Universidad Autónoma Metropolitana

División de Ciencias Básicas e Ingeniería

Ingeniería en Computación

Reporte de Proyecto Terminal

Implementación y Simulación de un Cliente IMS

Arturo Sánchez Martínez

205205487

Asesor: Arturo Zúñiga López

No. económico: 28779

Fecha de entrega: Abril 2011

A mis Padres

Agradecimientos

En primer lugar quiero agradecer el apoyo que me han brindado a lo largo de todo este camino mis padres Rubén e Isabel, y mis hermanos Jesús y Alejandra. Papá, gracias por enseñarme a que uno nunca debe darse por vencido. Mamá, gracias por enseñarme que con responsabilidad y perseverancia se pueden alcanzar todas las metas que uno se proponga. María Guadalupe, gracias por ser mi amiga y entregarme tantas sonrisas y alegrías y por todo ese apoyo anímico que siempre me brindaste, gracias porque siempre potenciaste mi confianza en las capacidades que poseo.

También quiero agradecer a mis amigos de la universidad por haber estado a mi lado durante este camino recorrido. Arturo, Luis Fernando, Hugo, Guillermo, gracias por hacer del paso por la universidad una experiencia de vida.

Finalmente, quiero extender mis más sinceros agradecimientos al profesor Arturo Zúñiga López por su apoyo y sus consejos para mi desarrollo profesional y su excelente disposición todo el tiempo, gracias por haberme permitido desarrollar este trabajo de título bajo su tutela.

En este sentido agradezco la posibilidad de haber abordado un tema actual, interesante y de amplio interés para la industria de las Telecomunicaciones como IMS.

“El fracaso consiste en no persistir, en desanimarse después de un error, en no levantarse después de caer”.

Thomas Alva Edison (1847-1931).

“Nunca consideres el estudio como una obligación, sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber”.

Albert Einstein (1879-1955) Científico alemán nacionalizado estadounidense.

Resumen

El uso de protocolos abiertos combinados con las ya baratas conexiones a internet garantizan que el internet y las redes continúen creciendo en las siguientes décadas. Actualmente el mundo requiere un mejoramiento de las telecomunicaciones para que provean calidad de servicio *QoS* (*Quality of Service*) a los usuarios.

IMS (*IP Multimedia Subsystem*) es la pieza clave de la arquitectura 3G. Es un sistema de control de sesión que puede proporcionar acceso celular ubicuo sobre conmutación de paquetes (*CP*) para todos los servicios que proporciona Internet.

- Permite usar las tecnologías celulares para proporcionar ubicuidad en el acceso y tecnologías de internet para ofrecer servicios más atractivos.

- Facilita todos los servicios sólo empleando conmutación de paquetes.

- Las tecnologías IP sobre las que se basa IMS (SIP para el control de sesiones e IPv6 para el transporte de red) hacen posible el desarrollo rápido de servicios y la reducción de costes de operación e infraestructura (cualidades del mundo Internet).

¿Por qué necesitamos IMS?

IMS facilita:

- Sesiones IP multimedia compuestas por flujos y contenidos multimedia diversos, con un nivel adecuado de *QoS* para cada contenido.

- Tarificación: Conseguir tarificar de forma apropiada. Una videoconferencia puede ser interesante que se tarifique por tiempo en vez de por bytes. *IMS* permite ofrecer información del servicio que usa el usuario para que el operador tarifique en consecuencia.

- Servicios integrados: Usar servicios desarrollados por terceras partes que se integren con otros ya existentes (Ej. Sobre los sms, integrar un servicio *text-tospeech* para invidentes).

- Define interfaces estándar para ser usados por los desarrolladores de servicios.

- Comunicación orientada a sesión de un usuario a otro/s usuario/s o de un usuario a 1 servicio. (Ej: Un usuario IMS - Un usuario de Internet)

- Identificador de usuarios con “nombres”, no con números de teléfono.

Contenido

Índice de Imágenes.

Introducción

En los últimos años han aparecido distintos sistemas de comunicaciones personales, la mayoría de ellos basados en el uso de Internet. El uso cada vez mayor de aplicaciones multimedia en tiempo real que intercambian contenidos 3D con calidad de servicio (*QoS*) impone una redefinición de las redes actuales.

El proceso de estandarización de la tecnología UMTS como estándar global 3G ha resultado en el desarrollo de IMS (*IP Multimedia Subsystem*), que es una arquitectura de control, basada en el protocolo IP, que proporciona un conjunto de funcionalidades que resultan esenciales en la provisión de los servicios multimedia de valor añadido con los requisitos de *QoS* que están previstos en el futuro de las redes móviles de tercera generación. Así, IMS provee al usuario, medios para acceder a un amplio rango de servicios implementados mediante los Servidores de Aplicación (*AS*). Las funcionalidades de control de sesión en IMS permiten establecer sesiones multimedia entre equipos de usuario final (*User Equipment, UE*). En este contexto, muchos de los servicios que actualmente están siendo considerados en el futuro de las redes 3G son de naturaleza multiusuario, en los que varios usuarios participan en la ejecución del servicio, de modo que la información debe ser intercambiada entre múltiples UEs.

De este modo, el desarrollo de especificaciones para una red de internet de próxima generación (*Next Generation Network, NGN*), permitirá desarrollar nuevas infraestructuras con capacidades y característica que soporten la apropiada provisión de los servicios multimedia del futuro sobre las distintas redes de acceso fijas y móviles disponibles.

El objetivo principal de este trabajo es integrar un cliente móvil IMS en una red que simule la arquitectura IMS. Esto implica simular una maqueta IMS bajo el SO Linux a partir de la herramienta Open IMS Core y desarrollar un cliente IMS usando *Java 2 Micro-Edition Platform* que interactúe con dicha maqueta utilizando el protocolo SIP.

Los objetivos específicos del trabajo son:

Desarrollar una aplicación que simule un cliente móvil IMS utilizando herramientas como SWTK, J2ME y SIP API JSR180.

- Implementar la red IMS utilizando Open IMS Core en Linux.
- Integrar al cliente con la red IMS.
- Evaluar la integración del cliente con la red IMS.

Plataforma IMS [2,3,4]

IMS (*IP Multimedia Subsystem*) es un estándar que define una arquitectura genérica para ofrecer voz sobre IP (*VoIP*) y servicios multimedia; y que forma parte del núcleo de la arquitectura de las nuevas redes NGN (*Next Generation Networking*).

Inicialmente fue especificado por el 3GPP/3GPP2, y en la actualidad está siendo acogido por otras entidades de estandarización como ETSI/TISPAN, OMA (*Open Mobile Alliance*) y JPC (*Java Community Process*). Este estándar soporta múltiples tipos de acceso como pueden ser GSM, WCDMA, CDMA2000, banda ancha y WLAN. Desde el punto de vista de los usuarios, los servicios basados en IMS permiten comunicaciones, usuario a usuario y usuario a contenido, de varias maneras (voz, texto, fotos y video, o una combinación de estos) de una forma personal y controlada. Los servicios basados en IMS tienen la capacidad de aumentar significativamente y cambiar la manera en que nos comunicamos. Los usuarios pueden cambiar entre diferentes maneras de comunicarse, añadir capacidades multimedia y participantes en una comunicación establecida, abrir grupos de información, etc.

Desde el punto de vista de las operadoras, IMS define una arquitectura horizontal donde servicios y funciones comunes ya definidos pueden ser reutilizados por múltiples aplicaciones. Esta arquitectura horizontal de IMS permite interoperabilidad, roaming, proporciona transporte de funciones de control, de tarificación y aspectos de seguridad. IMS permitirá a las operadoras controlar y facturar cada uno de los servicios que preste de manera multimedia. Además se integra perfectamente con redes de voz y de datos ya existentes adoptando muchos de los beneficios claves de estos dominios; lo que hace de IMS una llave que permita la convergencia fijo-móvil.

Entre las principales características de la arquitectura IMS se debe mencionar:

Soporte de sesiones en tiempo real (voz y video conferencia) y de no-tiempo real (*PTT/Push-To-Talk*, *PTS/Push-To-Show*, presencia, mensajería) sobre redes IP.

Interfaces y protocolos abiertos.

Plataforma que permite desarrollo de sofisticados servicios de valor agregado como *streaming* de audio y video online.

Integración horizontal dado que existen funciones genéricas en estructura e implementación que pueden ser reutilizadas por todos los servicios de la red (descubrimiento, enrutamiento, cobro, presencia, administración de usuarios, etc.) con la consiguiente disminución de costos para el operador.

Soporte de redes de acceso de distinto tipo que va de la mano con la movilidad generalizada.

A continuación podemos observar una aproximación a la arquitectura de IMS, [Ilustración 1]:

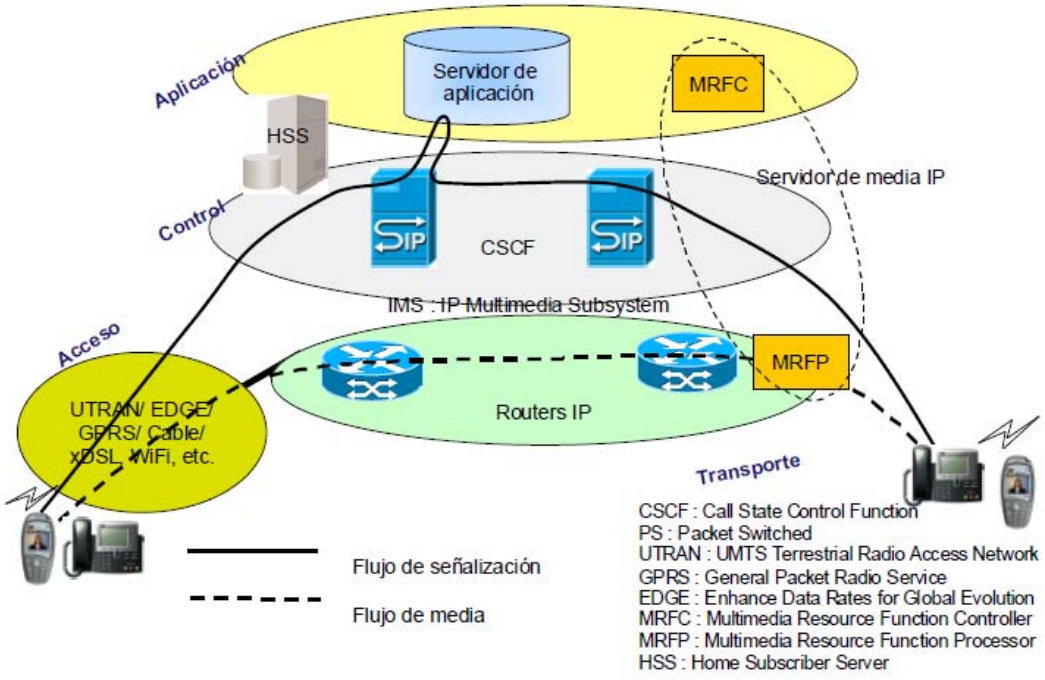

Ilustración 1: Arquitectura IMS

En la capa superior (capa de aplicaciones) se incluyen aplicaciones y contenidos de servidor para ejecutar servicios de valor añadido para el usuario. La capa de control comprende servidores de control de red para mantenimiento de llamadas o establecer, modificar y liberar sesiones. El elemento más importante de esta capa es el CSCF (*Call Session Control Function*), también conocido como servidor SIP. Esta capa también contiene un juego completo de funciones soportadas, como suministro y tarificación.

