forma sea el mismo administrador del servicio el que se encargue de administrar su propia base de datos.

Por tanto, lo que se pretende hacer es crear una base de datos externa, aunque teniendo en cuenta que la gestión de transacción de la conexión JDBC es controlada por el SLEE.

```
<jdbc>
  <jndi-name>ExternalDataSource</jndi-name>
  <datasource-class>org.postgresql.jdbc2.optional.SimpleDataSource</datasource-class>
  <parameter>
    <param-name>serverName</param-name>
    <param-type>java.lang.String</param-type>
    <param-value>localhost</param-value>
  </parameter>
  <parameter>
    <param-name>portNumber</param-name>
    <param-type>java.lang.Integer</param-type>
    <param-value>5432</param-value>
  </parameter>
  <parameter>
    <param-name>databaseName</param-name>
    <param-type>java.lang.String</param-type>
    <param-value>fmfm</param-value>
  </parameter>
  <parameter>
    <param-name>user</param-name>
    <param-type>java.lang.String</param-type>
    <param-value>fmfmslee</param-value>
  </parameter>
  <parameter>
  <param-name>password</param-name>
  <param-type>java.lang.String</param-type>
  <param-value>********</param-value>
  </parameter>
  <connection-pool>
    <max-connections>15</max-connections>
    <idle-check-interval>0.0</idle-check-interval>
  </connection-pool>
</jdbc>
```

Figura 4.48: Adición de una base de datos externa en el fichero *rhino-config.xml*.

Como ya se pudo ver en la Figura 4.41, correspondiente al código fuente de la clase *FindMeFollowMeSbb*, resulta que cuando se hace uso de la API JDBC, lo que se hace es buscar en *"fmfmDataSourceName"*, que como se puede apreciar en la Figura 4.42, correspondiente al fichero de configuración *sip.properties*, se corresponde con una variable cuyo valor es *jdbc/FMFMSubscribers*. Esto se hace gracias a la API JNDI, la cual permite a sus clientes descubrir y buscar datos y objetos a través de un nombre.

Para añadir lo que son bases de datos externas, se hará mediante la configuración del fichero *$RHINO_HOME/config/rhino-config.xml*. En él se tendrá un elemento <ejb-resorces> que debe tener al menos un elemento <jdbc> presente. Lo que se hará es añadir otro elemento <jdbc>, cuyo elemento <jndi-name> *"ExternalDataSource"* identificará al nombre JNDI al que será ligada la base de datos dentro del árbol JNDI (Figura 4.48).

Finalmente, lo que se hará es enlazar el nombre buscado por la clase *FindMeFollowMeSbb* con la nueva base de datos. Para entender esto hay que acudir al fichero *$RHINO_HOME/examples/sip/fmfm/META-INF/oc-sbb.jar.xml*, el cual habrá que modificar indicando que el nombre JNDI a buscar en los elementos <jdbc> será *ExternalDataSource"* (Figura 4.49), que precisamente se corresponde con la base de datos externa creada.

```
<?xml version="1.0"?>

<!DOCTYPE oc-sbb-jar PUBLIC "-//Open Cloud Ltd.//DTD JAIN SLEE SBB Extension 1.0//EN" "http://
www.opencloud.com/slee/oc-sbb-jar_1_0.dtd">

<oc-sbb-jar>
  <sbb id="fmfm">
    <resource-ref>
      <res-ref-name>@FMFM_DATASOURCE_NAME@</res-ref-name>
      <res-type>javax.sql.DataSource</res-type>
      <res-auth>Container</res-auth>
      <res-sharing-scope>Shareable</res-sharing-scope>
      <res-jndi-name>jdbc/ExternalDataSource</res-jndi-name>
    </resource-ref>
  </sbb>
  <security-permission>
    <!-- The proxy may need to do a hostname lookup when checking headers -->
    <security-permission-spec>
          grant {
            permission java.net.SocketPermission "*:1024-", "resolve";
          };
    </security-permission-spec>
  </security-permission>
</oc-sbb-jar>
```

Figura 4.49: Fichero *oc-sbb-jar.xml*.

Resumiendo, el SBB correspondiente al servicio *fmfm* será capaz de obtener la referencia de un objeto a través de una búsqueda JNDI. Sin embargo, la localización JNDI no es accesible directamente por el SBB, así que el SBB está ligado al nombre JNDI en su llamado descriptor de despliegue (*oc-sbb-jar.xml*). De esta forma, se ha conseguido que cuando el nuevo servicio quiera hacer uso de una base datos, acudirá a una base de datos externa gestionada por el propio desarrollador del servicio, en lugar de acudir a la base de datos local del SLEE.

En el fichero *rhino-config.xml* (Figura 4.48) se introducen todos los datos concernientes a la base de datos externa, que en este caso está ubicada en la misma máquina (*localhost*) por comodidad.

Así pues, se inicia servidor PostgreSQL y se crea una base de datos de acuerdo a los datos del fichero *rhino-config.xml*. Para ello, primero se tendrá que crear un usuario que será el que gestione la base de datos. Todo los pasos se pueden ver en la Figura 4.50.

De esta forma, si ahora se abre el gestor de bases de datos, se comprueba como se tiene una nueva base de datos llamada *fmfm* y perteneciente al usuario *fmfmslee* (Figura 4.51).

Finalmente, decir que el funcionamiento del servicio *fmfm* será el mismo, salvo que ahora la tabla *findme* no se encuentra en la base de datos *rhino_sdk*, sino en la base de datos *fmfm* (Figura 4.52).

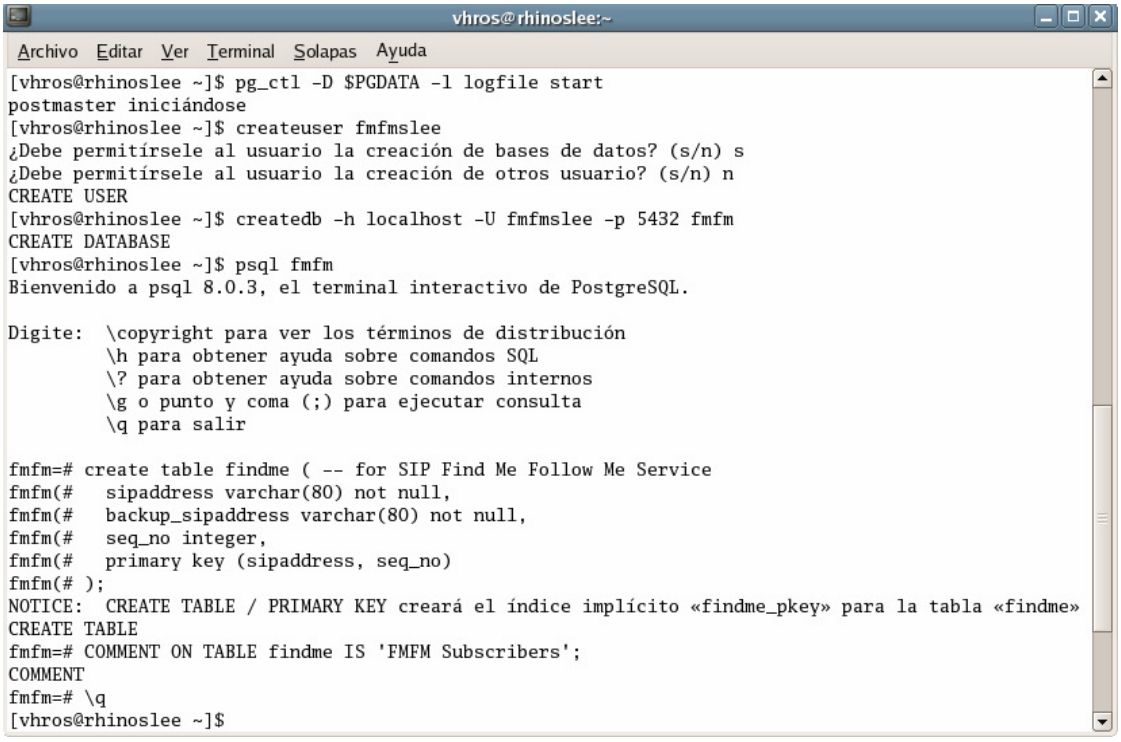


Figura 4.50: Creación de la base de datos independiente.

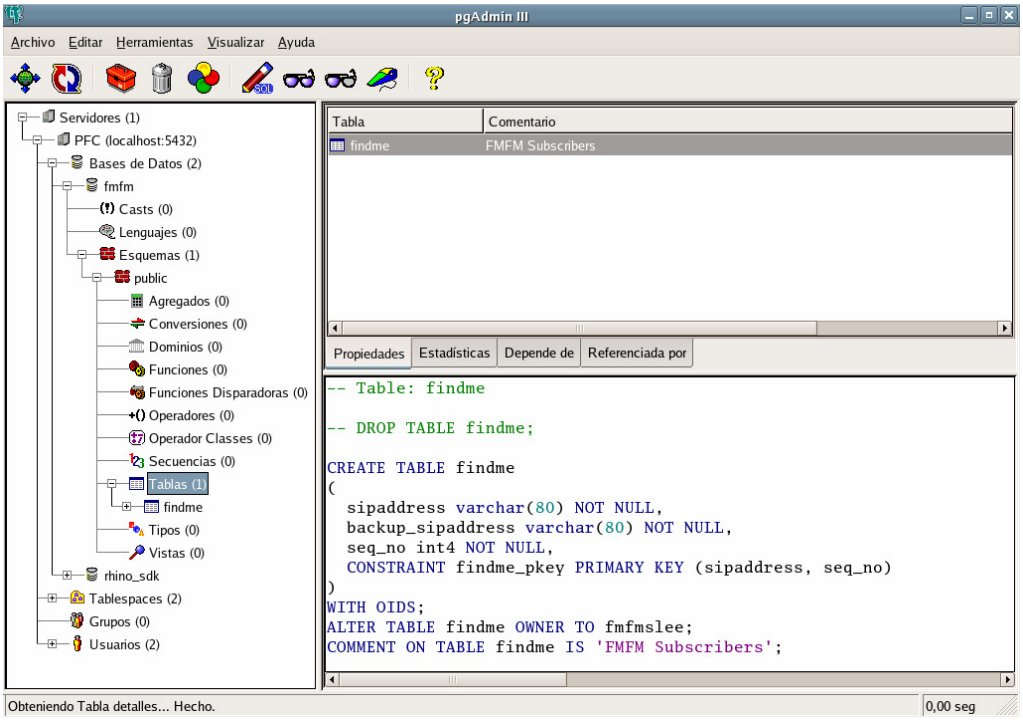
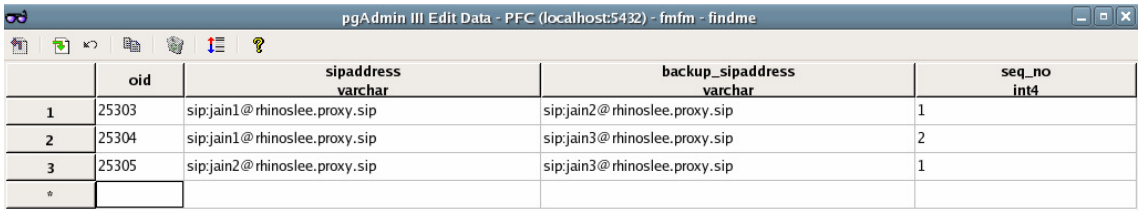


Figura 4.51: Base da datos *fmfm* vista desde el pgAdmin III.



	oid	sipaddress varchar	backup_sipaddress varchar	seq_no int4
1	25303	sip:jain1@rhinoslee.proxy.sip	sip:jain2@rhinoslee.proxy.sip	1
2	25304	sip:jain1@rhinoslee.proxy.sip	sip:jain3@rhinoslee.proxy.sip	2
3	25305	sip:jain2@rhinoslee.proxy.sip	sip:jain3@rhinoslee.proxy.sip	1
*				

Figura 4.52: Registro para el uso del servicio *fmfm* (Base de datos *fmfm*).

4.6. Aportaciones

Una vez concluidos los dos primeros objetivos del proyecto, ya se estaba en una fecha en la que el proyecto Mobicents había avanzado lo suficiente, por lo que se disponía de un nuevo SLEE donde trabajar. Ante la imposibilidad de continuar la línea del proyecto con la pretensión de desplegar aplicaciones OSA/Parlay dentro de un SLEE, debido a la inexistencia del adaptador de recurso correspondiente[13], se propuso el aplicar los conocimientos adquiridos en participar en el proyecto Mobicents e intentar realizar algunas aportaciones de mejora, a parte de ambicionar el avanzar en el conocimiento haciendo uso de esta plataforma de código abierto.

Comentar que Mobicents da la posibilidad de utilizar tanto Linux como Windows, y dado que con Rhino se ha trabajado exclusivamente en Linux, se decidió hacer uso de Windows para familiarizarse con un entorno diferente, a pesar del manejo adquirido y de las ventajas que ofrece Linux para trabajar en algo referente al desarrollo de servicios.

Antes de pasar a comentar las aportaciones realizadas al proyecto Mobicents, se planteó el hecho de comprobar si se podía hacer algo similar a lo realizado con la plataforma Rhino, es decir, intentar establecer una comunicación entre 2 clientes SIP[14]. El resultado obtenido fue satisfactorio. Además, la prueba se pudo hacer con un único ordenador, ya que se utilizaron clientes diferentes (SJphone y Windows Messenger) y no había conflicto en la llamada, aunque al final sea como llamarse a uno mismo. No se van a explicar los pasos seguidos, ya que se considera que con la información existente en la página *wiki* correspondiente al ejemplo de comunicación extremo a extremo en Mobicents [37] y con la experiencia adquirida con la configuración de los clientes Linphone en el punto 4.4.5, es más que suficiente para realizar esta prueba con éxito.