Las componentes básicas de la arquitectura IMS son las siguientes:

1. UE (*User Equipment*)

2. HSS (*Home Subscriber Server*)

3. CSCF (*Call Session Control Function*)

4. SLF (*Subscription Locator Function*)

5. PDF (*Policy Decision Function*)

6. MRFC (*Multimedia Resource Function Controller*)

7. MRFP (*Multimedia Resource Function Processor*)

8. BGCF (*Breakout Gateway Control Function*)

9. AS (*Application Server*)

10. MGCF (*Media Gateway Control Function*)

11. IMS-MGW (*IMS Media Gateway*)

12. T-SGW (*Trunking Signaling Gateway*)

Ilustración 2: Servicios IMS

Proyecto Open IMS Core [11]

A continuación se presenta el Proyecto “Open IMS Core”, iniciativa que potencia el desarrollo de una plataforma IMS básica con fines docentes y experimentales. El fin de esta introducción es mostrar las capacidades de este sistema para establecer su uso en el desarrollo de nuestro trabajo, [Ilustración 3].

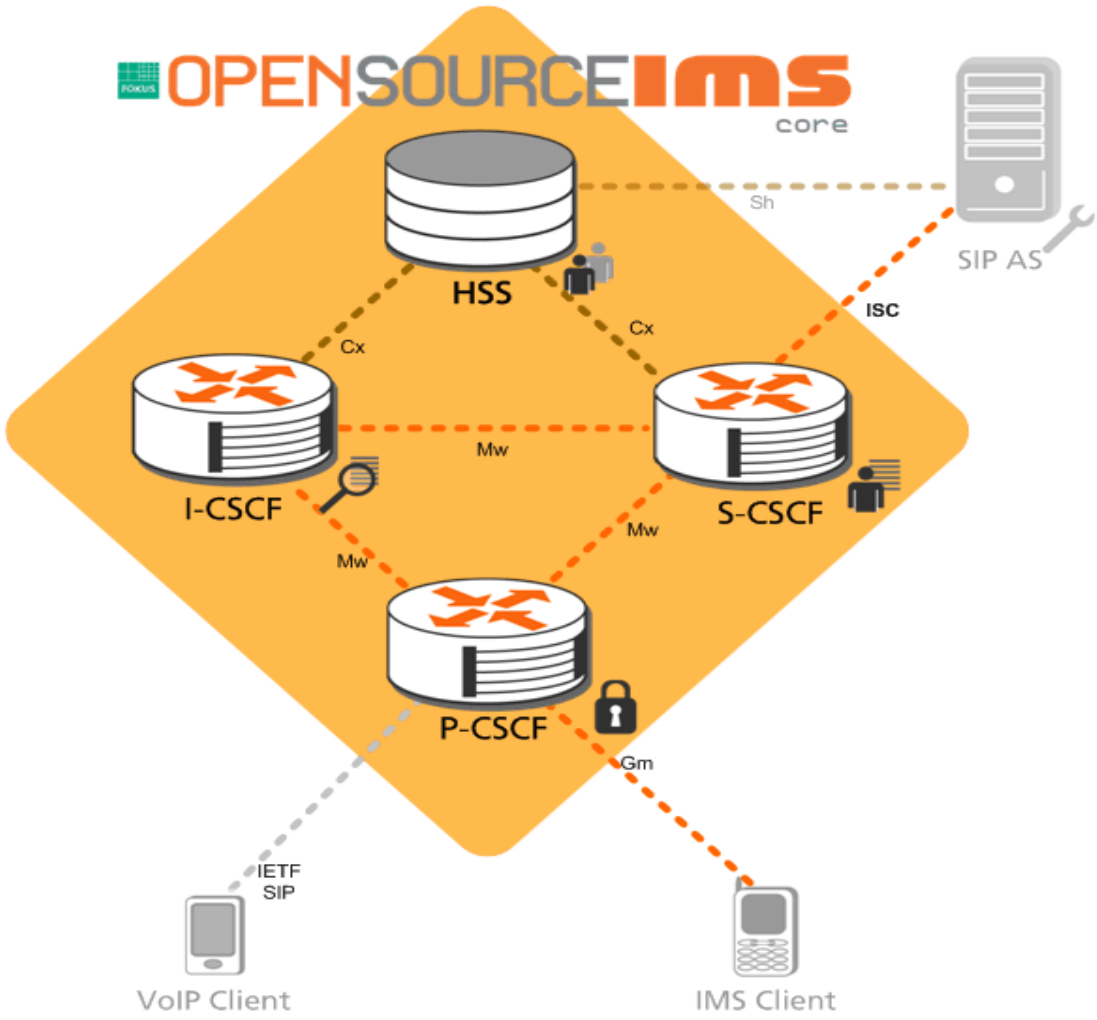

Ilustración 3: Arquitectura Open IMS Core

Descripción [1,2,3,4,6,11]

Open IMS Core es una implementación de las entidades *CSCFs* y de un *HSS* de bajo peso que en conjunto forman los elementos core de todas las arquitecturas *IMS/NGN* especificadas dentro de *3GPP, 3GPP2, ETSI TISPAN* y la iniciativa *PacketCable*. Las cuatro componentes (*S-CSCF, P-CSCF, I-CSCF, HSS*) están basadas en software Open Source.

Esta plataforma ha sido desarrollada por FOKUS (*Fraunhofer - Institute for Open Communication Systems*), institución de origen alemán y sin fines de lucro que provee servicios de Investigación y Desarrollo de sistemas de comunicación de próxima generación. Su principal objetivo es potenciar el desarrollo de nuevas tecnologías entregando un marco de trabajo basado en software open source que permita establecer prototipos de tecnologías acorde al mercado y los requerimientos del usuario final.

Las líneas de investigación de FOKUS son las siguientes:

- 3G - Infraestructura y servicios para soluciones móviles.
- Habilitación de tecnologías para ambientes inteligentes.
- Desarrollo y testing de modelos basados en software.

En este sentido, una temática que particularmente aborda FOKUS son las tecnologías emergentes. Dentro de este marco se ha diseñado e implementado en base a herramientas open source la plataforma de pruebas denominada “Open IMS Core” que permite testear la tecnología y probar servicios. Dicho desarrollo está disponible para que cualquier persona pueda experimentar y trabajar con fines experimentales y/o docentes. En ningún caso constituye un kit de desarrollo para la implementación de plataformas comerciales. Más aún, debe tenerse en cuenta que dada la cantidad de organizaciones involucradas en IMS, posteriormente podrían existir patentes de por medio que deberán ser respetadas.

IMS aún se encuentra en etapas de prueba con un número creciente de operadores interesados en la tecnología. Mientras que en el área de *VoIP* existen muchos proyectos open source asociados a clientes SIP, proxies y herramientas (alrededor del estándar SIP IETF), prácticamente no existen proyectos open source con foco en IMS. Luego, el objetivo principal de esta iniciativa es llenar ese vacío para entregar una herramienta que permita probar y desarrollar servicios IMS y permitir el estudio del core de dicha arquitectura.

Entidades de Redes IMS

Para lograr una mejor comprensión de IMS y su funcionamiento no es necesario explicar cada uno de los elementos que conforman su arquitectura. Basta con profundizar sobre los componentes que forman el núcleo de IMS en el plano de control e interconexión (*HSS y CSCF*) y los componentes que forman el plano de servicios (*AS*).

Terminal IMS

Se trata de una aplicación sobre un equipo de usuario que emite y recibe solicitudes SIP. Se materializa por un software instalado sobre una PC, sobre un teléfono IP o sobre una estación móvil UMTS ("*User Equipment"* o "UE").

Home Subscriber Server (*HSS*)

El Home Subscriber Server (HSS) registra toda la información acerca de los usuarios y servicios dentro de un dominio IMS. El HSS registra todos los distintos perfiles, identidades y credenciales que conforman los usuarios para realizar AAA (*Authentication, Authorization, and Accounting*); además contiene información acerca de los distintos servicios que tienen habilitados así como el *Serving-CSCF* (*S-CSCF*) al que se registraron, permitiendo saber su punto de conexión y habilitar servicios de itinerancia (*roaming*). En cuanto a los servicios, contiene reglas para permitir o negar su acceso de manera automática. Para permitir la interconexión con distintas redes también cuenta con parámetros de calidad que permiten la comunicación exitosa entre clientes de distintas tecnologías de acceso. En caso de que un solo HSS no tenga la capacidad de almacenar toda esta información se cuenta con un *Service Location Function* (*SLF*) el cual es una simple relación entre los usuarios y servicios cuya administración se delega al *HSS*. Esto permite escalar la funcionalidad de los repositorios de información al repartir los registros entre varias *HSS* y administrar los registros mediante un *SLF*.

CSCF

Call/Service Control Function (*CSCF*) es la función que controla las llamadas y sesiones dentro de IMS. Se encarga de establecer, destruir y enrutar llamadas, tratando éstas como sesiones SIP. Por sí solo proporciona un ambiente para ofrecer servicios de telefonía tradicional dentro del dominio IP. Junto con HSS forma el núcleo de IMS el cual permite migrar la telefonía tradicional del mundo de circuitos al mundo de paquetes. Para cumplir con esta tarea exigente CSCF se descompone en tres funciones especializadas: *Proxy-CSCF* (*P-CSCF*), *Interrogating-CSCF* (*I*-CSCF), y *S-CSCF*.