[13] Tan sólo la compañía Arpona [33] anunció el 27 de Octubre de 2005 que iba a trabajar con Mobicents en un Adaptador de Recurso Parlay para JAIN SLEE y que lo harían con el objetivo de anunciar su disponibilidad durante el primer cuatrimestre de 2006.

[14] Recordar que si se tiene algún cortafuegos en funcionamiento en el servidor Mobicents, se debe configurar para aceptar las llamadas de los clientes de la red.

4.6.1. Servicio Wake Up Call

En este apartado se explicará la aportación hecha al proyecto Mobicents con relación a este servicio. También se desplegará dicho servicio y se aprovechará para explicar ciertos aspectos del código a la hora de poner en práctica la especificación JSLEE.

Cuando se comenzó con el uso de la plataforma Mobicents, no había muchas iniciativas puestas en práctica en el proyecto. Sin embargo, si existían y existen buenas ideas e intentos constantes de seguir mejorando. Entre esas ideas estaba el servicio *Wake Up Call*, que consiste básicamente en aceptar mensajes, con un formato específico, de clientes SIP registrados y responderles a su petición. A simple vista puede parecer sencillo, pero es la base para una gran gama de posibles servicios de telefonía a desarrollar, así como una manera bastante eficaz de entender el funcionamiento del SLEE.

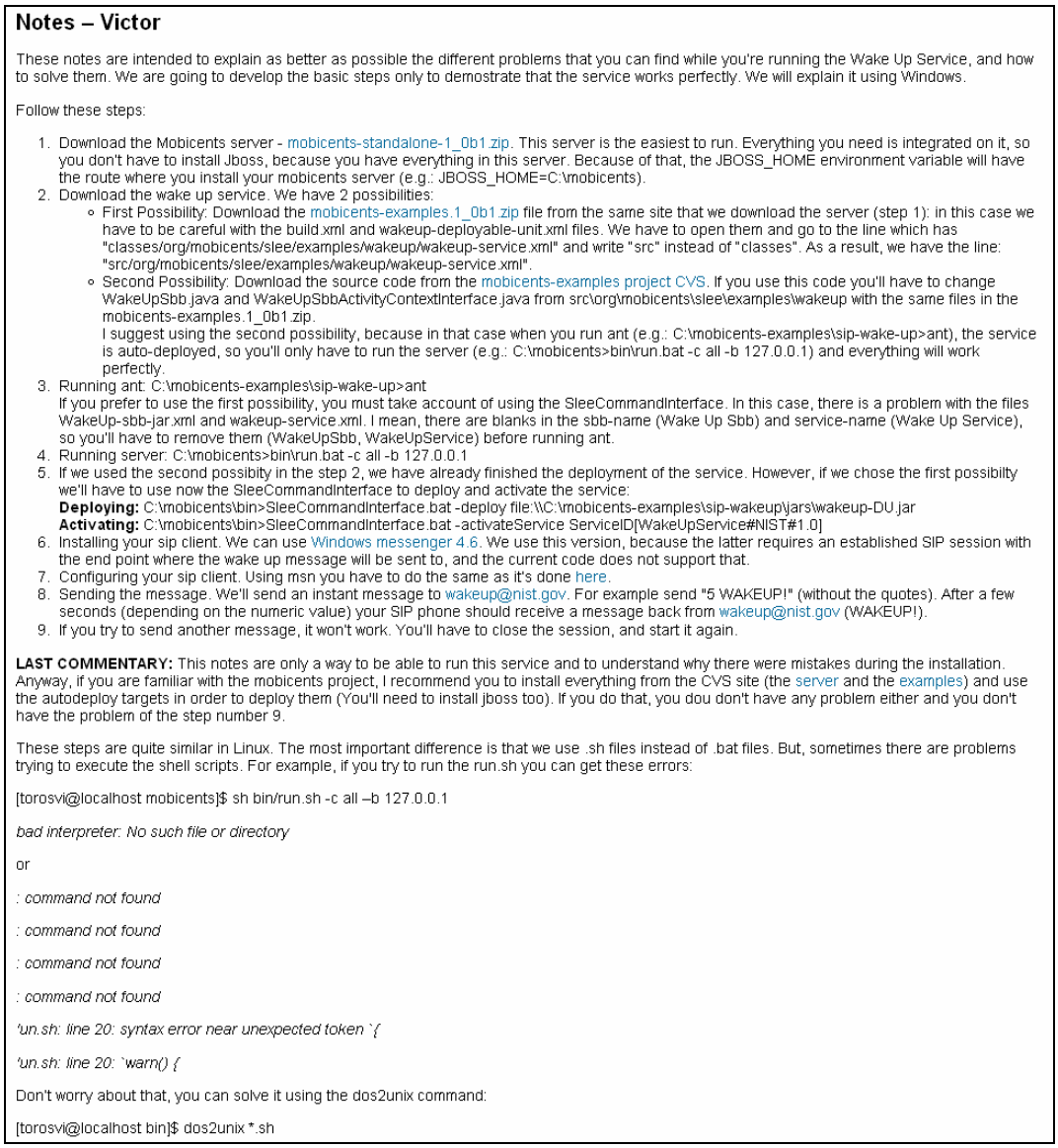
El poner en práctica este servicio fue más dificultoso de lo esperado en un principio, hasta el punto de que hubo un momento en que se pensó dejarlo aparcado. Pero es aquí donde entra en juego la gran virtud de este proyecto, y es que existe una comunidad de usuarios y desarrolladores. Se acudió al foro del proyecto [38] y se observó que había un tema llamado "*wake up call example*" en el que se pudo comprobar que habían más usuarios con el mismo problema, es decir, la imposibilidad de desplegar el servicio. En fin, el problema seguía existiendo, pero al menos había gente con la que intercambiar dudas y posibles soluciones.

Después de varios días trabajando en ello, se consiguió solucionar el problema. Tras postear en el hilo habilitado para este tema en el foro del proyecto [39], la respuesta fue inmediata y el interés general por saber como hacer funcionar el servicio. Así que se tuvo la idea de publicar una página *wiki* en el proyecto, con las notas necesarias para desplegar el servicio *Wake Up Call* sin ninguna dificultad. Para ello, se contactó con el fundador del proyecto Mobicents (Ivalin Ivanov), al que se le propuso la idea, la cual le pareció interesante y animó a llevarla a cabo. Finalmente, se comenzó con el trabajo de redacción de dichas notas (por supuesto en inglés), y el resultado final puede verse en la página *wiki* correspondiente al servicio *Wake Up Call* [40], o bien en la Figura 4.53.

La página tuvo muy buena aceptación, y desde su publicación (29 de Noviembre de 2005) hasta la fecha de la última actualización (15 de Diciembre de 2005) no se han planteado más dudas al respecto del servicio *Wake Up Call* en el foro del proyecto Mobicents.

A los pocos días de realizar la aportación en la página *wiki*, se recibió un e-mail de Ivelin Ivanov (2 de Diciembre de 2005), en el que tras felicitar por el trabajo realizado, decía lo siguiente: *"It seems like you are ready to receive the Developer Role and begin working your way to the core team"*. Gracias a esto, y tras leer el contrato del contribuyente (Anexo D) y aceptarlo, ya se estaba en disposición de hacer aportaciones y modificaciones de código con total libertad, y con la única recomendación de

consultar previamente en el foro si dicha modificación o aportación es aceptada por la comunidad de desarrolladores.



Notes – Victor

These notes are intended to explain as better as possible the different problems that you can find while you're running the Wake Up Service, and how to solve them. We are going to develop the basic steps only to demostrate that the service works perfectly. We will explain it using Windows.

Follow these steps:

1. Download the Mobicents server - mobicents-standalone-1_0b1.zip. This server is the easiest to run. Everything you need is integrated on it, so you don't have to install Jboss, because you have everything in this server. Because of that, the JBOSS_HOME environment variable will have the route where you install your mobicents server (e.g.: JBOSS_HOME=C:\mobicents).
2. Download the wake up service. We have 2 possibilities:
   - First Possibility: Download the mobicents-examples.1_0b1.zip file from the same site that we download the server (step 1): in this case we have to be careful with the build.xml and wakeup-deployable-unit.xml files. We have to open them and go to the line which has "classes/org/mobicents/slee/examples/wakeup/wakeup-service.xml" and write "src" instead of "classes". As a result, we have the line: "src/org/mobicents/slee/examples/wakeup/wakeup-service.xml".
   - Second Possibility: Download the source code from the mobicents-examples project CVS. If you use this code you'll have to change WakeUpSbb.java and WakeUpSbbActivityContextInterface.java from src\org\mobicents\slee\examples\wakeup with the same files in the mobicents-examples.1_0b1.zip.
     I suggest using the second possibility, because in that case when you run ant (e.g.: C:\mobicents-examples\sip-wake-up>ant), the service is auto-deployed, so you'll only have to run the server (e.g.: C:\mobicents>bin\run.bat -c all -b 127.0.0.1) and everything will work perfectly.
3. Running ant: C:\mobicents-examples\sip-wake-up>ant
   If you prefer to use the first possibility, you must take account of using the SleeCommandInterface. In this case, there is a problem with the files WakeUp-sbb-jar.xml and wakeup-service.xml. I mean, there are blanks in the sbb-name (Wake Up Sbb) and service-name (Wake Up Service), so you'll have to remove them (WakeUpSbb, WakeUpService) before running ant.
4. Running server: C:\mobicents>bin\run.bat -c all -b 127.0.0.1
5. If we used the second possibilty in the step 2, we have already finished the deployment of the service. However, if we chose the first possibilty we'll have to use now the SleeCommandInterface to deploy and activate the service:
   Deploying: C:\mobicents\bin>SleeCommandInterface.bat -deploy file:\\C:\mobicents-examples\sip-wakeup\jars\wakeup-DU.jar
   Activating: C:\mobicents\bin>SleeCommandInterface.bat -activateService ServiceID[WakeUpService#NIST#1.0]
6. Installing your sip client. We can use Windows messenger 4.6. We use this version, because the latter requires an established SIP session with the end point where the wake up message will be sent to, and the current code does not support that.
7. Configuring your sip client. Using msn you have to do the same as it's done here.
8. Sending the message. We'll send an instant message to wakeup@nist.gov. For example send "5 WAKEUP!" (without the quotes). After a few seconds (depending on the numeric value) your SIP phone should receive a message back from wakeup@nist.gov (WAKEUP!).
9. If you try to send another message, it won't work. You'll have to close the session, and start it again.

LAST COMMENTARY: This notes are only a way to be able to run this service and to understand why there were mistakes during the installation. Anyway, if you are familiar with the mobicents project, I recommend you to install everything from the CVS site (the server and the examples) and use the autodeploy targets in order to deploy them (You'll need to install jboss too). If you do that, you dou don't have any problem either and you don't have the problem of the step number 9.

These steps are quite similar in Linux. The most important difference is that we use .sh files instead of .bat files. But, sometimes there are problems trying to execute the shell scripts. For example, if you try to run the run.sh you can get these errors:

[torosvi@localhost mobicents]$ sh bin/run.sh -c all –b 127.0.0.1

bad interpreter: No such file or directory

or

: command not found

: command not found

: command not found

: command not found

'un.sh: line 20: syntax error near unexpected token `{

'un.sh: line 20: `warn() {

Don't worry about that, you can solve it using the dos2unix command:

[torosvi@localhost bin]$ dos2unix *.sh

Figura 4.53: Notas para el despliegue del servicio *Wake Up Call*.

4.6.1.1. Explicación

Como se ha dicho anteriormente, el servicio *Wake Up Call* lo que hará es aceptar mensajes, provenientes de usuarios registrados, en un formato específico (“Tiempo” + “Texto”). Esto significa, un número seguido de un espacio en blanco y de algún texto. Cualquier otro formato del mensaje será erróneo para la aplicación. Esto recuerda, por ejemplo, al envío de SMSs en los concursos de televisión y radio, en los que hay que escribir unos caracteres concretos al principio, seguidos de un espacio en blanco y a continuación la respuesta a la pregunta del concurso.

En este servicio, el número indica un retardo en segundos y lo que hace es disparar un temporizador, el cual empieza a contar desde el momento en que el servicio *Wake Up Call* recibe el mensaje. Justo después de expirar dicho retardo, el temporizador origina un mensaje automático que será enviado al cliente que realizó la petición y que contendrá el texto del mensaje original.