P-CSCF

Proxy-CSCF (*P-CSCF*) actúa como un proxy SIP entre el UE y el dominio IMS. Tiene cuatro principales funciones las cuales son empleadas para comenzar y finalizar la comunicación entre el UE y el dominio IMS. Se encarga de realizar la compresión del protocolo SIP con el objetivo de minimizar el uso del ancho de banda de la red de acceso. Autentica al usuario, evitándole esta tarea a las demás funciones CSCF al realizar asociaciones de seguridad con *Internet Protocol Security* (*IPSec*), permitiendo garantizar la integridad de los mensajes SIP intercambiados. Es el encargado de establecer las políticas que reservan los recursos multimedia necesarios, así como garantizar la *QoS* solicitada vía *Policy Decision Function* (*PDF*). Dado que IMS piensa reemplazar la telefonía tradicional, se requiere ajustar a las necesidades regulatorias de emergencia (como son llamadas al 911 y servicios de operador); *P-CSCF* se encarga de detectar estas llamadas y enrutarlas con el trato especial que exigen. El *P-CSCF* puede localizarse dentro o fuera del dominio IMS.

I-CSCF

Interrogating-CSCF (*I-CSCF*) se encarga de interrogar las solicitudes generadas por los usuarios con el fin de determinar el dominio IMS o *Serving-CSCF* (*S-CSCF)* destino al cual se dirige la sesión para ser atendida exitosamente; dentro de esta capacidad se incluye la obtención del nombre del siguiente elemento que atenderá la solicitud. Adicionalmente determina si el S-CSCF cuenta con la capacidad necesaria para atender las solicitudes después de consultar con el

HSS. El *I-CSCF* normalmente se encuentra en las orillas de cada dominio IMS, aunque es posible que se encuentre fuera del dominio. Cuando se decide ocultar la topología de la red mediante técnicas *Topology Hiding Internetwork Gateway* (*THIG*), el *I-CSCF* se encarga de encriptar los encabezados correspondientes.

S-CSCF

Serving-CSCF (*S-CSCF*) es la función más importante del *CSCF* ya que se encargada de enrutar las llamadas y sesiones hacia su destino final. Además de que se encarga de comunicar al usuario con el AS (salvo aquel que fue contactado por el *I-CSCF*). Primeramente verifica la autenticidad de los usuarios consultando al *HSS*, después registra la localización del usuario en el *HSS* haciendo un mapeo entre la dirección IP del que acede el usuario y su identidad pública, finalmente verifica que el usuario tenga activado el servicio al cual desea acceder.

Usuarios dentro de IMS

Toda red requiere identificar de manera única a sus usuarios, en particular si tiene como objetivo mejorar la *QoE* como es el caso de IMS. Anteriormente *PSTN* empleaba los números telefónicos para identificar a sus usuarios. Bajo el esquema de IMS los usuarios se identifican mediante Perfiles, los cuales se componen de al menos una Identificación Privada de Usuario y un Perfil de Servicio. Además, los usuarios dentro de IMS poseen dos tipos de identificaciones: la pública y la privada.

Identificación Pública de Usuario

La Identificación Pública de Usuario es aquella que se conoce fuera del dominio IMS del operador. En otras palabras, es la manera en que el usuario se identifica ante el mundo. Esto permite que un usuario pueda utilizar varios UEs con distintas funcionalidades y capacidades con la misma cuenta de usuario. Estos se componen de al menos un *Telephone Universal Resource Identifier* (*TEL-URI*) o bien un *Session Initiation Protocol Universal Resource Identifier* (*SIP-URI*).

Dentro de IMS las identificaciones públicas son empleadas para enrutar señalización SIP. Se pueden registrar simultáneamente varias identificaciones

Públicas de manera implícita, ahorrando ancho de banda y recursos de red. Además esto permite utilizar el mismo UE con distintas identificaciones simultáneamente.

Identificación Privada de Usuario

La Identificación Privada de Usuario es aquella que únicamente es conocida por el dominio IMS del operador. Es empleada para realizar la autenticación y suscripción. Dado que son empleadas primordialmente entre los UEs y el operador, éstas pueden o no ser conocidas por el usuario. Es importante mencionar que la Identificación Privada no identifica al usuario, como lo hace la Identificación Pública, sino que identifica su subscripción dentro del dominio IMS. Un usuario puede poseer varias Identificaciones Públicas, mientras que únicamente puede poseer una Identificación Privada, ya que ésta posee una subscripción con el operador y puede tener distintos UEs para acceder a sus servicios. A partir del *Release* 6 de IMS, un usuario puede tener más de una subscripción, pero se continúa respetando la relación N: M entre las Identificaciones Públicas (N) y Privadas (M), donde N > M.

Perfil de Servicio

El Perfil de Servicio es un conjunto de información almacenado dentro del *HSS* que detalla los servicios que tiene registrado el usuario. Dicha información es intercambiada entre el *HSS* y *S-CSCF* cuando un usuario solicita acceder a cualquier servicio; únicamente les provee el servicio a los usuarios que cumplen con los criterios definidos. El perfil de servicio se descompone en tres principales atributos:

- Identificación Pública: *SIP-URI* que identifica al servicio
- Autorización de Servicio: reglas definiendo quienes pueden acceder al servicio.
- Criterio de Filtrado Inicial (*Initial Filter Criteria*: iFC): reglas empleadas por S-CSCF para determinar la ruta hacia el AS que surte dicho

Call State Control Function (*CSCF*)

El control de llamada iniciado por un terminal IMS tiene que ser asumido en la red nominal (red a la cual el usuario suscribe sus servicios IMS) ya que el usuario puede suscribir a una gran cantidad de servicios y algunos de ellos pueden no ser disponibles o pueden funcionar de manera diferente en una red visitada, entre otros por problemas de interacción de servicios. Eso induce la definición de tres entidades: "*Proxy CSCF*" o "*P-CSCF*", "*Interrogating CSCF"* o "*ICSCF"* y *"Serving CSCF" o "S-CSCF".*

Uso en el contexto del Trabajo

El proyecto Open IMS Core constituye una plataforma de pruebas IMS que posee las principales funcionalidades del core de la arquitectura, permitiendo explotar las características básicas que requiere el desarrollo del trabajo "Implementación y Simulación de un cliente IMS". A través de la página web de *FOKUS* se puede acceder a todo el desarrollo en software para instalar, configurar y utilizar los módulos mencionados previamente. En este sentido, se decidió utiliza eta plataforma principalmente por dos razones. Por un lado el "desarrollar desde cero" todas las componentes para implementar las entidades tomaría mucho tiempo y por otra parte el proyecto en sí ha sido adoptado con gran aceptación por variadas universidades alrededor del mundo.

Posterior a dicho análisis, en conjunto con el profesor guía se decidió utilizar esta herramienta para levantar las entidades IMS, y con estas poder integrar el cliente móvil que será desarrollado.

IMS y SIP [1,2,11,14]

SIP es un protocolo de señalización definido por el "*Internet Engineering Task Force*" o "*IETF"* que permite el establecimiento, la liberación y la modificación de sesiones multimedia. SIP es usado en el IMS como protocolo de señalización para el control de sesiones y el control de servicio. El reemplaza entonces a la vez los protocolos "*ISDN User Part*" o "*ISUP*" y "*Intelligent Network Application Part*" o "*INAP*" del mundo de la telefonía aportando la capacidad multimedia. El hereda de ciertas funcionalidades de los protocolos "*Hyper Text Transport Protocol*" o "http" utilizados para navegar sobre la web, y "*Simple Mail Transport Protocol*" o "*SMTP*" usados para la transmisión de mensajes electrónicos (*e-mails*). SIP se apoya sobre un modelo transaccional cliente/servidor como "http". El direccionamiento utiliza el concepto de "*Uniform Resource Locator*" o "*URL SIP*" parecido a una dirección e-mail. Cada participante en una red SIP es alcanzable por medio de una URL SIP. Por otra parte, las solicitudes SIP son satisfechas por respuestas identificadas por un código numérico. Cabe subrayar que la mayor parte de los códigos de repuestas SIP provienen del protocolo http. Por ejemplo, cuando el destinatario no se ha podido ubicar, un código de respuesta "4004 Not Found" es enviado. Un pedido SIP está constituido de encabezamientos (*headers*) como mando SMTP. SIP de igual manera que SMTP es un protocolo textual.

El protocolo SIP proporciona los siguientes servicios:

- Localización de usuarios: los usuarios pueden moverse a una locación remota y, no obstante, continuar disponiendo de las funcionalidades ofrecidas por las aplicaciones SIP.

- Disponibilidad de usuarios: SIP permite determinar si un usuario está disponible para recibir llamadas.

- Capacidades de comunicación: el numero, tipo y calidad de medios a intercambiar se negocia de forma dinámica dentro del establecimiento de la sesión.

- Establecimiento de sesiones: SIP permite establecer sesiones de comunicación multimedia, tanto punto a punto como multipunto.

Gestión de la sesión: una vez establecida una sesión, SIP permite terminarla, transferirla, modificarla o invocar nuevos servicios sobre ella.

Modelo transaccional

SIP es un protocolo transaccional, es decir, en las interacciones entre componentes tienen lugar mediante una serie de intercambios independientes de mensajes. Una transacción SIP consiste en una única petición y su respuesta o respuestas asociadas, que consistirán habitualmente en cero o mas respuestas provisionales y una o varias respuestas finales.

La primera diferencia entre peticiones y respuestas SIP se da en la primera línea ya que, mientras en las peticiones consiste en un método que indica la naturaleza de la petición y una URI que determina donde ha de ser enviada, en las respuestas consiste en un código de respuesta junto con su descripción. El resto de la cabecera de los mensajes SIP consiste en una serie de líneas, cada una de ellas identificada por una etiqueta de la cabecera, muy similares a las que aparecen en un mensaje de correo electrónico.

Para las peticiones SIP, en la RFC 3261 se definen los siguientes métodos:

- REGISTER: Permite a un usuario registrar su dirección IP actual y asociarla con las URIs en las que desea recibir llamadas.
- INVITE: Se utiliza para establecer una sesión de intercambio de medio entre 2 agentes.
- ACK: Confirma la correcta recepción de un mensaje.
- CANCEL: Cancela una transacción pendiente.
- BYE: Termina una sesión de intercambio de medios.

Aparte de los métodos definidos en la especificación SIP base, existen multitud de extensiones SIP definidas en RFCs complementarias. Entre los métodos adicionales más importantes se encuentran:

INFO: Para el envío de señalización fuera de banda durante el transcurso de la llamada.

PRACK: ACK de una respuesta provisional.

UPDATE: Permite actualizar la descripción de medios de uno de los participantes en un determinado dialogo sin afectar al estado del mismo.

REFER: Transfiere al usuario a una determinada URI.

SUBSCRIBE: Solicita ser notificado cuando tenga lugar un determinado evento.

NOTIFY: Notifica a los suscriptores de la ocurrencia de un determinado evento.

MESSAGE: Permite intercambiar mensajería instantánea.

Las respuestas SIP son un claro ejemplo del parecido que el modelo transaccional de SIP guarda con el de HTTP. Las principales familias de respuestas de SIP son:

1xx: Provisional o informativo. Indica que la petición está en curso pero aun no ha sido completada.

2xx: Éxito. La petición ha sido completada satisfactoriamente.

3xx: Redirección. La petición debe reiniciarse con un destino diferente.

4xx: Error de cliente. La petición contiene un error sintáctico o no puede ser atendida en el servidor destino.

5xx: Error de servidor. El servidor ha sido incapaz de atender una petición aparentemente valida.

6xx: Fallo global. La petición no puede ser atendida en ningún servidor

Herramientas de desarrollo [11,12,13]

Diferentes herramientas de software y hardware, así como bibliotecas están disponibles para que los desarrolladores realicen ricas aplicaciones para dispositivos portátiles como son los teléfonos celulares, PDAs entre otros. Para el desarrollo de nuestro cliente, la plataforma de desarrollo elegida fue J2ME, [Ilustración 4].

Ilustración 4: Herramientas de Desarrollo

Java Development Kit (JDK)

JDK consiste de un compilador de java y es un software que provee herramientas de desarrollo para la creación de programas en java. Puede instalarse en una computadora local o en una unidad de red. Este puede ser descargado de manera

libre desde la pagina web de ORACLE. Para el desarrollo del cliente IMS, jdk 1.6 fue usado.

J2ME Sun Wireless Toolkit

Sun Java Wireless Toolkit (anteriormente conocido como *Java 2 Platform*, *Micro Edition (*J2ME) *Wireless Toolkit* es un conjunto de herramientas que permiten el desarrollo de aplicaciones que puedan funcionar en teléfonos celulares, asistentes digitales personales, y otros dispositivos móviles pequeños. El kit de herramientas incluye el entorno de emulación, la optimización del rendimiento y las características de ajuste, documentación y ejemplos que los desarrolladores necesitan para crear sus aplicaciones.

Netbeans

NetBeans es un proyecto exitoso de código abierto con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo. *Sun MicroSystems* fundó el proyecto de código abierto *NetBeans* en junio 2000 y continúa siendo el patrocinador principal de los proyectos.

NetBeans IDE es un entorno de desarrollo - una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Está escrito en Java - pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el *NetBeans* IDE. *NetBeans* IDE es un producto libre y gratuito sin restricciones de uso.

Configuraciones [9,10]

J2ME presenta dos configuraciones: *CLDC* y *CDC*. La primera se dedica a dispositivos con estrictas limitaciones de memoria, capacidad de cálculo, consumo y conectividad de red. Por otro lado, *CDC* se encarga de dispositivos con más potencia. Parte de *CLDC* es un subconjunto de *CDC*, por lo que la portabilidad de aplicaciones se puede conseguir cuando nos movemos de un entorno más restringido a otro más rico. De la misma manera, y siguiendo en el hilo de la portabilidad, una aplicación en J2ME podrá ejecutarse en J2SE normalmente, salvo que se utilicen las bibliotecas específicas de J2ME.

CLDC

Veamos más detalladamente algunas características de *CLDC*, ya que es la configuración que utilizamos en el desarrollo de nuestro cliente IMS. Comencemos por las propiedades mínimas requeridas a un dispositivo para poder desarrollar con esta configuración:

- De 160 a 512 Kbytes de memoria disponible para el entorno de Java.
- Un procesador de 16 o 32 bits.
- Consumo de energía bajo (generalmente utilizan baterías).
- Permiten algún tipo de conectividad a una red

Wireshark

Antes conocido como *Ethereal*, es un analizador de protocolos utilizado para realizar análisis y solucionar problemas en redes de comunicaciones para desarrollo de software y protocolos, y como una herramienta didáctica para educación.

Permite examinar datos de una red viva o de un archivo de captura salvado en disco. Se puede analizar la información capturada, a través de los detalles y sumarios por cada paquete. *Wireshark* incluye un completo lenguaje para filtrar lo que queremos ver y la habilidad de mostrar el flujo reconstruido de una sesión de TCP. Utilizamos *Wireshark* para observar el tráfico de paquetes SIP.

JSR API para JAVA [10,14]

JSR 180 define un paquete opcional que proporciona una API general SIP para clientes J2ME, de modo que las aplicaciones SIP pueden funcionar en dispositivos con memoria limitada. Esta interfaz permite a los dispositivos habilitados para Java móvil enviar y recibir mensajes SIP. El API es compacto, por lo que ocupa poco espacio, y genéricos, así que puedes usarlo con cualquier perfil de J2ME. Como mínimo, se requiere la versión 1.0 del dispositivo conectado Configuración Limitada (*CLDC*), pero la API también se puede utilizar con el dispositivo (*CDC*).

El API se define en el paquete *javax.microedition.sip*. Esta api es el principal paquete que se uso para implementar el cliente IMS.

Algunas de las interfaces más relevantes que utiliza la API son:

- *SipConnection:* La interfaz base para las conexiones SIP, contiene las propiedades y métodos comunes para subinterfaces *SipClientConnection y SipServerConnection.*

- *SipClientConnection:* Representa una transacción de cliente SIP. Una aplicación puede crear un nuevo *SipClientConnection* utilizando *javax.microedition.io.Connector* o *SipDialog*.

- *SipConnectionNotifier*: Define un servidor SIP notificador respecto, que las colas de mensajes entrantes. Para recibir las solicitudes de entrada, las aplicaciones utilizan esta interfaz de *acceptAndOpen()* método, que acepta y abre un nuevo *SipServerConnection*. Si no hay mensajes en la cola, el método se bloquea hasta que una nueva solicitud que se reciba.

- *SipServerConnectionListener*: Una interfaz de escucha para las solicitudes entrantes SIP.

- *SipClientConnectionListener*: Una interfaz de escucha de las respuestas entrantes SIP.

  Las interfaces de la JSR 180 API se muestran en el diagrama de clases [Ilustración 5].

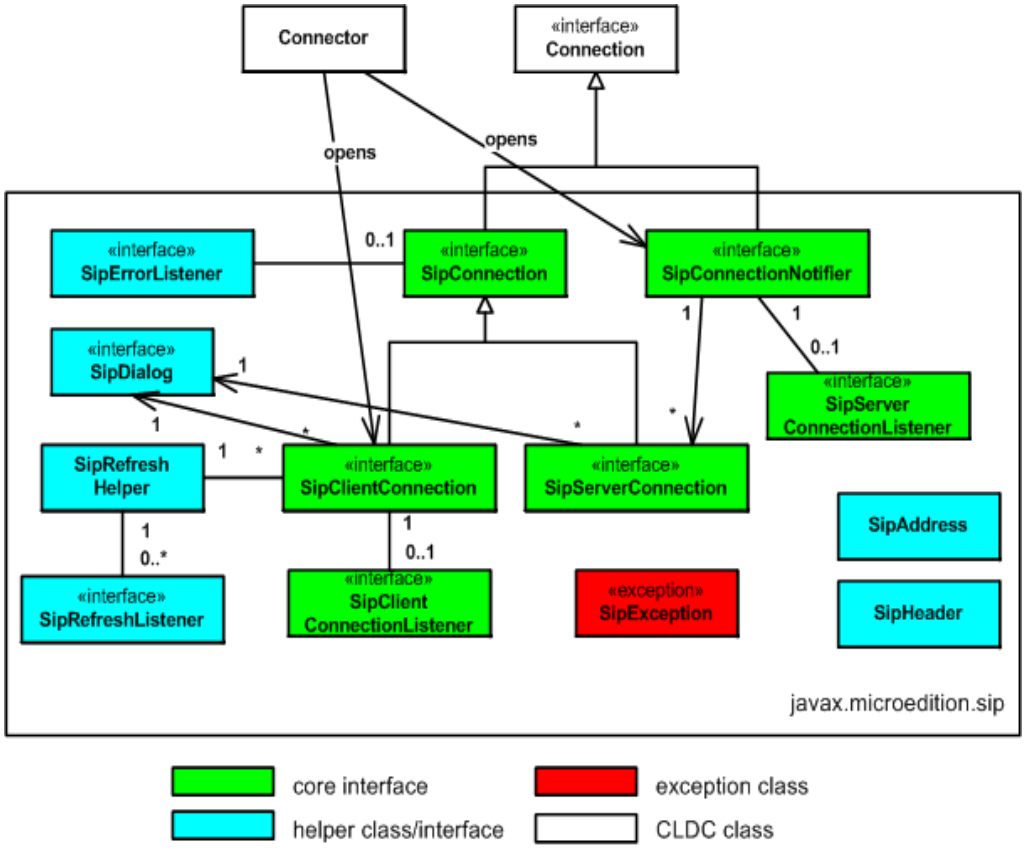

Ilustración 5: Diagrama de Clases SIP API

Protocolo SDP - SIP

El protocolo SDP (Session Description Protocol), se utiliza para describir sesiones multicast en tiempo real, siendo útil para invitaciones, anuncios y cualquier otra forma de inicio de sesiones.

La propuesta original de SDP fue diseñada para anunciar información necesaria para los participantes y para aplicaciones de multicast MBONE (*Multicast Backbone).* Actualmente, su uso está extendido para el anuncio y la negociación de las capacidades de una sesión multimedia en internet.

Puesto que SDP es un protocolo de descripción, los mensajes SDP se pueden transportar mediante distintos protocolos entre ellos SIP. Como el SIP, el SDP utiliza la codificación del texto. Un mensaje de SDP se compone de una serie de líneas, denominados campos, donde los nombres son abreviados por una letra, y está en una orden requerida para simplificar el análisis. SDP no fue diseñado para ser ampliamente extensible.

La única manera de ampliar o agregar nuevas capacidades al SDP es definir un nuevo atributo. Sin embargo, los atributos desconocidos pueden ser ignorados. En el Cuadro 1, podemos observar todos los campos.

Tipo	Descripción Obligatorio	
V	Versión del protocolo (obligatorio)	
o	Identificador (obligatorio)	
S	Nombre de sesión (obligatorio)	
I	Información de la sesión (obligatorio)	
U	URI de la descripción	
e	Dirección de correo	
p	Número de teléfono	
C	Información de conexión	
b	Ancho de banda	
Z	Tiempo de corrección	
K	Clave de encriptación	
a	Atributos	
T	Tiempo de sesión(Start y stop) (obligatorio)	
R	Tiempo de repetición	
m	Información del protocolo de transporte(media)	(obligatorio)

Cuadro 1: Valores de campos SDP

Desarrollo del cliente IMS

En este apartado explicaremos cual fue el proceso de desarrollo del cliente IMS, explicando el proceso de instalación del Dominio IMS, el registro del cliente en el core, el establecimiento de la sesión SIP e intercambio de mensajes entre los clientes.