A continuación se explicará con un mapa (Figura 4.54) los pasos seguidos en el funcionamiento del servicio *Wake Up Call*:

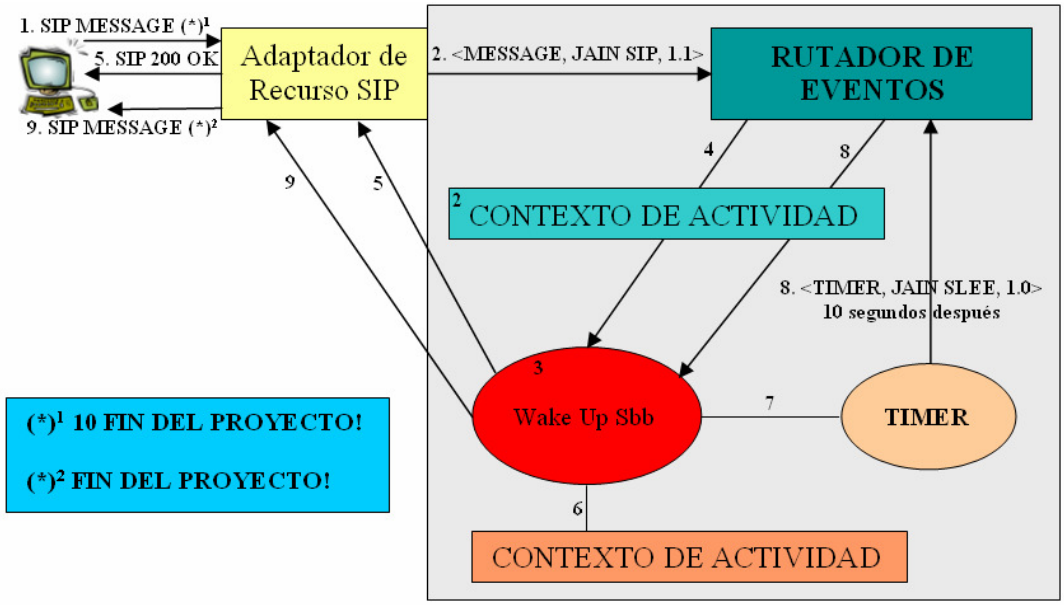


Figura 4.54: Pasos seguidos en el servicio *Wake Up Call*.

1. Llega un SIP MESSAGE al Adaptador de Recurso SIP (SIP RA).
2. El Adaptador de Recurso crea las representaciones Java del mensaje y lo dispara dentro del servidor de aplicación JSLEE. Es en ese momento cuando se crea el Contexto de Actividad sobre el que se dispara el evento.
3. Entra en juego el rutador de eventos (event router), el cual funciona recibiendo eventos de todos los productores de eventos y entregándolos a las múltiples instancias de componentes (SBBs) que estén interesadas en cada evento concreto. Resulta que el evento correspondiente a recibir un SIP MESSAGE es un evento inicial. Por tanto, una nueva entidad del SBB raíz es creada para procesar dicho evento, y además el SLEE une la nueva entidad creada con el Contexto de Actividad sobre el cual fue disparado el evento inicial.
4. Una vez que la nueva entidad SBB es unida al Contexto de Actividad, la entidad del SBB raíz recibirá el evento cuando el rutador de eventos haga entrega del mismo a las entidades SBB unidas al Contexto de Actividad, que en este caso se trata solamente de una.

5. El SBB utiliza el Contexto de Actividad creado para enviar un mensaje de acuse de recibo al cliente.
6. Se crea un Contexto de Actividad sobre el cual será disparado el evento temporizador, y que además servirá para almacenar y posteriormente retirar los datos de *contacto* (dirección del cliente que hace la petición) y *cuerpo* del mensaje (texto que envía el cliente).
7. Programación del temporizador utilizando el valor numérico en el cuerpo del mensaje SIP.
8. Se dispara el evento, y el rutador de eventos lo dirigirá al SBB unido al Contexto de Actividad sobre el que se disparó dicho evento.
9. Envío de la respuesta a la petición inicial (utilizando la dirección de *contacto* y el *cuerpo* del mensaje almacenados en el paso 6), que irá primero al Adaptador de Recurso, para finalmente dirigirse al Cliente SIP.

En la Figura 4.55 se muestra a modo de lista el contenido de la clase de Java correspondiente al SBB del servicio *Wake Up Call* (*WakeUpSbb.java*).

Figura 4.55: Listado del contenido del SBB.

Seguidamente se pasará a dar una explicación más pormenorizada de este servicio, sentando las bases para que cualquier lector del proyecto, interesado en estos temas, se inicie en un estudio más profundo de la tecnología JAIN SLEE.

Para la realización de este servicio, se hará uso de los siguientes bloques de construcción JSLEE:

1. **wakeup-service.xml:** Un Servicio SLEE.
2. **WakeUpSbb:** Un Bloque de Construcción de Servicio.
3. **WakeUpSbbActivityContextInterface:** Una Interfaz de Contexto de Actividad.

Como es lógico, se necesitará un punto de entrada al servicio. Es aquí donde entra en juego el fichero de definición del servicio (*wakeup-service.xml*), el cual informa al contenedor del SLEE de que se está desplegando y activando un nuevo servicio que aceptará un conjunto de eventos dados. El contenedor del SLEE se asegurará de entregar todos los eventos a los que el servicio se suscribe. Para ser más precisos, el servicio solamente declara el SBB raíz (*WakeUpSbb*), que suscribirá uno tras otro los eventos del SLEE. Esto es una abstracción adicional, que permite a un SBB ser utilizado por múltiples servicios u otros SBBs.

El SBB raíz, del que se ha hablado, vendrá implementado en el fichero *WakeUpSbb.java*. Éste será el responsable de aceptar un evento SIP MESSAGE, analizarlo sintácticamente y programar un temporizador. Además, también estará a la escucha de eventos de temporizador (timer events) y enviará un SIP MESSAGE en respuesta a la petición de un cliente.

En relación al *WakeUpSbbActivityInterface*, decir que se trata de una vista dentro de un Contexto de Actividad del SLEE a través de una interfaz de Java. Esta interfaz tiene dos propiedades, que son el *contacto* (contact) y el *cuerpo* (body). El *contacto* lo que hará es almacenar la dirección física del cliente SIP, mientras que el *cuerpo* será el que almacene el mensaje de texto enviado por el usuario. Primero se pedirá al contenedor del SLEE la creación de una instancia de objeto concreta que implemente la interfaz. Será entonces cuando se unirá a un temporizador, se programará y finalmente se retirarán los datos almacenados cuando el evento temporizador se dispare.

Seguidamente, se verán algunas pinceladas del código para entender mejor como funcionan las cosas.

A continuación se muestra el contenido completo del descriptor de servicio (*wakeup-service.xml)*:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE service-xml PUBLIC "-//Sun Microsystems, Inc.//DTD JAIN SLEE Service 1.0//EN"
                             "http://java.sun.com/dtd/slee-service_1_0.dtd">
<service-xml>
    <service>
```

```
        <description>Wake up service</description>
        <service-name>Wake Up Service</service-name>
        <service-vendor>NIST</service-vendor>
        <service-version>1.0</service-version>
        <root-sbb>
            <description>This Sbb send a wake up call to the user.</description>
            <sbb-name>Wake Up Sbb</sbb-name>
            <sbb-vendor>NIST</sbb-vendor>
            <sbb-version>1.0</sbb-version>
        </root-sbb>
        <default-priority>0</default-priority>
    </service>
</service-xml>
```

Hay dos cosas importantes a destacar, que son:

1. El servicio es identificado en el contenedor del SLEE únicamente por la tupla *"service-name, service-vendor, service-version"*.
2. El SBB raíz debe ser referenciado a través de un identificador único, compuesto por la tupla *"sbb-name, sbb-vendor, sbb-version"*.

El fichero descriptor de servicio es empaquetado en una Unidad de Despliegue del SLEE junto con el archivo del SBB. Este paquete será el *wakeup-DU.jar* y tendrá la siguiente estructura:

- **/wakeup-service.xml.**
- **/META-INF/deployable-unit.xml:** Este fichero simplemente lista los archivos del SLEE y los servicios contenidos en el Deployment Unit (DU), que son en este caso *jars/WakeUp-sbb.jar* y *wakeup-service.xml*.
- **/jars/WakeUp-sbb.jar.**

Ahora se hablará del SBB, el cual está implementado en el fichero *WakeUpSbb.java* y es anunciado al SLEE a través del fichero descriptor *WakeUp-sbb-jar.xml*. Ambos ficheros serán empaquetados en el archivo *WakeUp-sbb.jar* con la siguiente estructura:

- **/META-INF/WakeUp-sbb-jar.xml:** Fichero descriptor del SBB.
- **/org/mobicents/*:** Contendrá las clases de Java.

El fichero descriptor *WakeUp-sbb-jar.xml* declara lo siguiente:

- **El identificador único del SBB:** Wake Up Sbb, NIST, 1.0.
- **org.mobicents.slee.examples.wakeup.WakeUpSbb:** Se trata de la implementación de la clase de Java. Darse cuenta de que es realmente una clase abstracta. El SLEE crea una subclase concreta y la instancia en el arranque. Los métodos abstractos implementados por el SLEE proporcionan una información contextual importante a la aplicación en el arranque, y también permiten al SLEE el proporcionar una persistencia transparente de los campos del SBB.

- **org.(…).WakeUpSbbActivityContextInterface:** Es la interfaz de contexto de actividad que será usada para las acciones del temporizador.
- **La lista de eventos que el SBB suscribe:**
  - *javax.sip.message.Request.MESSAGE:* Se trata del evento SIP MESSAGE. Destacar que este evento está marcado como evento inicial (initial-event). Esto significa que los eventos MESSAGE recibidos por el servicio *Wake Up Call* deben originar una nueva instancia del SBB raíz para ser creada con su propio contexto de actividad asociado con dicha instancia.
  - *javax.slee.facilities.TimerEvent:* Este es el evento SLEE Timer, el cual originará el mensaje de respuesta a la petición del cliente.
- **jain-sip:** Es la referencia al Adaptador de Recurso SIP. El SBB utilizará el SIP RA para interactuar con el cliente.

Finalmente, se detallará el SBB propiamente dicho. Implementado en el fichero *WakeUpSbb.java*, goza de unos cuantos métodos, pero tan sólo se resaltarán aquéllos que son propios del servicio y que con más probabilidad serán diferentes a los de otros servicios orientados a SIP.

```
public void onMessageEvent(javax.sip.RequestEvent event, ActivityContextInterface aci)
```

Este método es llamado cuando una petición MESSAGE es recibida por el SLEE. Lo primero que hace este método es mandar un acuse de recibo al cliente. Lo que hace es utilizar el contexto de actividad, creado por el servidor durante la interacción con el cliente, para llevar la respuesta.

```
// Notifiy the client that we received the SIP MESSAGE request
ServerTransaction st = (ServerTransaction) aci.getActivity();
Response response = messageFactory.createResponse(Response.OK, request);
st.sendResponse(response);
```

A continuación, se crea una señal y se asocia con la instancia del SBB. Esta señal será la que se pase al temporizador cuando sea programado para la futura intervención del método *TimerEvent()*.

```
//   CREATE A NEW NULL ACTIVITIY
NullActivity timerBus = this.nullActivityFactory.createNullActivity();

// ATTACH ITSELF TO THE NULL ACTIVITY
// BY USING THE ACTIVITY CONTEXT INTERFACE
ActivityContextInterface timerBusACI =
       this.nullACIFactory.getActivityContextInterface(timerBus);
timerBusACI.attach(sbbContext.getSbbLocalObject());
```

Lo siguiente será analizar sintácticamente el mensaje entrante. Se extraerá el tiempo de retardo y el cuerpo del texto:

```
        // PARSING THE MESSAGE BODY
        String body = new String(request.getRawContent());
        int i = body.indexOf(" ");
        String timerValue = body.substring(0,i);
        int timer = Integer.parseInt(timerValue);
        String bodyMessage = body.substring(i+1);
```

Para conseguir un objeto que implemente la Interfaz de Contexto de Actividad del SBB y así poder hacer uso de sus métodos, se hará lo siguiente:

```
        // SETTING VALUES ON THE ACTIVITY CONTEXT
        // USING THE SBB CUSTOM ACI
        WakeUpSbbActivityContextInterface myViewOfTimerBusACI =
                this.asSbbActivityContextInterface(timerBusACI);
```

A continuación, se hace uso del *WakeUpSbbActivityContextInterface* y se almacena el cuerpo del mensaje en la señal (*NullActivity*) creada antes para este propósito.

```
        myViewOfTimerBusACI.setBody(bodyMessage);
```

Lo siguiente será buscar la dirección de contacto del cliente donde el servicio *Wake Up Call* tiene que enviar la respuesta:

```
// The From field of each SIP MESSAGE has the UA Address of Record (logical address),
// which can be mapped to a current physical contact address. The mapping is provided by
// the LocationService, which works together with the SIP Registrar service.

        FromHeader fromHeader = (FromHeader)request.getHeader(FromHeader.NAME);
        Address fromAddress = fromHeader.getAddress();
        URI contactURI = findLocalTarget(fromHeader.getAddress().getURI());
        Address contactAddress = addressFactory.createAddress(contactURI);
        ContactHeader contactHeader = headerFactory.createContactHeader(contactAddress);
```

Al igual que antes, pero ahora con la dirección de contacto, se almacenará la dirección de destino en la señal de actividad.

```
        myViewOfTimerBusACI.setContact(contactHeader);
```

Por último, se crea el temporizador y se programa usando la facilidad de temporizador del SLEE.

```
        // SETTING THE TIMER BY USING THE VALUE
        // IN THE SIP MESSAGE BODY
        TimerOptions options = new TimerOptions();
        options.setPersistent(true);
        this.timerFacility.setTimer(timerBusACI,
                null,
                System.currentTimeMillis()+timer*1000,
                options);
```

```
public void onTimerEvent(TimerEvent event, ActivityContextInterface aci)
```

Cuando se lanza la alarma del temporizador, que ha sido programada por el método *onMessageEvent()*, la facilidad de temporizador del SLEE genera un evento

temporizador. Todos los SBBs ligados a la señal (*NullActivity*), que estaba asociada con el temporizador, recibirán el evento.

Este método (*onTimerEvent*) será invocado, ya que así lo especifica el *WakeUp-sbb-jar.xml*.

Primero se extrae el *contacto* y el *cuerpo* del *NullActivity*.