Dominio IMS

El dominio IMS se compone del núcleo IMS implementado con la herramienta *OpenIMS Core* elaborado por FOKUS. El dominio forma el corazón de la simulación, ya que es quien controla las sesiones y flujos entre los usuarios. La instalación del dominio requiere bajar el código fuente de la herramienta y compilarlo. La instalación y configuración del dominio fue probado exitosamente sobre la versión 10.10 de Ubuntu.

Para instalar el dominio IMS se puede apoyar en los manuales electrónicos encontrados en las siguientes páginas:

1. www.openimscore.org

2. http://uctimsclient.berlios.de/openimscore_on_ubuntu_howto.html

Suponiendo que se logró instalar el dominio IMS exitosamente, es necesario hacer una ligera modificación al script que inicializa el HSS. Con su editor de texto favorito abra el archivo /opt/OpenIMSCore/FHoSS/deploy/startup.sh.

Se necesita inicializar la variable CLASSPATH con el código mostrado en el Cuadro 2 y cambiar la última línea de código a la que muestra abajo:

java -cp $CLASSPATH de.fhg.fokus.hss.main.HSSContainer $1 $2 $3 $4 $5 $6 $7 $8 $9

```
CLASSPATH="/usr/share/java/log4j.jar:/opt/OpenIMSCore/FHoSS/bin/:/usr/
    share/tomcat5.5/server/lib/tomcat-util.jar:/opt/OpenIMSCore/FHoSS/
    tomcat/lib/commons-logging.jar:lib/xml-apis.jar:lib/xercesImpl.jar
    :lib/xerces-2.4.0.jar:lib/xalan-2.4.0.jar:lib/tomcat-util.jar:lib/
    tomcat-http.jar:lib/tomcat-coyote.jar:lib/struts.jar:lib/servlets-
    default.jar:lib/servlet-api.jar:lib/naming-resources.jar:lib/
    naming-factory.jar:lib/mysql-connector-java-3.1.12-bin.jar:lib/
    mx4j-3.0.1.jar:lib/log4j.jar:lib/junit.jar:lib/junitee.jar:lib/jta
    .jar:lib/jsp-api.jar:lib/jmx.jar:lib/jdp.jar:lib/jasper-runtime.
    jar:lib/jasper-compiler-jdt.jar:lib/jasper-compiler.jar:lib/
    hibernate3.jar:lib/FHoSS.jar:lib/ehcache-1.1.jar:lib/dom4j-1.6.1.
    jar:lib/commons-validator.jar:lib/commons-modeler.jar:lib/commons-
    logging.jar:lib/commons-logging-1.0.4.jar:lib/commons-lang.jar:lib
    /commons-fileupload.jar:lib/commons-el.jar:lib/commons-digester.
    jar:lib/commons-collections-3.1.jar:lib/commons-beanutils.jar:lib/
    cglib-2.1.3.jar:lib/catalina-optional.jar:lib/catalina.jar:lib/
    c3p0-0.9.1.jar:lib/base64.jar:lib/asm.jar:lib/asm-attrs.jar:lib/
    antlr-2.7.6.jar"
```

Cuadro 2: Contenido de Variable CLASSPATH

Para probar el funcionamiento correcto de cada uno de los dominios es necesario ejecutar cuatro procesos desde distintas terminales en el mismo sistema como el usuario *root*. Los procesos a ejecutar son los siguientes:

/opt/OpenIMSCore/pcscf.sh

/opt/OpenIMSCore/icscf.sh

/opt/OpenIMSCore/scscf.sh

/opt/OpenIMSCore/FHoSS/deploy/startup.sh

El funcionamiento correcto del dominio se verá reflejado en la información que despliegan estos procesos al momento de comunicarse entre sí. También se podrá hace uso de una herramienta web en el puerto 8080. El Cuadro 3 muestra los puertos que utilizan las funciones del dominio IMS y las ilustraciones 7 y 8 muestran el correcto funcionamiento del core. Es importante mencionar que los dominios IMS ya cuentan con usuarios de prueba llamados *sip:alice@DOMINIO* y *sip:bob@DOMINIO* con las contraseñas *alice* y *bob* respectivamente.

Función	Número de puerto ocupado
P-CSCF	4060
I-CSCF	5060
S-CSCF	6060
HSS (Diameter)	3868, 3869, 3870, 8080

Cuadro 3: Puertos usados por IMS.

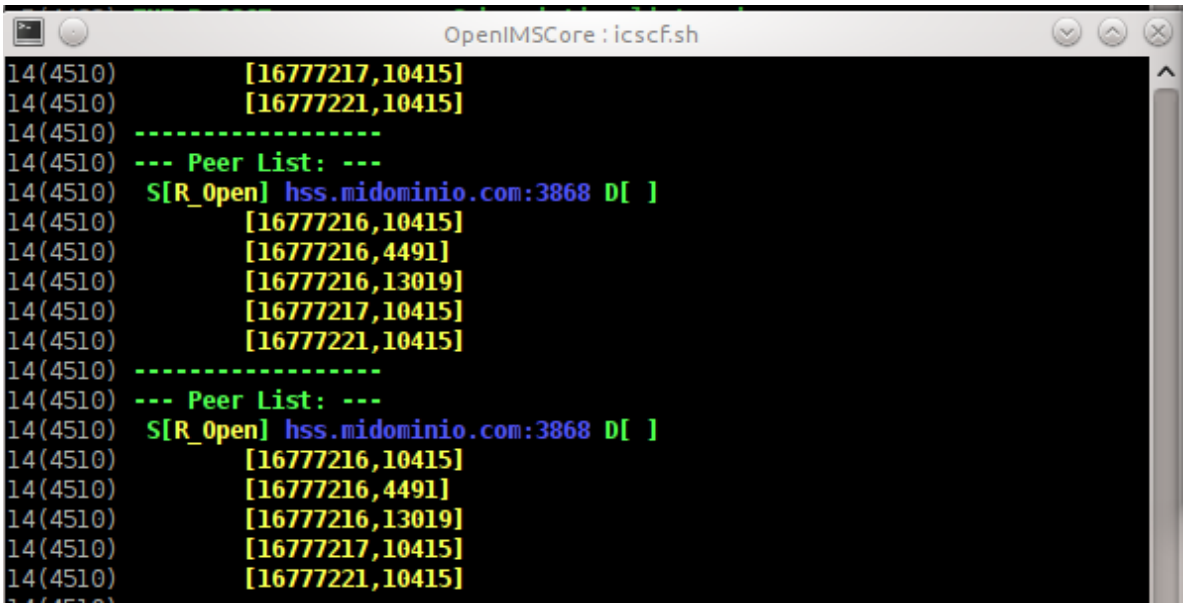

Ilustración 6: Ejecución correcta de icscf.sh

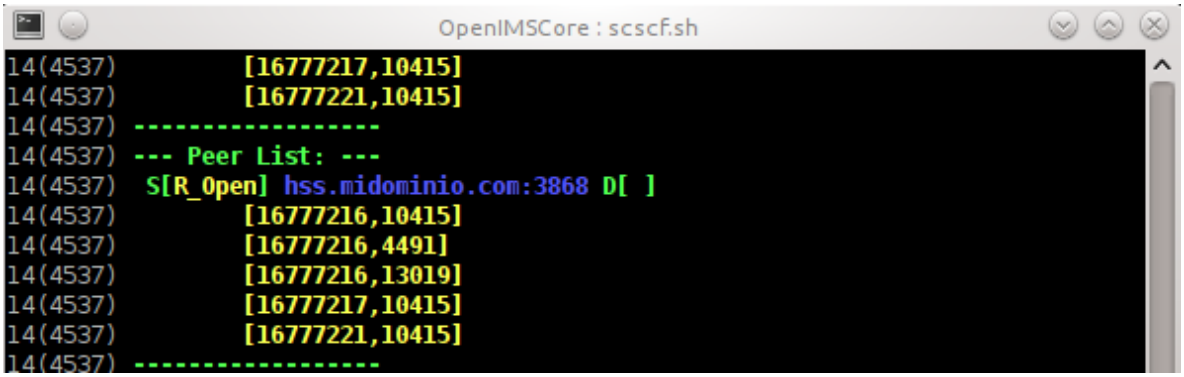

Ilustración 7: Ejecución correcta de scscf.sh

¿Cómo saber si mi Core funciona?

Una vez que el core es iniciado, se necesita saber si en verdad funciona, para probar esto, nos podemos ayudar de un cliente SIP (UCTIMSClient) que podemos descargar de la pagina http://uctimsclient.berlios.de/ , de aquí podemos descargar un archivo deb para Ubuntu/Debian o si bien lo prefieres descargar las fuentes y compilarlas.

Nota: Para la instalación del cliente, seguramente te faltarán resolver varias dependencias, para solucionarlo basta con buscar el paquete incumplido en la página http://packages.ubuntu.com/.

Ya con el cliente instalado, bastara con entrar a la pestaña de preferencias, poner las credenciales correctas y realizar el intercambio de mensajes.

Cliente IMS

El desarrollo del cliente fue basado en las herramientas J2ME, WTK y SIP API ya mencionadas. Las funciones que necesitamos que el cliente realice son las siguientes:

1. Registrarse en el Core.
2. Establecer una sesión.
3. Intercambiar mensajes de texto.
4. Terminar una sesión.

Las peticiones que se espera genere el cliente son mostradas a continuación [Ilustración 9].

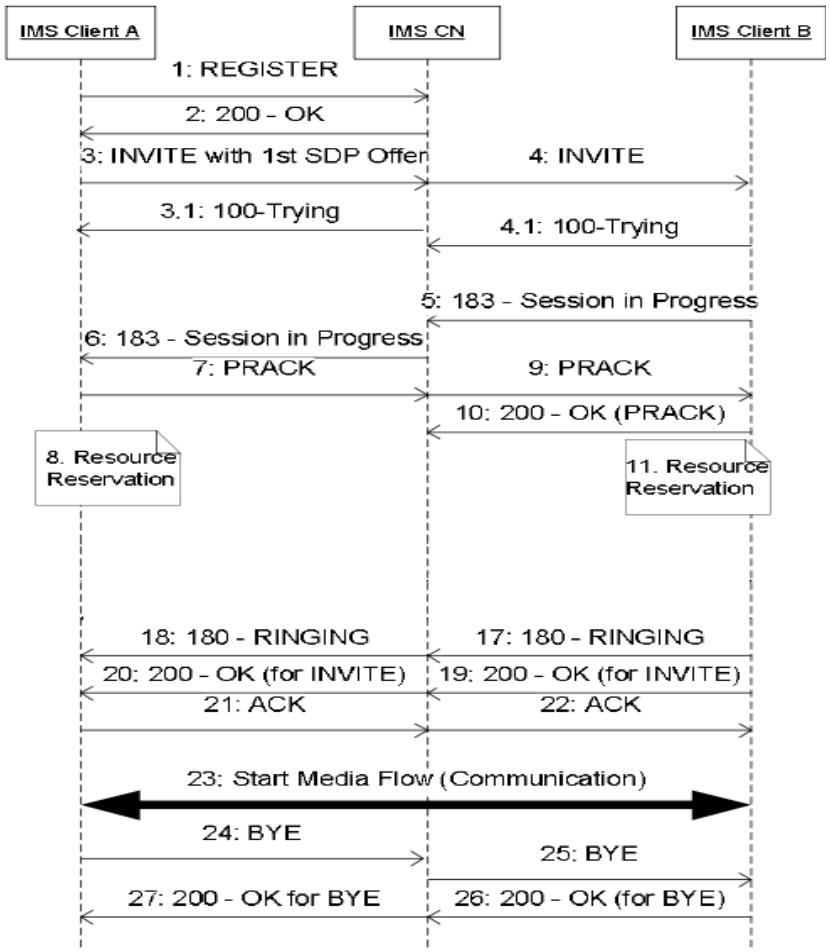

Ilustración 8: Peticiones SIP

Registro SIP

Para que un usuario pueda accesar a sus servicios, primero debe ser registrado en su dominio IMS. El core tiene diferentes métodos de autenticación para soportar múltiples clientes, los cuales pueden verse en la herramienta web del core en el puerto 8080 (IMS Subscription → Search → Serach → Name → IMPIs), ver [Ilustración 10], para nuestro propósito, no necesitamos explorar todos los niveles de seguridad del core por lo que utilizaremos el método de autenticación llamado Early-IMS, el cual se basa en checar que los mensajes provengan de las direcciones IP asignadas a los usuarios, este método se utiliza para dispositivos móviles que no requieren implementar toda la parte de seguridad del core IMS.

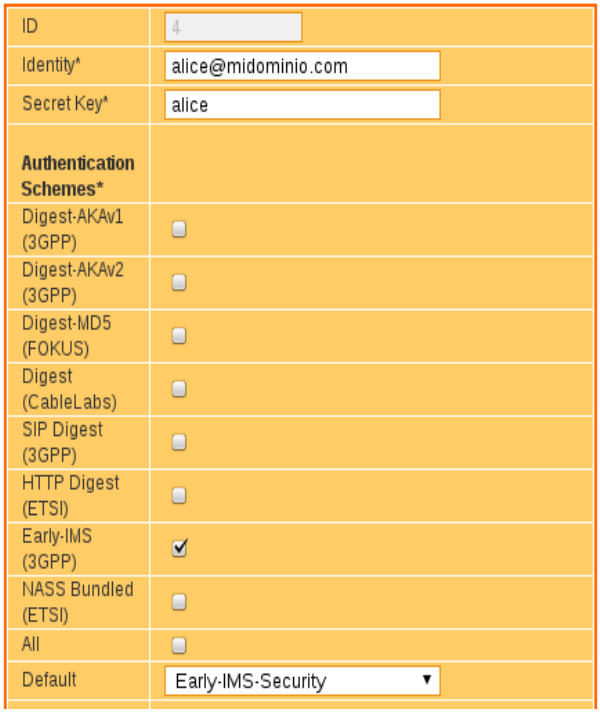

Ilustración 9: Métodos de autenticación del Core

Para poder utilizar este método de autenticación debemos modificar unas líneas en el archivo de configuración del S-CSCF (/opt/OpenIMSCore/scscf.cfg), abrimos el archivo con nuestro editor preferido y lo configuramos como se muestra en la Ilustración 11, asegúrate de que tu archivo quede como se muestra en la imagen.

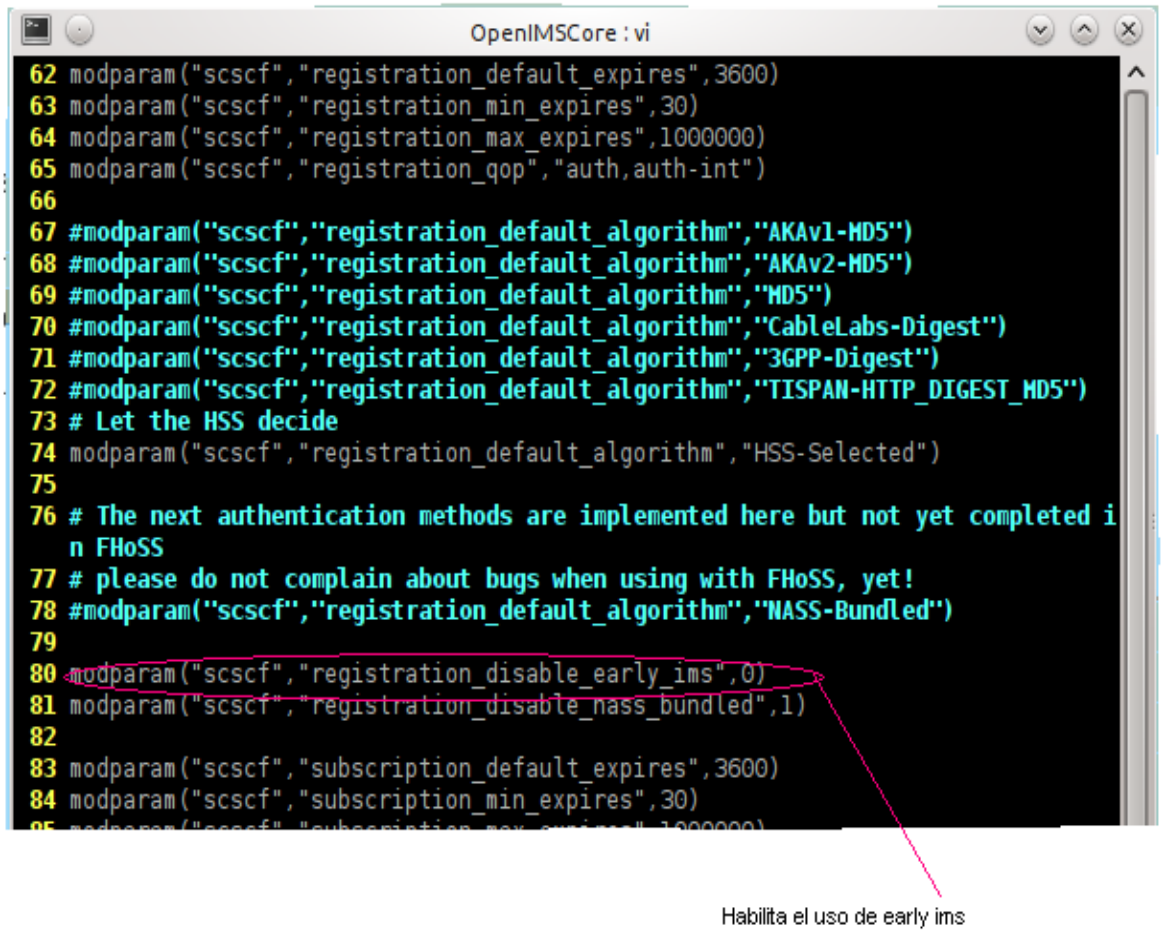

Ilustración 10: Modificación al scscf.cfg

Hecho esto, debemos modificar el perfil de los usuarios, para ello abriremos la herramienta web del core y en la ruta IMS Subscription → Search → Serach → Name → IMPIs, desmarcar todas las casillas de métodos de autenticación y solo dejar Early-IMS, como Early-IMS se basa en checar IP de origen, es de suma importancia asignarle una IP al usuario, ver [Ilustración 12]. Con esto tenemos todo listo para empezar a programar nuestro método de registro con Java SIP.

Nota: como se cambio el método de autenticación, el UCTIMSClient no podrá registrase en el core ya que utiliza otro método de autenticación.

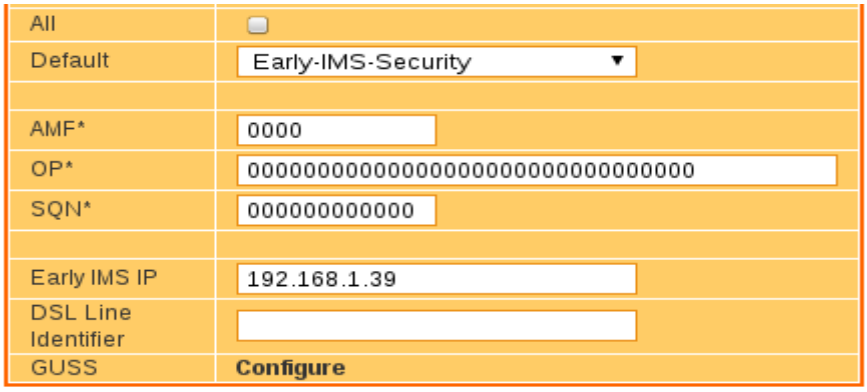

Ilustración 11: Asignación de IP al Usuario

Haciendo uso de SIP API (JSR 180), un SIP REGISTER request es mostrado, así como los headers más importantes.

Nota: si quieres programar un de-register solo basta poner el valor de expires a cero " `cc.setHeader("Expires", "0");` "

```
proxyAddress = new TextField("\nDireccion Proxy: ", "pcscf.open-
ims.test:4060", 27, TextField.ANY);

private SipClientConnection scc = null;

startListener()

 scc = (SipClientConnection) Connector.open("sip:" + proxyAddress);

            scc.setListener(this);

            scc.initRequest("REGISTER", null);

            scc.setHeader("Content-Length", "0");
            scc.setHeader("Max-Forwards", "7");

            scc.setHeader("Require", "sec-agree");
            scc.setHeader("Proxy-Require", "sec-agree");
            scc.setHeader("Expires", "3600");
            scc.setHeader("Private", "none");
            scc.setHeader("Supported", "path");


            scc.setHeader("From", myURI);
            scc.setHeader("To", myURI);
```

```
            scc.setHeader("Contact", contact);

            scc.setHeader("Allow", "INVITE, ACK, CANCEL, BYE, PRACK,
UPDATE, REFER, MESSAGE");
            scc.setHeader("User-Agent", "Sun WTK 2.5 Emulator");

            scc.send();
```

Antes de que un usuario sea registrado debe crearse en el Core (S-CSCF). Si un usuario no es conocido en su dominio, nunca podrá registrase. Después de que se envía el REGISTER, el cliente debe esperar por una respuesta. Un 200-OK significa que el usuario ha sido registrado correctamente.