```
// RETRIEVING STORED VALUE FROM THE ACTIVITY CONTEXT INTERFACE
WakeUpSbbActivityContextInterface myViewOfACI =
        this.asSbbActivityContextInterface(aci);
Header contact = myViewOfACI.getContact();
String body = myViewOfACI.getBody();
```

Finalmente, se pasan los valores recogidos al método *sendWakeUpCall*.

```
// SENDING BACK THE WAKE UP CALL
sendWakeUpCall(contact, body);
```

```
private void sendWakeUpCall(Header toContact, String body)
```

Este método crea un nuevo SIP MESSAGE, dirigido a *toContact* y con el texto (*body*) para enviar.

Es interesante mencionar, que para enviar el mensaje al cliente, tal y como es pedido en el *WakeUp-sbb-jar.xml*, este método usa el proveedor de Adaptador de Recurso SIP.

El proveedor de SIP es referenciado primero en el momento que el contenedor del SLEE establece el contexto del SBB

```
// Getting JAIN SIP Resource Adaptor interfaces
fp = (SipFactoryProvider) myEnv.lookup("slee/resources/jainsip/1.1/provider");

provider = fp.getSipProvider();
```

La variable *provider* será más tarde usada por el método, para establecer una transacción SIP con el cliente y enviarle el mensaje.

```
ClientTransaction ct = provider.getNewClientTransaction(req);
ct.sendRequest();
```

4.6.1.2. Despliegue

Una vez comprobados los requisitos de la plataforma Mobicents (9.2) e instalada correctamente (Anexo A.2), se podrá efectuar el despliegue y la posterior utilización del servicio *Wake Up Call* siguiendo estos pasos:

1. Descargar el servicio mediante CVS. El procedimiento a seguir será el mismo que el seguido en el Anexo A.2, con la salvedad de que ahora en el *checkout* lo que se escribirá en el campo referente al módulo a descargar será *mobicents-examples/sip-wake-up*.
2. Desplegar el servicio (sin tener arrancado el SLEE) mediante el uso de la herramienta Ant (*C:\PFC\mobicents-examples\sip-wake-up>ant all*).
3. Descargar el cliente MSN Messenger para el envío de mensajes instantáneos mediante el uso del protocolo SIP. Se utilizará la versión 4.6 [41], ya que las últimas versiones requieren el establecimiento de una sesión SIP entre los equipos finales, y el código actual no soporta eso.
4. Instalación y configuración el cliente SIP. Mirar Figura 4.56.
5. Arrancar el SLEE (*C:\PFC\jboss-3.2.6>bin\run.bat -c all -b p4*).
6. Iniciar la sesión en el cliente MSN.
7. Enviar una petición a la dirección *wakeup@nist.gov*. Se envía concretamente el mensaje *"10 FIN DEL PROYECTO!"* (sin las comillas) y transcurridos 10 segundos se recibirá el mensaje *"FIN DEL PROYECTO!"* (Figura 4.57).

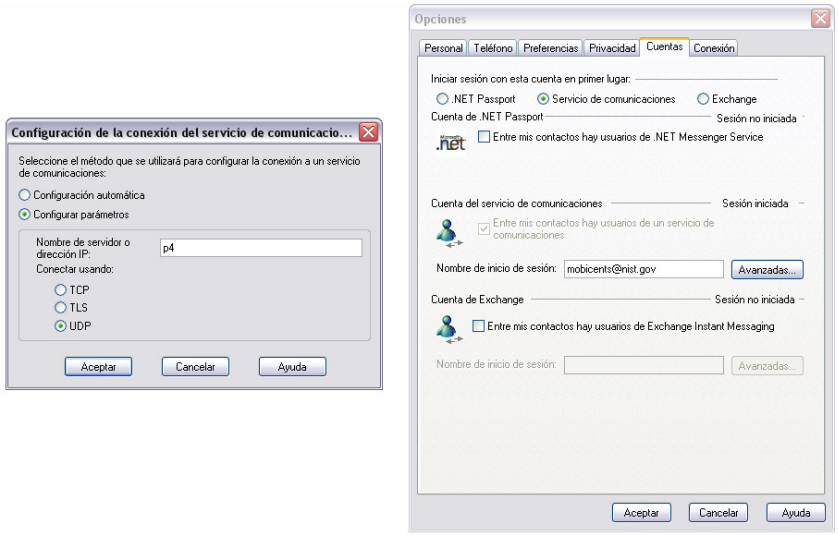


Figura 4.56: Configuración del cliente MSN.

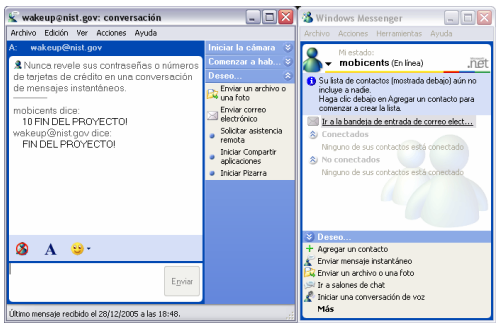


Figura 4.57: Utilización del servicio *Wake Up Call*.

4.6.2. Corrección de los constructores

Como se ha mencionado con anterioridad, ya se disponía, dentro del proyecto Mobicents, de los permisos para, por ejemplo, hacer las modificaciones que se estimasen oportunas del código existente.

Se pudo comprobar que tanto para llevar a cabo la compilación del SLEE, como para efectuar el despliegue de los servicios *proxy* y *registrar*, se producían errores motivados por la ausencia de una generalidad completa, es decir, que había que editar el fichero constructor y cambiar ciertas rutas para que concordasen con la estructura de directorios. Por lo tanto, se decidió corregir este problema para que cualquiera que quisiese instalar la plataforma Mobicents tuviese la comodidad de hacerlo con la simple ejecución del comando *ant*, teniendo tan sólo que preocuparse de crear las correspondientes variables de entorno vistas en Anexo A.2.

Los dos ficheros modificados fueron los siguientes: *C:\PFC\mobicents\build.xml* y *C:\PFC\mobicents\ra\sipra\build.xml*. Antes de llevar a cabo la actualización en el proyecto Mobicents, se dio parte a la comunidad de desarrolladores para conocer su opinión al respecto. La respuesta fue satisfactoria, por lo que ya solamente se tenía que efectuar la contribución de los cambios.

Lo primero es abrir el cliente CVS y seleccionar *Admin* → *Login...*. Una vez introducido el *login* y el *password*, sólo quedará señalar en un primer momento el fichero *C:\PFC\mobicents\build.xml*, para seguidamente seleccionar *Modify* → *Commit…*, introducir el texto que se considere oportuno (*"Some lines modified to avoid errors during the deployment"*), y pulsar *Aceptar*. En ese mismo instante el CVS del proyecto Mobicents se actualizará, y esta nueva versión del fichero *build.xml* será accesible a todos aquellos usuarios unidos al proyecto (Figura 4.58 y Figura 4.59).

Finalmente, se señalará el fichero *C:\PFC\mobicents\ra\sipra\build.xml* y se repetirá la acción anterior (Figura 4.60 y Figura 4.61), tan sólo que ahora el texto introducido será diferente *("One line added to avoid errors during the old deployment without using the auto-deploy-sip target"*).

Estas aportaciones también tuvieron una buena acogida dentro de la comunidad de desarrolladores y usuarios, y se espera poder seguir contribuyendo a la mejora del proyecto en el futuro.

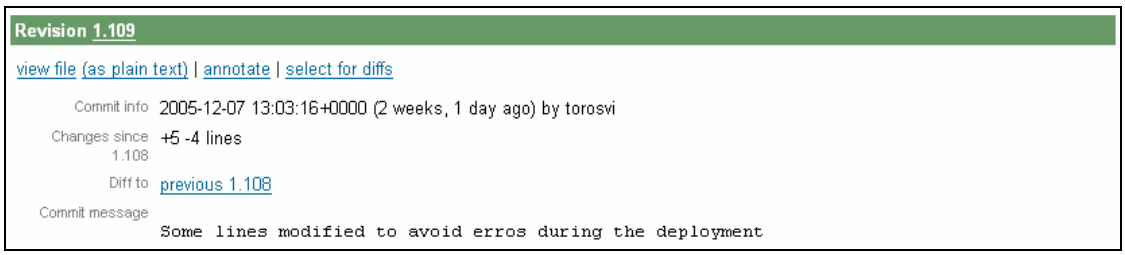
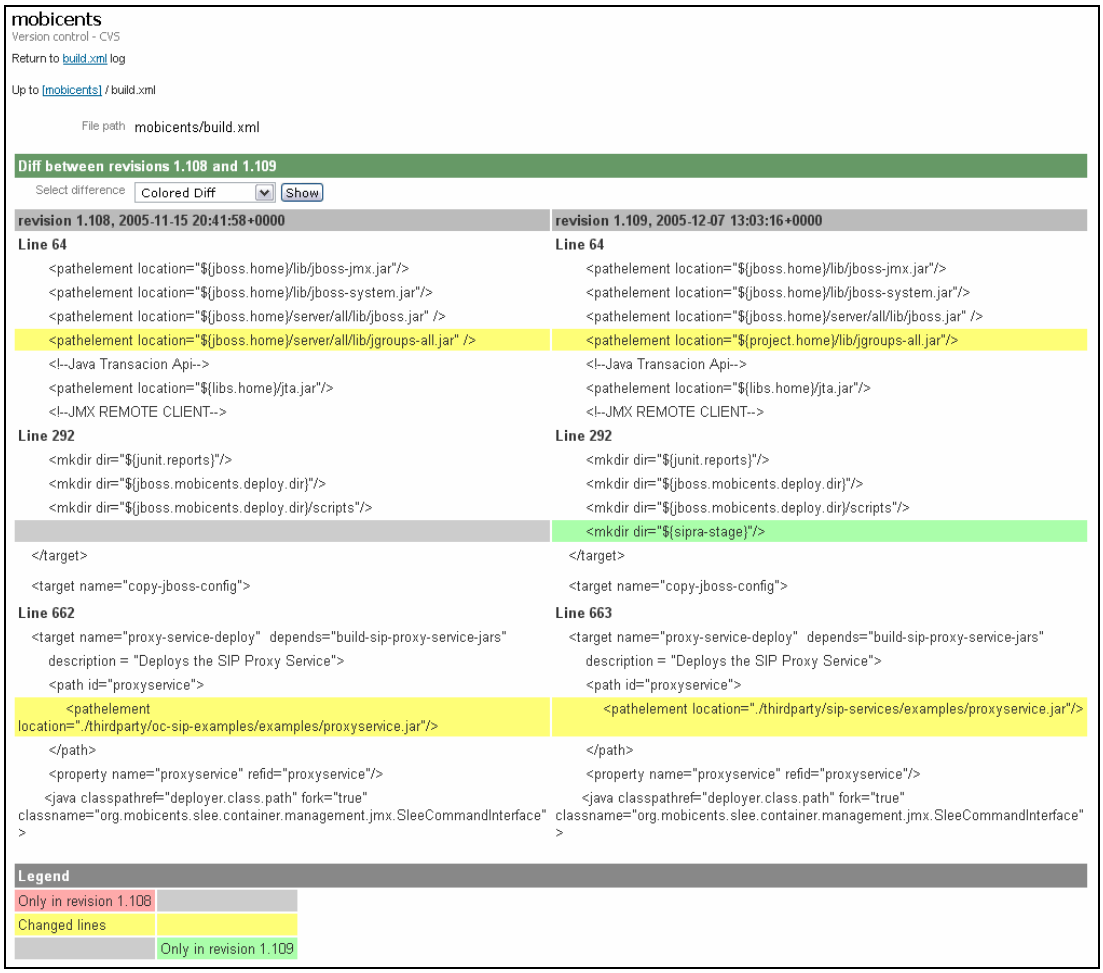


Figura 4.58: Nueva versión del fichero *build.xml* de Mobicents.



mobicents
Version control - CVS
Return to build.xml log
Up to [mobicents] / build.xml