La Ilustración 13 muestra la pantalla del cliente antes de que sea registrado en el core, aquí debemos de elegir el usuario a registrar y presionar el comando "Registrar" para mandar la petición al core, si el comando "Éxito" es presionado la aplicación será cerrada.

Ilustración 12: Pantalla de Inicio del Cliente IMS

Establecimiento de la sesión SIP.

Después de que hemos sido registrados exitosamente en el core (i.e. recibimos un 200-OK) el cliente muestra una pantalla diferente en donde esperamos mensajes de otra entidad llamada Servidor Back to Back (B2B AS), [Ilustración 14].

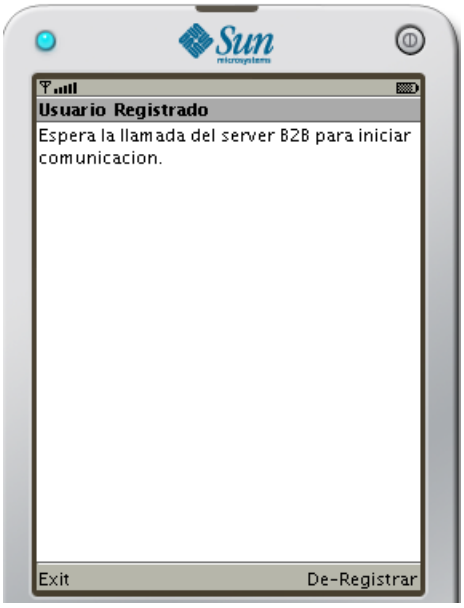

Ilustración 13: Pantalla de espera del Cliente IMS

El servidor Back to Back es el encargado de realizar las llamadas a dos usuarios registrados, se tienen 3 etapas las cuales se muestran en [Ilustración 15].

En la primera etapa el servidor B2B manda un mensaje al usuario 1 (Caller) sin SDP, y el usuario debe contestarle con un 200 ok y su propio SDP.

En la segunda etapa el servidor B2B envía el invite al usuario 2 (Callee), quien debe responderle al server B2B con un 200-OK mas SDP de Callee.

En la tercera etapa el servidor B2B envía un ACK a los dos clientes y abre el flujo para que se puedan intercambiar los mensajes entre los clientes.

Para terminar una sesión, cualquiera de los 2 clientes debe mandar un BYE al B2B quien reenviara la petición a los dos clientes y la sesión será cerrada.

El servidor B2B utilizado puede ser descargado de la siguiente página web: http://uctimsclient.berlios.de/back-2-back_user_agent_howto.html

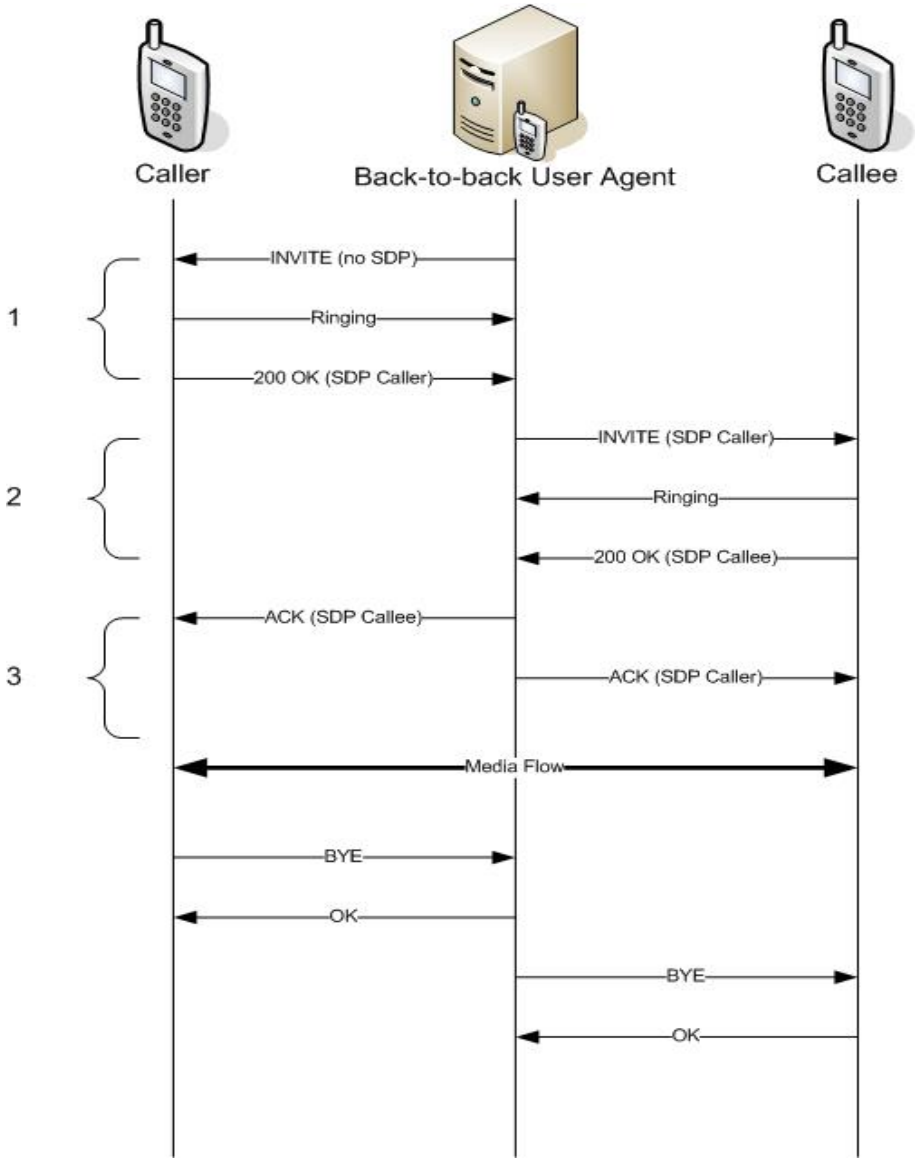

Ilustración 14: Flujo de sesión generado por server B2B

Una vez que todo funciona correcto y el server B2B abre el flujo de sesión, los clientes intercambian mensajes SIP, para mandar el mensaje debemos teclearlo y oprimir el botón “Menu” y después enviar, [Ilustración 16].

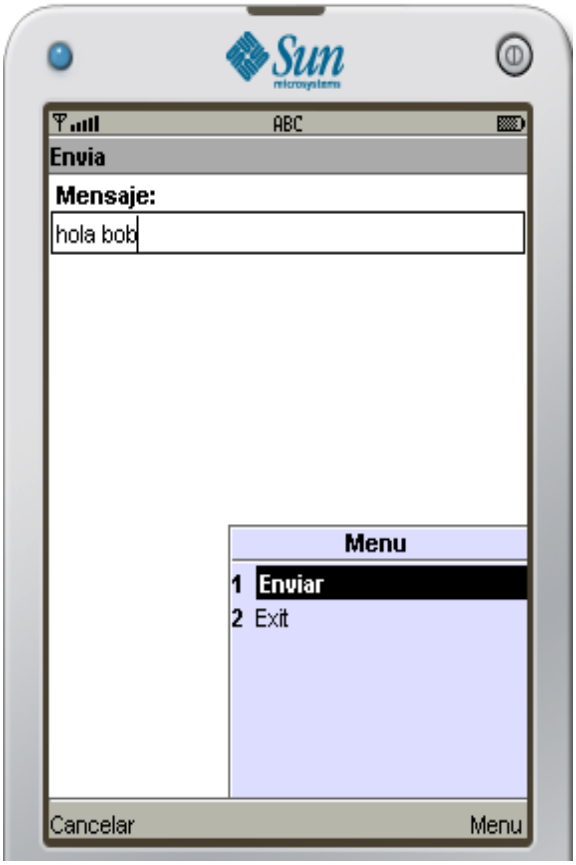

Ilustración 15: Pantalla de envío de mensaje

Para cerrar la sesión solo basta con mandar un BYE, el servidor B2B se encargara de reenviarlo a los 2 clientes y la sesión cera cerrada, ver Ilustración 17.

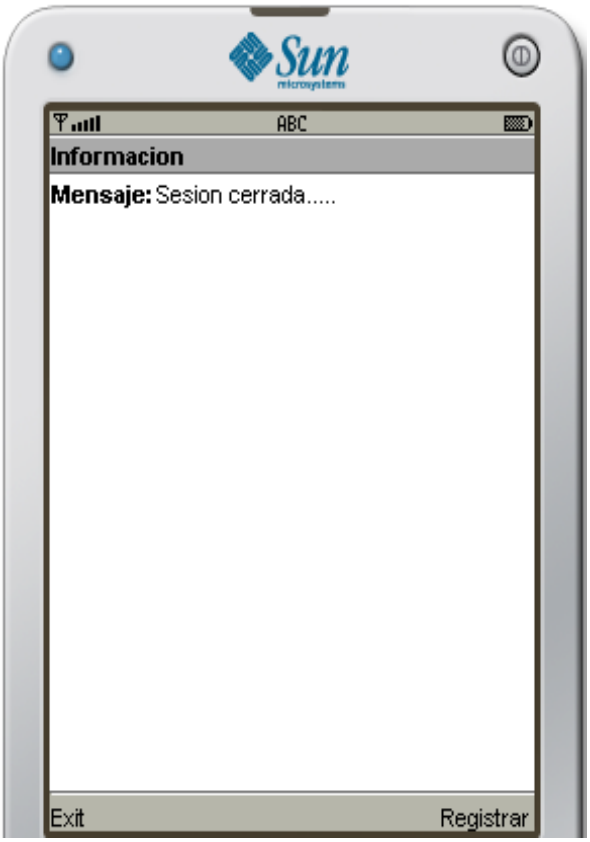

Ilustración 16: Finalización de sesión

Integración del cliente con la red IMS

En este apartado se presentaran los resultados del desarrollo del cliente IMS. Las pruebas fueron hechas con 2 clientes IMS, uno en una máquina con Linux y el otro en una máquina con Windows, el core que autentica a los usuarios, junto con el servidor B2B que nos ayuda a establecer la sesión entre los clientes IMS fue ejecutado en una máquina con Linux Ubuntu 10.10.

Las pruebas se hicieron en una red de área local [Ilustración 18], en donde los componentes estaban basados en direccionamiento IPv4. Se probaron con los usuarios de default, alice y bob.