File path mobicents/build.xml

Diff between revisions 1.108 and 1.109

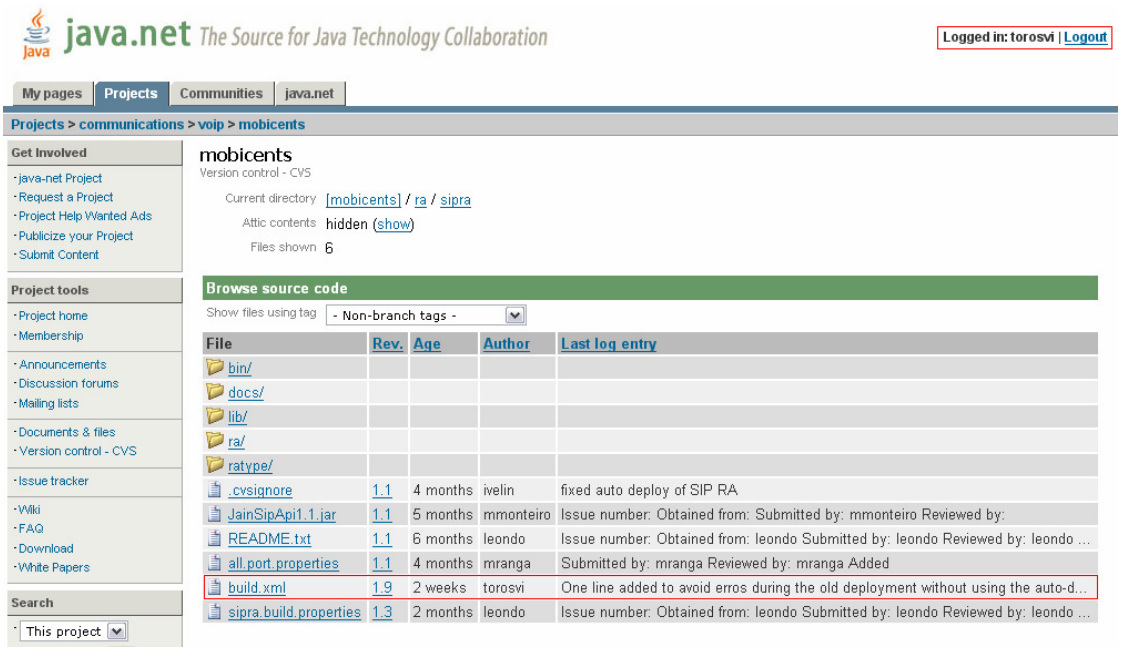
Select difference: Colored Diff — Show

revision 1.108, 2005-11-15 20:41:58+0000	revision 1.109, 2005-12-07 13:03:16+0000
Line 64	**Line 64**
<pathelement location="${jboss.home}/lib/jboss-jmx.jar"/>	<pathelement location="${jboss.home}/lib/jboss-jmx.jar"/>
<pathelement location="${jboss.home}/lib/jboss-system.jar"/>	<pathelement location="${jboss.home}/lib/jboss-system.jar"/>
<pathelement location="${jboss.home}/server/all/lib/jboss.jar" />	<pathelement location="${jboss.home}/server/all/lib/jboss.jar" />
<pathelement location="${jboss.home}/server/all/lib/jgroups-all.jar" />	<pathelement location="${project.home}/lib/jgroups-all.jar"/>
<!--Java Transacion Api-->	<!--Java Transacion Api-->
<pathelement location="${libs.home}/jta.jar"/>	<pathelement location="${libs.home}/jta.jar"/>
<!--JMX REMOTE CLIENT-->	<!--JMX REMOTE CLIENT-->
Line 292	**Line 292**
<mkdir dir="${junit.reports}"/>	<mkdir dir="${junit.reports}"/>
<mkdir dir="${jboss.mobicents.deploy.dir}"/>	<mkdir dir="${jboss.mobicents.deploy.dir}"/>
<mkdir dir="${jboss.mobicents.deploy.dir}/scripts"/>	<mkdir dir="${jboss.mobicents.deploy.dir}/scripts"/>
	<mkdir dir="${sipra-stage}"/>
</target>	</target>
<target name="copy-jboss-config">	<target name="copy-jboss-config">
Line 662	**Line 663**
<target name="proxy-service-deploy" depends="build-sip-proxy-service-jars"	<target name="proxy-service-deploy" depends="build-sip-proxy-service-jars"
description = "Deploys the SIP Proxy Service">	description = "Deploys the SIP Proxy Service">
<path id="proxyservice">	<path id="proxyservice">
<pathelement location="./thirdparty/oc-sip-examples/examples/proxyservice.jar"/>	<pathelement location="./thirdparty/sip-services/examples/proxyservice.jar"/>
</path>	</path>
<property name="proxyservice" refid="proxyservice"/>	<property name="proxyservice" refid="proxyservice"/>
<java classpathref="deployer.class.path" fork="true" classname="org.mobicents.slee.container.management.jmx.SleeCommandInterface" >	<java classpathref="deployer.class.path" fork="true" classname="org.mobicents.slee.container.management.jmx.SleeCommandInterface" >

Legend

Only in revision 1.108	
Changed lines	
	Only in revision 1.109

Figura 4.59: Diferencias con la versión anterior del fichero *build.xml* de Mobicents.



java.net The Source for Java Technology Collaboration
Logged in: torosvi | Logout

My pages | Projects | Communities | java.net

Projects > communications > voip > mobicents

Get Involved
- java-net Project
- Request a Project
- Project Help Wanted Ads
- Publicize your Project
- Submit Content

Project tools
- Project home
- Membership
- Announcements
- Discussion forums
- Mailing lists
- Documents & files
- Version control - CVS
- Issue tracker
- Wiki
- FAQ
- Download
- White Papers

Search: This project

mobicents
Version control - CVS

Current directory [mobicents] / ra / sipra
Attic contents hidden (show)
Files shown 6

Browse source code

Show files using tag: - Non-branch tags -

File	Rev.	Age	Author	Last log entry
bin/				
docs/				
lib/				
ra/				
ratype/				
.cvsignore	1.1	4 months	ivelin	fixed auto deploy of SIP RA
JainSipApi1.1.jar	1.1	5 months	mmonteiro	Issue number: Obtained from: Submitted by: mmonteiro Reviewed by:
README.txt	1.1	6 months	leondo	Issue number: Obtained from: leondo Submitted by: leondo Reviewed by: leondo ...
all.port.properties	1.1	4 months	mranga	Submitted by: mranga Reviewed by: mranga Added
build.xml	1.9	2 weeks	torosvi	One line added to avoid erros during the old deployment without using the auto-d...
sipra.build.properties	1.3	2 months	leondo	Issue number: Obtained from: leondo Submitted by: leondo Reviewed by: leondo ...

Figura 4.60: Nueva versión del fichero *build.xml* del SIP RA.

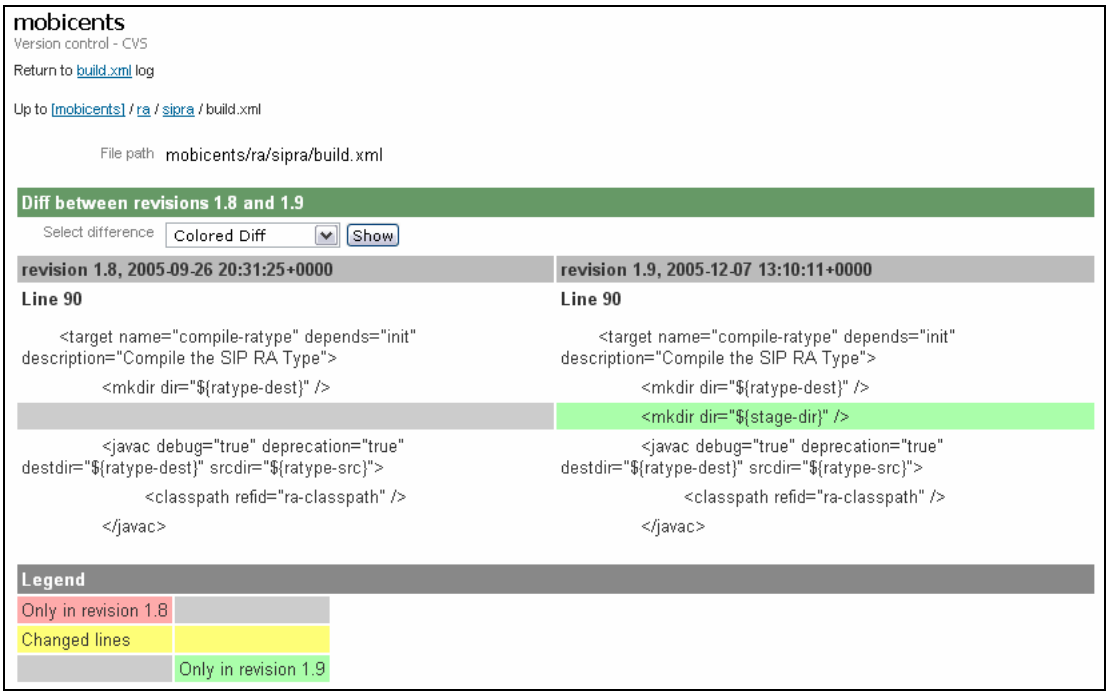


Figura 4.61: Diferencias con la versión anterior del fichero *build.xml* del SIP RA.

4.7. Nueva aplicación

Con la llegada a este punto, los tres objetivos del proyecto ya se han cumplido. Lo que se hará a continuación es comentar una de las aplicaciones que se están desarrollando en el proyecto Mobicents y que se podría decir que es la versión mejorada del servicio *Wake Up Call* desplegado.

Se trata del *Google Talk Bot*, el cual consiste básicamente en un SBB que hace uso del Adaptador de Recurso XMPP. El Protocolo de Presencia y Mensajería Extensible (XMPP) [23] es un protocolo abierto basado en XML (*Extensible Messaging and Presence Protocol - Protocolo de Presencia y Mensajería Extensible*) que se utiliza para servicios de mensajería, presencia y petición-respuesta en tiempo real. Para ser más exactos, cercano al tiempo real.

El *Bot* simplemente recibe mensajes instantáneos procedentes de usuarios del *Google Talk* [43] y los envía de vuelta indicando el número de caracteres del mensaje. Además, si se le envía el mensaje *"time"*, lo que hace es responder con el mensaje *"it is…Party time! :)"*.

Para poder hacer uso de esta aplicación, habrá que disponer de dos cuentas de correo en *gmail* y del cliente *Google Talk*.

Previo al despliegue de esta aplicación, se tendrá que configurar el fichero *xmppra.properties*, correspondiente al Adaptador de Recurso XMPP, atendiendo a los datos de una de las cuentas *gmail* (Figura 4.62).

```
XMPPIP=talk.google.com
XMPPPort=5222
XMPPPassword=********

# If you are setting a GoogleTalk user id, do not append @gmail.com to it. Only set the name before @.
XMPPUser=victorosvillegas

# XMPPMode is one of client, component, googletalk
XMPPMode=googletalk
responseTimeout=300000
```

Figura 4.62: Fichero *xmppra.properties*.

Una vez desplegado el Adaptador de Recurso XMPP, ya se podrá hacer lo propio con la aplicación antes de arrancar el SLEE. Con la otra cuenta *gmail* se iniciará una sesión con el *Google Talk*, y después de invitar al otro usuario, el cual será precisamente el *Google Talk Bot*, se obtiene lo siguiente:

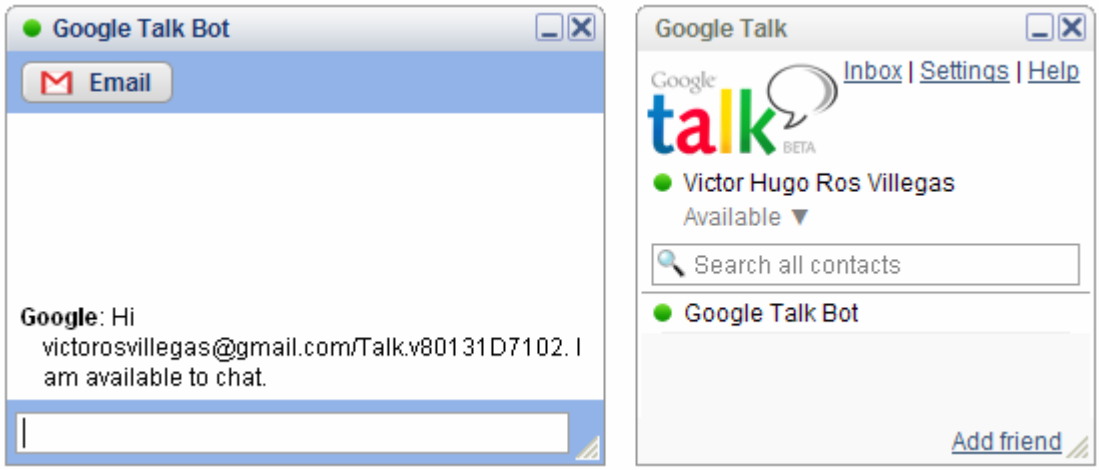


Figura 4.63: Aplicación *Google Talk Bot*.

El resultado obtenido al chatear con el *Bot* se muestra en la Figura 4.64.

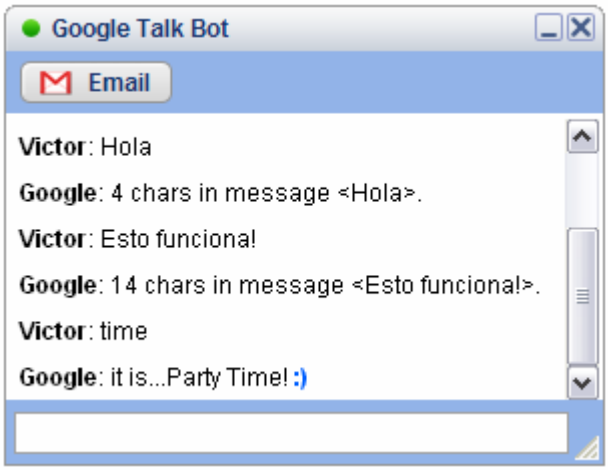


Figura 4.64: Chateando con el *Bot*.

Capítulo 5
Líneas futuras y conclusiones

5. Líneas futuras y conclusiones

5.1. Líneas futuras

Dentro de lo que es el propio proyecto Mobicents, tienen marcadas una serie de tareas [42], cada una con un grado de dificultad, a cumplir a corto, medio o largo plazo, dependiendo del tipo de tarea.

Respecto a la incorporación de nuevos ejemplos y aplicaciones al proyecto Mobicents, se puede hacer mención a dos aplicaciones disponibles desde los meses de Noviembre y Diciembre de 2005 y que resultan ser bastante interesantes.

Una de ellas es el *Google Talk Bot*, del cual ya se habló en el capítulo 4, apartado 4.7. Esta aplicación puede ser usada como punto de partida para el desarrollo de servicios más sofisticados que ofrezcan una alta funcionalidad. Algunos de ellos serían los siguientes:

1. Servicio similar al *Wake Up Call*. Se trataría de que un usuario enviase una petición indicando una hora a la que quiere ser avisado. El servicio recogerá la petición y se encargará de avisar a dicho usuario a la hora indicada.
2. Alerta para los juegos online. Permitiría la entrada en una sala correspondiente a un juego y se podría pedir el ser avisados cuando se incorpore el número de jugadores requerido.
3. Las típicas notificaciones de calendario.
4. Aviso de últimas ofertas de vuelos. Es poco útil para los usuarios el tener que descargar e instalar una aplicación distinta para cada compañía aérea con la que estén dispuestos a volar. Así que el poder simplemente incorporar un contacto para cada compañía aérea, en una lista del *Google Talk*, es una opción más realista.

La otra aplicación de interés es el *Slee Graph Viewer*, el cual se trata de un visor del SLEE que podría llegar a convertirse en un centro de control online.

Mediante el uso de Swing (biblioteca gráfica de Java) [44], se consigue visualizar los componentes propios del SLEE (Servicios, Bloques de Construcción de Servicios y Adaptadores de Recurso) y las relaciones entre ellos.

En la Figura 5.1 se muestra un ejemplo de este visor arrancado en el servidor Mobicents. Se puede observar como el servicio *proxy* está compuesto de dos SBBs. El *Proxy SBB* es precisamente el SBB raíz, y es el que acepta los eventos SIP enviados por el SIP RA, y bien los maneja directamente o delega ese trabajo (evento REGISTER) al *Registrar SBB*. Además, como se puede observar en el gráfico, ambos SBBs utilizan el SIP RA para enviar señales de protocolo.

Comentar como tanto el servicio *Wake Up Call* como el *Google Talk Bot* están compuestos de un único SBB, que será precisamente el SBB raíz de cada uno. La diferencia está en que el Wake Up SBB hace también uso del SIP RA, mientras que el Talk Bot SBB utiliza, como ya se dijo en su momento, el XMPP RA.

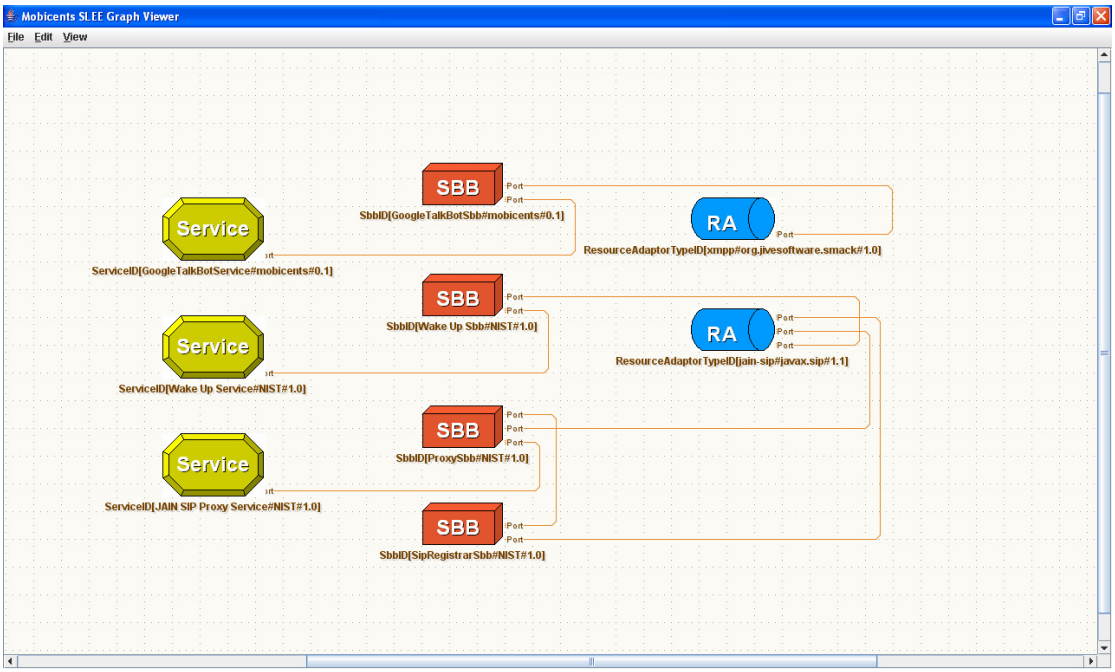


Figura 5.1: Aplicación *Slee Graph Viewer*.

Los posibles avances en la aplicación *Slee Graph Viewer* son más que interesantes. Entre estas posibles mejoras se pueden destacar las siguientes:

1. Convertir la aplicación en algo más que un visor del SLEE, es decir, que también se pueda gestionar y controlar el SLEE.
2. La aplicación debería incluir los tipos de eventos (event types) y su relación directa con los demás componentes.
3. Indicar también en el gráfico el número de eventos enviados entre los nodos.
4. Tener la posibilidad de actualizar el gráfico, bien de forma automática cada 5 ó 10 segundos, o bien tener esa opción en la propia ventana Swing.
5. Añadir la posibilidad de mostrar las estadísticas correspondientes a un componente cuando sea seleccionado.

Una vez vistas las líneas futuras a seguir desde el punto de vista de estas dos aplicaciones concretas desarrolladas en la plataforma Mobicents, se pasa a comentar aspectos más generales y que podrían servir también para futuros Proyectos Fin de Carrera.

Durante el desarrollo de la memoria ya se ha hecho mención a la posibilidad de integrar la tecnología OSA/Parlay con la tecnología JSLEE. Para ello Mobicents está llevando a cabo el proyecto *mobicents-parlay-ra*.

Se tiene que tanto JSLEE como OSA/Parlay son tecnologías complementarias. Por un lado JAIN SLEE proporciona un entorno de aplicación para las aplicaciones OSA/Parlay, mientras que OSA/Parlay no define dicho entorno. Por otro lado JAIN SLEE no define conectividad de red, mientras que OSA/Parlay sí lo hace. Por tanto, además de no ser tecnologías que compitan entre ellas, resulta que cada una mejora a la otra.

Tanto el OSA/Parlay Gateway como los SCFs (*Service Capability Features - Características de Capacidades de Servicio*) vistos desde la perspectiva JAIN SLEE son recursos. El OSA/Parlay Gateway está compuesto de varios SCSs (*Service Capability Servers - Servidores de Capacidades de Servicio*), que son entidades funcionales que proporcionan interfaces OSA/Parlay para las aplicaciones. Cada SCS se ve desde las aplicaciones como una o más SCFs, que son una abstracción de la funcionalidad ofrecida por la red accesible mediante el API de OSA/Parlay (a los SCFs algunas veces se les llama servicios).

La idea consiste en que JAIN SLEE tiene Adaptadores de Recurso instalados que presentan una API a la aplicación que se está ejecutando en el SLEE. Esta API se comunica a través de un protocolo con la implementación del Gateway y del SCF, y finalmente la implementación de OSA/Parlay proporciona la integración de red.

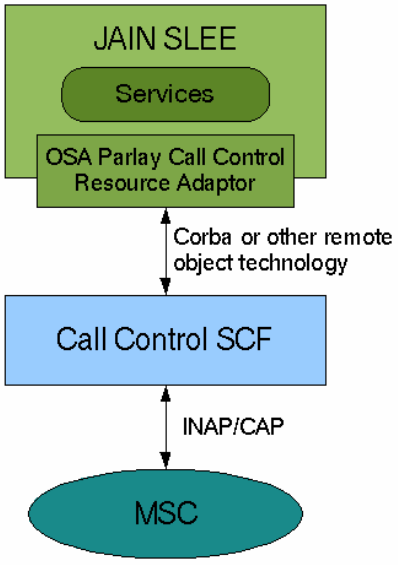


Figura 5.2: JAIN SLEE y OSA/Parlay.

La aplicación es abstraída de los detalles de los protocolos de red mediante el uso de las APIs de alto nivel de abstracción sacadas de las especificaciones de Parlay UML (*Unified Modelling Language - Lenguaje Unificado de Modelado*).

Finalmente Aepona y Mobicents se han unido para poner en marcha el Adaptador de Recurso Parlay. El objetivo de este RA es habilitar el despliegue de aplicaciones Parlay/Parlay-X en contenedores SLEE como es el caso de Mobicents.

La primera meta del proyecto *mobicents-parlay-ra* es proporcionar un Adaptador de Recurso que obedezca al SLEE, proveyendo de una conectividad de primera línea empresarial a las redes de telecomunicaciones a través de los servicios OSA/Parlay. Otra meta adicional es el estandarizar el modelo de programación para el acceso y el uso de los servicios Parlay en un entorno SLEE.

La primera etapa del proyecto será la definición de una API y un RA Type adecuado. Una vez completada esa etapa, lo que se hará será el elegir un servicio Parlay e implementar el RA, y a partir de ahí comenzar con el desarrollo de nuevos servicios.

Por supuesto, otra futura línea de investigación puede ser el estudio de la nueva versión de la especificación JSLEE (JSR 240 [18]: JAIN SLEE v1.1), la cual está dirigida a rellenar los vacíos dejados por la versión 1.0, centrándose en especificar la API de la Arquitectura del Adaptador de Recurso. Una vez terminada dicha especificación, comenzará el trabajo de implementación.

Otra posibilidad de innovación es la que recientemente (30 de Diciembre de 2005) planteó el fundador del proyecto Mobicents y a la que ha invitado a participar tras la conclusión de este proyecto. Se trata de una tarea de portabilidad. La idea es hacer Mobicents utilizable en una variedad de plataformas J2EE además de JBoss. Para poder alcanzar esta meta hay que plantearse la utilización de un entorno de aplicación como puede ser Spring [45], que es una herramienta extendida para abstraer las dependencias de los contenedores J2EE. Por tanto, el poder ser capaz de arrancar Mobicents junto a otros servidores J2EE importantes como pueden ser WebLogic [46], Glassfish [47] o WebSphere [48] sería un gran progreso.

5.2. Conclusiones

Trabajar en este proyecto ha sido una experiencia muy enriquecedora desde el punto de vista personal. El hecho de haber tenido la posibilidad de estudiar una tecnología tan innovadora, de poner en práctica sus enormes beneficios y de realizar un amplio trabajo de documentación, alberga la esperanza de que este proyecto proporcione una base sólida a todos aquellos que pretendan implicarse en esta tecnología.

El poco tiempo que lleva desarrollándose esta tecnología ha sido sin duda el obstáculo más difícil de superar, pero gracias al trabajo diario y al esfuerzo aplicado, así como la incursión en el proyecto Mobicents, han hecho que finalmente se alcancen los objetivos pedidos y se sienten las bases para una larga lista de futuros Proyectos Fin de Carrera.

Motivo de orgullo es la valoración recibida por parte de la comunidad de desarrolladores y usuarios del proyecto Mobicents a las aportaciones realizadas, hechas no sólo con vistas al cumplimiento del tercer y último objetivo del proyecto, sino también como satisfacción personal, poniendo al servicio de los demás miembros de la comunidad Mobicents los conocimientos adquiridos. Prueba de esta importante valoración ha sido, no sólo la felicitación por parte del fundador de Mobicents (Ivelin Ivanov), sino también su propuesta de continuar dentro de su línea de investigación.

Como se ha podido comprobar a lo largo de todo el proyecto, las posibilidades que alberga la tecnología JAIN SLEE son infinitas. Hay muchas aplicaciones interesantes que pueden ser implementadas cuando movilidad, localización, voz, video, y web se mezclan unas con otras. Así que se espera que esta memoria sirva para poder iniciar los nuevos retos que se plantean con una base sólida en la que apoyarse.

Capítulo 6
Fases del proyecto

6. Fases del proyecto

6.1. Introducción

En este capítulo se explicará la evolución del proyecto a lo largo del tiempo, haciendo uso al final de un diagrama de Gantt que muestre de forma gráfica la planificación del proyecto.

6.2. Evolución del proyecto

La primera toma de contacto con el profesorado para la realización del proyecto se llevó a cabo durante el curso académico 2004-2005. Sin embargo, la elaboración real del mismo ha tenido lugar entre los meses de Julio de 2005 y Enero de 2006 (ambos inclusive).

Los dos primeros meses fueron dedicados en exclusividad a la tarea de documentación de todo aquello que tuviese que ver con la tecnología JSLEE. Debido a las características de este proyecto y a la novedad del mismo, la tarea de documentación ha sido una constante a lo largo de todo el desarrollo del proyecto, teniendo que ir en algunos momentos prácticamente al mismo ritmo del avance de la tecnología.

En el mes de Septiembre fue la primera toma de contacto con Rhino de Open Cloud. La primera fase de documentación de esta plataforma se realizó en Dublín, donde se estuvo desde la semana 10 a la 13 (ambas inclusive) con una beca concedida para el perfeccionamiento del inglés. Al regreso de Dublín, se continuó con dicha documentación, pero al mismo tiempo complementándola con el desarrollo práctico de los objetivos pedidos.

Hasta mediados de Noviembre no fue cuando se empezó a involucrarse seriamente en el proyecto Mobicents y desde entonces se ha tratado de ir en línea con dicho proyecto, tanto en documentación como en realización de aportaciones.

Es en el mes de Diciembre, una vez que los objetivos del proyecto estaban prácticamente cumplidos, cuando se comienza con la redacción de la memoria, la cual ha llevado prácticamente mes y medio, ya que se ha compaginado con el seguimiento del proyecto Mobicents, en el cual se estába involucrado y en el que seguirá formando parte si finalmente llegan a buen puerto las conversaciones que el fundador de Mobicents ha propuesto mantener tras la conclusión de este proyecto.

Finalmente, se dedicaron un par de semanas a la preparación de la presentación y se dió por concluido el proyecto.

6.3. Diagrama de Gantt

En este punto tan sólo se representará de forma gráfica (Figura 6.1) la planificación temporal explicada en el punto anterior (6.2).

Figura 6.1: Diagrama de Gantt.

Tan sólo comentar que el hecho de que las tareas de documentación de JSLEE y de Mobicents, así como la parte de desarrollo en Mobicents hayan continuado tras la conclusión de la redacción de la memoria ha sido motivado por la pertenencia al equipo de la comunidad Mobicents como miembros activos del mismo.

Bibliografía

7. Bibliografía

Tanto las características del proyecto como su carácter innovador han hecho que la documentación utilizada haya sido prácticamente en su totalidad extraída de Internet. En la mayoría de los casos por obligación, dada la inexistencia o inaccesibilidad a otros formatos como pueden ser libros o revistas especializadas en la tecnología JAIN SLEE, pero en otros casos por la comodidad que supone el poder acceder a manuales o a foros en los cuales resolver dudas de una manera rápida y eficaz.

En primer lugar se mostrará una lista correspondiente a distintos sitios web a los que se hace referencia a lo largo de la memoria y que son de obligada consulta para el desarrollo del proyecto.

[1] Grupo de desarrollo de tecnología Java
http://java.net/

[2] jNetX Incorporated
http://www.jnetx.com/

[3] Sun Microsystems Incorporated
http://sun.com

[4] Compañía Open Cloud
http://www.opencloud.com

[5] Grupo Mobicents
http://www.mobicents.org

[6] Grupo JBoss
http://www.jboss.com

[7] Página oficial de IBM
http://www.ibm.com

[8] Página oficial de Websphere
http://www.websphere.org/

[9] Página oficial de Oracle
http://www.oracle.com/

[10] Especificación de JMX
www.jcp.org/jsr/detail/3.jsp

[11] Referencia a la tecnología JavaBeans
http://java.sun.com/products/javabeans/

[12] Protocolo SIP
http://www.faqs.org/rfcs/rfc3261.html

[13] Iniciativa JAIN
http://java.sun.com/products/jain/

[14] Miembros de la comunidad JAIN
http://java.sun.com/products/jain/members.html

[15] Especificaciones de las APIs de JAIN
http://java.sun.com/products/jain/api_specs.html

[16] Especificación de JAIN SIP
http://jcp.org/en/jsr/detail?id=32

[17] Especificación de JAIN SLEE 1.0
http://www.jcp.org/en/jsr/detail?id=22

[18] Especificación de JAIN SLEE 1.1
http://www.jcp.org/en/jsr/detail?id=240

[19] Documentación sobre EJB
http://es.wikipedia.org/wiki/EJB

[20] Tecnología JAIN SLEE
http://jainslee.org/

[21] Artículo de Michelle de Lussanet
http://www.forrester.com/go?docid=15658

[22] Proceso de Comunidad Java
http://www.jcp.org

[23] Especificación de XMPP
http://www.xmpp.org/specs

[24] Aplicación de código abierto de una centralita telefónica
http://www.asterisk.org/

[25] Arquitectura del Conector J2EE
http://java.sun.com/j2ee/connector/

[26] Fedora Core
http://fedora.redhat.com

[27] Página ofical de PostgreSQL
http://www.postgresql.org/

[28] Proyecto Apache Ant
http://ant.apache.org

[29] Cliente SJphone
http://www.sjlabs.com/sjp.html

[30] Cliente Linphone
http://www.linphone.org

[31] Descarga del pgAdmin III
http://www.pgadmin.org/download.php

[32] Analizador de paquetes
http://www.ethereal.com/

[33] Compañía Aepona
http://www.aepona.com/

[34] Edición Estándar de la Plataforma Java 2
http://java.sun.com/j2se

[35] Descarga de JBoss
http://prdownloads.sourceforge.net/mobicents/jboss-3.2.6-mobicents-profile.zip?download

[36] Cliente CVS (WinCvs)
http://www.wincvs.org

[37] Ejemplo de comunicación extremo a extremo en Mobicents
http://wiki.java.net/bin/view/Communications/MobicentsSimplePBXExample

[38] Foro de Mobicents
http://forums.java.net/jive/category.jspa?categoryID=36

[39] Tema del foro de Mobicents (*wake up call example*)
http://forums.java.net/jive/thread.jspa?threadID=2026&tstart=30

[40] Notas aportadas para el servicio *Wake Up Call*
http://wiki.java.net/bin/view/Communications/MobicentsExamplesWakeUp#Notes_Victor

[41] Cliente MSN Messenger
http://www.microsoft.com/technet/prodtechnol/exchange/downloads/2000/imclient46.mspx

[42] Tareas del proyecto Mobicents
http://wiki.java.net/bin/view/Communications/MobicentsTODO

[43] Google Talk
http://www.google.com/talk

[44] Biblioteca Gráfica de Java
http://java.sun.com/products/jfc/index.jsp

[45] Interfaz Spring
http://www.springframework.org/

[46] Servidor de aplicaciones BEA WebLogic
http://www.beasys.es/productos/weblogic/weblogic_server.jsp

[47] Comunidad GlassFish
http://java.sun.com/javaee/glassfish/

[48] Comunidad WebSphere
http://www.websphere.org/

En segundo y último lugar se irá mostrando la bibliografía empleada para la elaboración de cada capítulo, por lo que puede que algunas referencias sean repetidas.

Capítulo 1

Artículo: "*Una nueva convergencia: ¿Java en la red?*"
Ángel Cruz, 08/03/2005
http://infoworld.ediworld.com.mx/iw_SpecialReport.asp

Artículo: "*Vodafone Spain Proves Software Portability for Programmable Network Services*"
Vodafone Spain, 05/07/2005
http://www.opencloud.com/opencloud/news-OC-VFES-PR-20050705.html

Presentación: "*Mobicents Open Source VoIP Platform*"
Ivelin Ivanov, 15/12/2005
http://prdownloads.sourceforge.net/mobicents/JavaPolis-2005-Mobicents.pdf?download

Capítulo 2

Lenguaje de programación Java
http://en.wikipedia.org/wiki/Java_programming_language

Página oficial de Sun Microsystems
http://sun.com

Página oficial de Open Cloud
http://www.opencloud.com

Noticias y eventos de Open Cloud
http://www.opencloud.com/opencloud/news-OC-VFES-PR-20050705.html

Grupo Mobicents
http://www.mobicents.org

Grupo JBoss
http://www.jboss.com

Revista de las tecnologías de la información y las comunicaciones
http://telematica.cicese.mx/revistatel/archivos/Telem@tica_AnoII_No24.pdf

Artículo: "*Java Management Extensions (JMX)*"
María Elena Munaro, 22/06/2005
http://www.codejava.org/detalle_nota.htm?idxnota=36997&destacada=1

Documentación sobre JMX
http://es.wikipedia.org/wiki/JMX

Especificación de JMX
www.jcp.org/jsr/detail/3.jsp

Documentación sobre RMI
http://es.wikipedia.org/wiki/RMI

Miembros de la comunidad JAIN
http://java.sun.com/products/jain/members.html

Especificaciones de las APIs de JAIN
http://java.sun.com/products/jain/api_specs.html

Protocolo SIP
http://www.faqs.org/rfcs/rfc3261.html

Especificación de JAIN SIP
http://www.jcp.org/en/jsr/detail?id=32

Especificación de JAIN SLEE 1.0
http://www.jcp.org/en/jsr/detail?id=22

Especificación de JAIN SLEE 1.1
http://www.jcp.org/en/jsr/detail?id=240

Iniciativa JAIN
http://java.sun.com/products/jain

Documento: "*JAIN: A New Approach to Services in Communication Networks*"
John de Keijzer, Douglas Tait, Rob Goedman y Sun Microsystems, Inc.
IEEE Communications Magazine, Enero del 2000
http://www.comsoc.org/ci/private/2000/jan/Dekeijzer.html

Documento: "*JAIN Protocol APIs*"
Ravi Raj Bhat y Rajeev Gupta, Trillium Digital Systems, Inc.
IEEE Communications Magazine, Enero del 2000
http://www.comsoc.org/ci/private/2000/jan/Bhat.html

Documento: "*JAIN™: Integrated Network APIs for the Java™ Platform*"
Sun Microsystems, Inc. Noviembre del 2000
http://java.sun.com/products/jain/WP2000.pdf

Documentación sobre EJB
http://es.wikipedia.org/wiki/EJB

Artículo: "*JAIN/SLEE: EJB for Communications. Opening the telecommunications world for Java*"
Sven Haiges, 07/09/2004
http://java.sys-con.com/read/46230.htm

Portal web sobre JAIN SLEE
http://www.jainslee.org

Capítulo 3

Especificación de JAIN SLEE 1.0
http://www.jcp.org/en/jsr/detail?id=22

Especificación de JAIN SLEE 1.1
http://www.jcp.org/en/jsr/detail?id=240

Artículo de Michelle de Lussanet
http://www.forrester.com/go?docid=15658

Documento: "*A SLEE for all Seasons*"
Open Cloud, 04/03/2003
http://www.opencloud.com/slee/downloads/asfas.pdf

Presentación: "*Overview of the JAIN SLEE Philosophy, Architecture, and Component Model*"
Dr. David Page para Open Cloud
http://jungla.dit.upm.es/~gabriel/seminario/jainslee-introduction.pdf

Presentación: "*Mobicents: The First Certified Open Source Implementation of JAIN SLEE 1.0*"
Ivelin Ivanov, 2005
https://mobicents.dev.java.net/files/documents/1075/16989/Mobicents-JavaOne-05.pdf

Presentación: "*JAIN SLEE Tutorial: Serving the Developer Community*"
Swee Lim, Phelim O'Doherty, David Ferry y David Page, 2003
http://java.sun.com/products/jain/JAIN-SLEE-Tutorial.pdf

Aspectos fundamentales de JAIN SLEE
http://jainslee.org/slee/fundamentals.html

Manual de Rhino: RhinoSDK-1.4.0 admin-manual
http://www.opencloud.com/slee/downloads.html

Grupo Mobicents
http://www.mobicents.org

Capítulo 4

Página oficial de Open Cloud
http://www.opencloud.com

Fedora Core
http://fedora.redhat.com

Manual de Red Hat Linux
http://www.europe.redhat.com/documentation/rhl9/rhl-gsg-es-9/s1-q-and-a-starting.php3

Manual de Rhino: RhinoSDK-1.4.0 admin-manual
Open Cloud, 23/06/2005
http://www.opencloud.com/slee/downloads.html

Documentación sobre PostgreSQL
http://www.postgresql.org/docs/8.0/interactive/index.html

Manual del usuario de PostgreSQL
http://es.tldp.org/Postgresql-es/web/navegable/user/user.html

SJ Labs (software para VoIP)
http://www.sjlabs.com/sjp.html

Linphone (telefonía sobre Linux)
http://www.linphone.org

Herramienta pgAdmin para PostgreSQL
http://www.pgadmin.org

Documentación sobre Iptables
http://en.wikipedia.org/wiki/Iptables

Manual de iptables de Linux (man iptables)

Manual de iptables-save de Linux (man iptables-save)

Documentación sobre Ethereal
http://es.wikipedia.org/wiki/Ethereal

Edición Estándar de la Plataforma Java 2
http://java.sun.com/j2se

Documentación sobre JDBC
http://es.wikipedia.org/wiki/JDBC

Documentación sobre JNDI
http://en.wikipedia.org/wiki/Java_Naming_and_Directory_Interface

Proyecto Apache Ant
http://ant.apache.org

Descargas de Mobicents desde sourceforge.net
https://sourceforge.net/project/showfiles.php?group_id=102670

Artículo: "*Aepona Announces Full Complement of Java Support*"
Aepona, 27/10/2005
http://www.aepona.com/press_zone/mobicents.htm

Grupo Mobicents
http://www.mobicents.org

Cliente CVS (WinCvs)
http://www.wincvs.org

Documentación sobre CVS
http://es.wikipedia.org/wiki/CVS

Ejemplo de comunicación extremo a extremo en Mobicents
http://wiki.java.net/bin/view/Communications/MobicentsSimplePBXExample

Foro de Mobicents
http://forums.java.net/jive/category.jspa?categoryID=36

Tema del foro de Mobicents (*wake up call example*)
http://forums.java.net/jive/thread.jspa?threadID=2026&tstart=30

Documento: "*The Mobicents Open Source SLEE Platform*"
Ivelin Ivanov, M. Ranganathan, F. Moggia y Phelim O'Doherty, 2005
http://prdownloads.sourceforge.net/mobicents/All-VON2005.pdf?download

Servicio *Wake Up Call* en Mobicents
http://wiki.java.net/bin/view/Communications/MobicentsExamplesWakeUp

Cliente MSN Messenger
http://www.microsoft.com/technet/prodtechnol/exchange/downloads/2000/imclient46.mspx

Google Talk
http://www.google.com/talk

Capítulo 5

Ejemplo de visor del SLEE en Mobicents
http://wiki.java.net/bin/view/Communications/MobicentsExamplesSleeGraph

JAIN SLEE y OSA/Parlay
http://jainslee.org/application/application.html

Proyecto del Adaptador de Recurso Parlay en Mobicents
https://mobicents-parlay-ra.dev.java.net/

Presentación: "*Middleware para provisión de servicios móviles*"
Tomás Joaquín Robles Salvachúa
http://www.catedra-amena.etsit.upm.es/descargas/middleware.ppt

Glosario

8. Glosario

A continuación se dará la lista de los distintos acrónimos que aparecen a lo largo de la memoria, así como de aquellas palabras que requieran explicación aparte.

3GPP
3rd Generation Project Partnership - Asociación del Proyecto de 3ª Generación.
Acuerdo de colaboración que fue establecido en Diciembre de 1988. Se trata de la cooperación entre la ETSI (Europa), ARIB/TTC (Japón), CCSA (China), ATIS (Norte América) y TTA (Corea del Sur).

API
Application Programming Interface - Interfaz de Programación de Aplicaciones.
Conjunto de especificaciones de comunicación entre componentes software.

BSS
Base Station Subsystem - Subsistema de Estación Base.

CAP
Common Alerting Protocol - Protocolo de Alerta Común.

CIM
Common Information Model - Modelo de Información Común.
Modelo de datos orientado a objetos que permite describir toda la información de gestión en ambientes de redes.

CMP
Container Managed Persistence - Persistencia Gestionada por Contenedor.
Sirve para proporcionar persistencia a las aplicaciones J2EE.

CORBA
Common Object Request Broker Architecture - Arquitectura de Agente de Petición de Objeto Común.
Estándar que establece una plataforma de desarrollo de sistemas distribuidos facilitando la invocación de métodos remotos bajo un paradigma orientado a objetos.

CRM
Customer Relationship Management - Gestión de las Relaciones con el Cliente.
Recoge el conjunto de aplicaciones informáticas que tratan de poner en relación los datos relacionados con los clientes, las ventas y en general todo lo relacionado con la actividad comercial de las empresas, a fin de explotar todos estos datos de cara a una mejor gestión de los procesos de negocio.

CVS
Concurrent Versions System - Sistema de Control de Versiones.

Utiliza una arquitectura cliente-servidor: un servidor guarda la(s) versión(es) actual(es) del proyecto y su historia, y los clientes conectan al servidor para sacar una copia completa del proyecto, trabajar en esa copia y entonces ingresar sus cambios.

DMS
Dynamic Monitoring Service - Servicio de Monitorización Dinámico.

EAI
Enterprise Application Integration - Integración de Aplicaciones Empresariales.
Es la integración de nuevas aplicaciones con las ya existentes, incluyendo las aplicaciones heredadas o los paquetes de software, de forma que todas juntas proporcionen las funcionalidades necesarias para soportar los procesos de negocio de la empresa. Esta integración permite a la organización mantener el ritmo de los cambios del mercado y reaccionar a tiempo frente a ellos.

EJB
Enterprise JavaBeans - JavaBeans para la Empresa.
Son una de las APIs que forman parte del estándar de construcción de aplicaciones empresariales J2EE de Sun Microsystems.

ETSI
European Telecommunications Standards Institute - Instituto de Estándares de Telecomunicación Europeos.
Organización de estandarización de la industria de las telecomunicaciones (fabricantes de equipos y operadores de redes) de Europa, con proyección mundial. ETSI ha tenido gran éxito al estandarizar el sistema de telefonía móvil GSM.

H.323
Recomendación del ITU-T, que define los protocolos para proveer sesiones de comunicación audiovisual en cualquier paquete de la red.

HLR
Home Location Register - Registro de Localización Local.
Base de datos centralizada de la red que almacena y maneja a todos los suscriptores móviles de un operador en concreto. Actúa como un almacén permanente para la información de suscriptores hasta que la suscripción se cancele.

HTTP
Hyper Text Transfer Protocol - Protocolo de Transferencia de Hipertexto.
Es el protocolo de la web usado en cada transacción. Es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores. Al finalizar la transacción todos los datos se pierden.

IETF
Internet Engineering Task Force - Grupo de Trabajo en Ingeniería de Internet.

IMS
IP Multimedia Subsystem - Subsistema Multimedia para IP.
Representa la implantación conservadora de la arquitectura All-IP en 3G y promueve la convergencia con la Internet multimedia, proporcionando servicios de contenidos y comunicaciones multimedia en tiempo real.

INAP/AIN
Intelligent Network Application Protocol/Advanced Intelligent Network - Protocolo de Aplicación de Red Inteligente/Red Inteligente Avanzada.

IoC
Inversion of Control - Control de Inversión.

IP
Internet Protocol - Protocolo de Internet.
Protocolo no orientado a conexión usado tanto por el origen como por el destino para la comunicación de datos a través de una red de paquetes conmutados.

ISUP
Integrated Services Digital Network User Part - Parte de Usuario de la Red Digital de Servicios Integrados.

ITU-T
Internacional Telecommunication Union Telecommunications Sector - Sector de Normalización de las Telecomunicaciones.

J2EE
Java 2 Enterprise Editions - Edición Empresarial del Paquete Java.
Comprende un conjunto de especificaciones y funcionalidades orientadas al desarrollo de aplicaciones empresariales.

JAIN
Java APIs for the Advanced Intelligent Network - APIs de Java para la Red Inteligente Avanzada.

JAR
Java Archive - Archivo Java.
Mecanismo, basado en Java, de almacenamiento de datos de forma comprimida.

JCA
J2EE Connector Architecture - Arquitectura del Conector J2EE.

JCAT
Java Coordination and Transaction - Transacciones y Coordinación Java.

JCC
Java Call Control - Control de Llamada Java.

JCP
Java Community Process - Proceso de Comunidad Java.
Comunidad de programadores que desarrollan nuevos paquetes de aplicaciones.

JDBC
Java Database Connectivity - Conectividad de Base de Datos Java.
API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema de operación donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL del modelo de base de datos que se utilice.

JDK
Java Development Kit - Kit de Desarrollo Java.

JMS
Java Message Service - Servicio de Mensajes Java.
Solución de Sun para los sistemas de mensajes.

JMX
Java Management Extensions - Extensiones de Gestión Java.
API encargada de definir todo aquello referente a la monitorización y administración de aplicaciones basadas en Java.

JNDI
Java Naming Directory Interface - Sistema de Nombrado en Java.
Especificación que permite ubicar fuentes de datos en directorios distintos, distribuidos o no.

JRE
Java Runtime Environment - Entorno de Ejecución Java.
Proporciona únicamente un subconjunto del lenguaje de programación Java sólo para ejecución.

JSPA
JAIN Service Provider APIs - APIs del Proveedor de Servicio JAIN.

JSR
Java Specification Request - Petición de Especificación Java.

JTA
Java Transaction API - API de Transacción Java.
Interfaz estándar para el manejo de transacciones en las aplicaciones Java. Usando transacciones, se puede proteger la integridad de los datos en las bases de datos y manejar el acceso a esos datos por aplicaciones o ejemplares

simultáneos de la aplicación. Una vez que una transacción comienza, todas las operaciones transaccionales deben terminar con éxito o todas se deben deshacer.

JTAPI
Java Telephony API - API de Telefonía Java.

LAN
Local Area Network - Red de Área Local.

LDAP
Lightweight Directory Access Protocol - Protocolo Ligero de Acceso a Directorios.
Servicio de directorio ordenado y distribuido para buscar diversa información en un entorno de red.

MAP
Mobile Application Part - Parte de Aplicación Móvil.

MBEAN
Managed Bean - Bean Gestionado.
Un modo sencillo de ver los MBeans es pensar que son aquellas aplicaciones que se encargan de monitorizar otras aplicaciones.

MEDIA GATEWAY
Denominación genérica para referirse a varios productos agrupados bajo el protocolo MGCP (Media Gateway Control Protocol). La principal misión de un Media Gateway es la conversión IP/TDM bajo el control de un Softswitch.

MGCP
Media Gateway Control Protocol - Protocolo de Control Media Gateway.
Es un protocolo de control de dispositivos, donde un gateway esclavo (MG, Media Gateway) es controlado por un maestro (MGC, Media Gateway Controller).

MIDDLEWARE
Agentes software que actúan como un intermediario entre diferentes componentes de aplicación.

MMS
Multimedia Messaging System - Sistema de Mensajería Multimedia.

MSC
Mobile Switching Centre - Centro de Conmutación Móvil.
Proporciona conectividad entre la Red Telefónica de Conmutación Pública y las estaciones base, y entre todos los abonados móviles de un sistema.

NAT
Network Address Translation - Traducción de Dirección de Red.
Estándar que permite a una LAN utilizar un conjunto de direcciones IP internamente y un segundo conjunto de direcciones externamente. El dispositivo que hace NAT se sitúa en el punto de salida a Internet y realiza todas las traducciones de direcciones IP que sean necesarias.

NEP
Network Equipment Provider - Proveedor de Equipo de Red.

NGIN
Next Generation Intelligent Networks - Redes Inteligentes de Nueva Generación.

NGN
Next Generation Network - Red de Próxima Generación.
Es una arquitectura de red orientada a reemplazar las redes PSTN de telefonía para servicios de voz y multimedia.

NMS
Network Management Systems - Sistemas de Gestión de Red.

OMA
Open Mobile Alliance - Alianza Móvil Abierta.

OSA
Open Services Architecture - Arquitectura de Servicios Abierta.
Permite que las aplicaciones usen las capacidades de red de forma normalizada, el desarrollo de aplicaciones por terceros y añadir capacidades de red.

OSS
Operations Support Systems **-** *Sistemas de Soporte de Operaciones.*
Sistemas que ejecutan funciones de gestión, inventario, planeamiento y reparación en redes.

PMI
Performance Monitoring Interface - Interfaz de Supervisión de Rendimiento.

POJO
Plain Old Java Objetct - Antiguo Objeto Java Plano.
Este nombre se les da a las clases que no son de algún tipo especial (EJBs, JavaBeans, etc) y no cumplen ningún otro rol ni implementan alguna interfaz especial.

PRODUCTIZATION
Estandarización de los elementos en una oferta software.

PSTN
Public Switched Telephone Network - Red Telefónica Pública Conmutada.

RA
Resource Adaptor - Adaptador de Recurso.

RFC
Request For Comments - Peticiones de Comentarios.
Conjunto de notas técnicas y organizativas donde se describen los estándares o recomendaciones de Internet.

RI
Reference Implementation - Implementación de Referencia.

RMI
Remote Method Invocation - Invocación Remota de Métodos.
Mecanismo que permite invocar métodos de objetos escritos en Java que residen en otro ordenador (remotos) de la misma forma que si los objetos fueran locales. Tanto el componente que invoca al método, como el que contiene el método invocado deben ser escritos en Java.

RTP
Real-Time Transport Protocol - Protocolo de Transporte de Tiempo Real.
Es un protocolo de nivel de transporte utilizado para la transmisión de información en tiempo real como por ejemplo audio y video en una video-conferencia.

SBB
Service Building Block - Bloque de Construcción de Servicios.

SCF
Service Capability Features - Características de Capacidades de Servicio.

SCS
Service Capability Servers - Servidores de Capacidades de Servicio.

SDK
Software Development Kit - Kit de Desarrollo Software.

SDP
Session Description Protocol - Protocolo de Descripción de Sesión.
Lo utiliza SIP para describir las capacidades multimedia de los participantes en la llamada y negociar un conjunto común de capacidades multimedia a utilizar.

SIP
Session Initiation Protocol - Protocolo de Inicio de Sesión.

Es un protocolo de señalización para conferencia, telefonía, presencia, notificación de eventos y mensajería instantánea a través de Internet.

SLEE
Service Logic Execution Environment - Entorno de Ejecución de Lógicas de Servicio.

SNMP
Simple Network Management Protocol - Protocolo de Gestión de Red Simple.
Protocolo propuesto por la IETF para la gestión de Internet.

SOA
Service Oriented Architecture - Arquitectura Orientada a Servicios.
Es un concepto de arquitectura de software que define la utilización de servicios para dar soporte a los requerimientos de software del usuario.

SOFTSWITCH
Término genérico para cualquier software pensado para actuar de pasarela entre la red telefónica y algún protocolo de VoIP, separando las funciones de control de una llamada del *media gateway*.

SQL
Structured Query Language - Lenguaje de Consulta Estructurado.
Lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas.

SS7
Signaling System 7 - Sistema de Señalización por canal común nº 7.

TCAP
Transaction Capabilities Application Part - Parte de Aplicación de Capacidades de Transacción.
Interfaz entre las aplicaciones de usuario y el servicio de la capa de red.

TCK
Technology Compatibility Kit - Kit de Compatibilidad Tecnológica.

TMN
Telecommunication Management Network - Red de Gestión de Telecomunicaciones.
Entorno que garantiza la interconexión y comunicación entre sistemas y redes de telecomunicaciones heterogéneos.

UML
Unified Modelling Language - Lenguaje Unificado de Modelado.
Lenguaje gráfico para visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software.

URI
Uniform Resource Identifier - Identificador Uniforme de Recursos.
Texto corto que identifica unívocamente cualquier recurso (servicio, página, documento, dirección de correo electrónico, enciclopedia, etc.) accesible en una red. También se utiliza URL.

VoIP
Voice over Internet Protocol - Voz sobre el Protocolo de Internet.
Sistema de enrutamiento de conversaciones de voz mediante paquetes basados en IP por la red de Internet.

VPN
Virtual Private Network - Red Virtual Privada.
Es una tecnología de red que permite una extensión de la red local sobre una red pública o no controlada, como por ejemplo Internet.

WBEM
Web Based Entreprise Management - Gestión Empresarial Basada en Web.
Iniciativa que permite las tareas de gestión haciendo uso de la tecnología web.

XML
Extensible Markup Language - Lenguaje de Marcas Extensible.
Es un subconjunto derivado de SGML que permite definir otros lenguajes para la descripción de datos.

XMPP
Extensible Messaging and Presence Protocol - Protocolo de Presencia y Mensajería Extensible.
Es un protocolo extensible, abierto y estándar basado en XML para el intercambio en tiempo real de mensajes y presencia entre dos puntos en Internet.