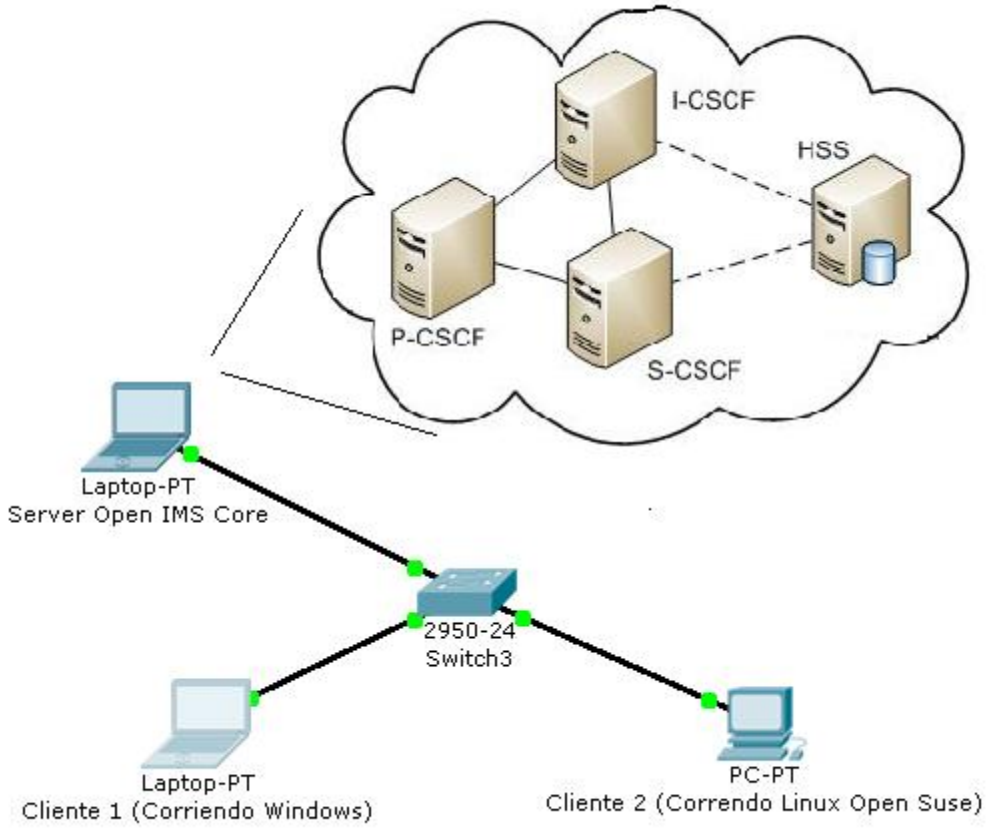

Ilustración 17: Arquitectura básica de prueba

Hardware utilizado

Server Open IMS Core:

- Laptop Acer.
- Procesador Intel Core 2 Duo.
- 320 GB Disco Duro
- 4 GB de Memoria RAM
- SO Linux Ubuntu 10.10

Cliente 1:

- Laptop HP
- Procesador Intel Core 2 Duo
- 320 GB Disco Duro
- 3 GB de Memoria RAM
- SO Windows XP

Cliente 2:

- PC Dell
- Procesador Intel Pentium D
- 180 GB de Disco Duro
- 2 GB de Memoria RAM
- SO Linux Open Suse 11.3

Pruebas y Resultados

La [Ilustración 19] muestra las diferentes pantallas que tiene el cliente durante el establecimiento de la sesión. Al inicio, el usuario elige con cuál de los usuarios desea realizar el registro y al presionar el comando “Registro” se lanza la petición y el cliente queda registrado en el core, una vez que está registrado se muestra una pantalla en donde tenemos que esperar a que el usuario lance en el core al servidor B2B, se realizan las transacciones y si todo esta correcto el cliente desplegara la pantalla en donde podemos introducir los mensajes de texto, para terminar la sesión solo basta con presionar “Cancel” y el servidor B2B se encarga de reenviar la petición y terminar la sesión.

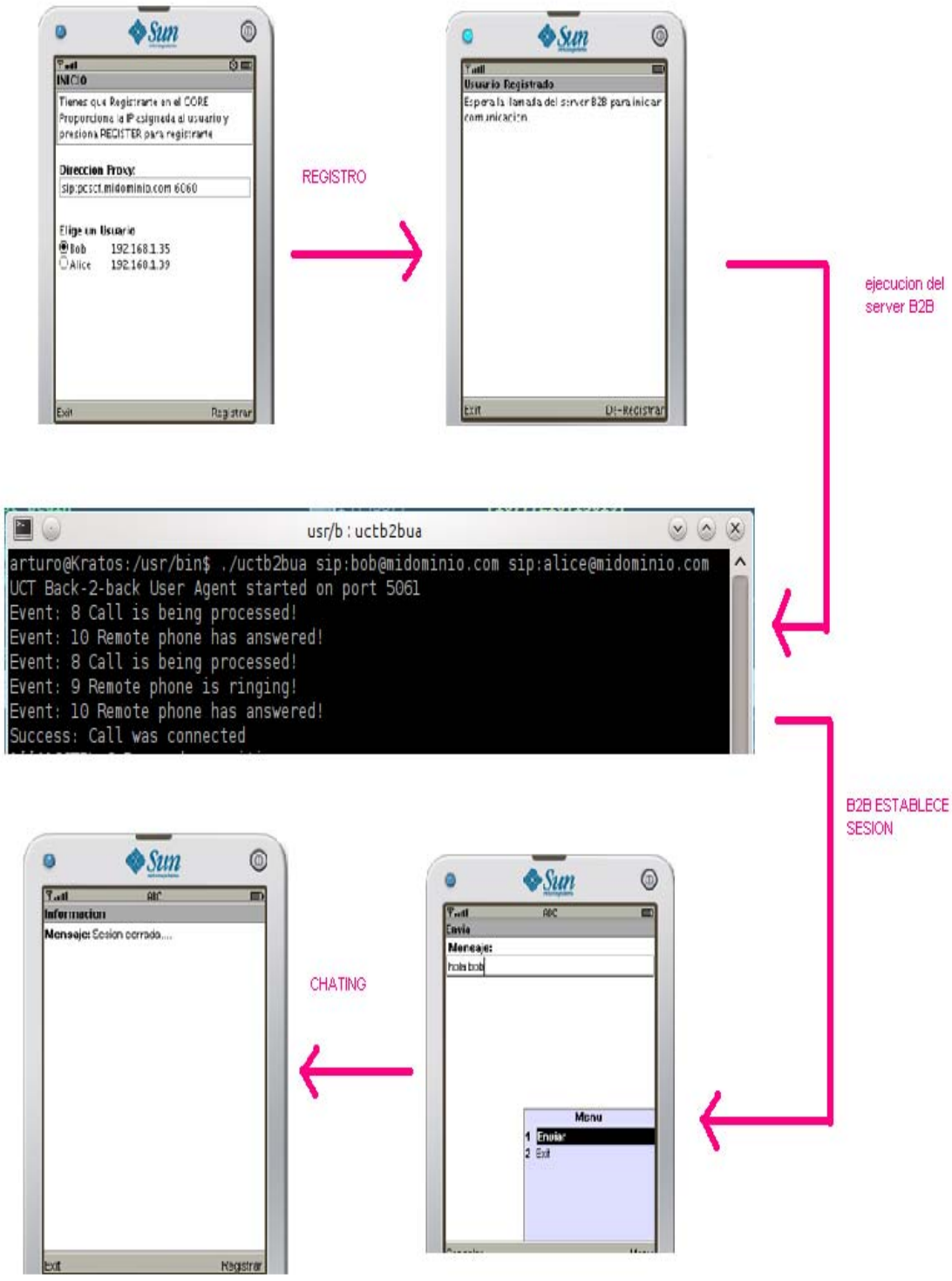

Ilustración 18: Flujo de Pantallas durante una sesión SIP

Petición de Registro

Analizaremos el flujo de paquetes SIP enviados por la petición de registro, el header Contact es de suma importancia ya que nos indica en donde está parado el cliente IMS.

En la Ilustración 20 vemos que se lanzo el método REGISTER y el server Open IMS Core nos respondió con un 200-OK lo que significa que fuimos registrados con éxito.

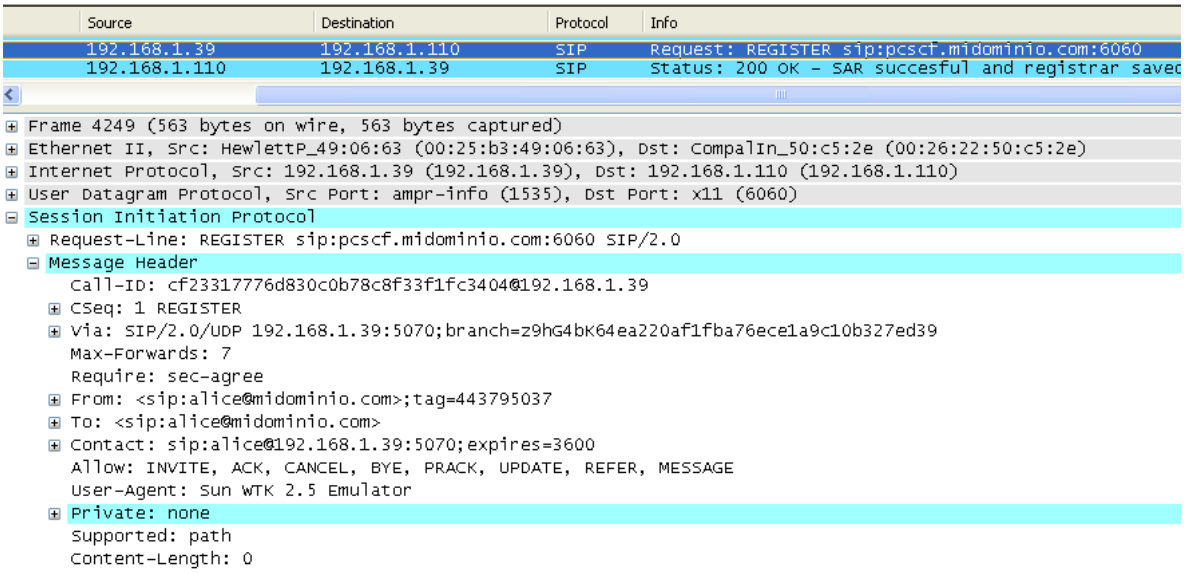

Ilustración 19: Captura SIP del REGISTER

Petición de establecimiento de sesión.

Se analiza el flujo de paquetes SIP enviado por el servidor B2B, claramente se ve que las entidades responden de manera correcta, de este modo el servidor B2B abre la sesión y el flujo para que los clientes intercambien mensajes, [Ilustración 21].

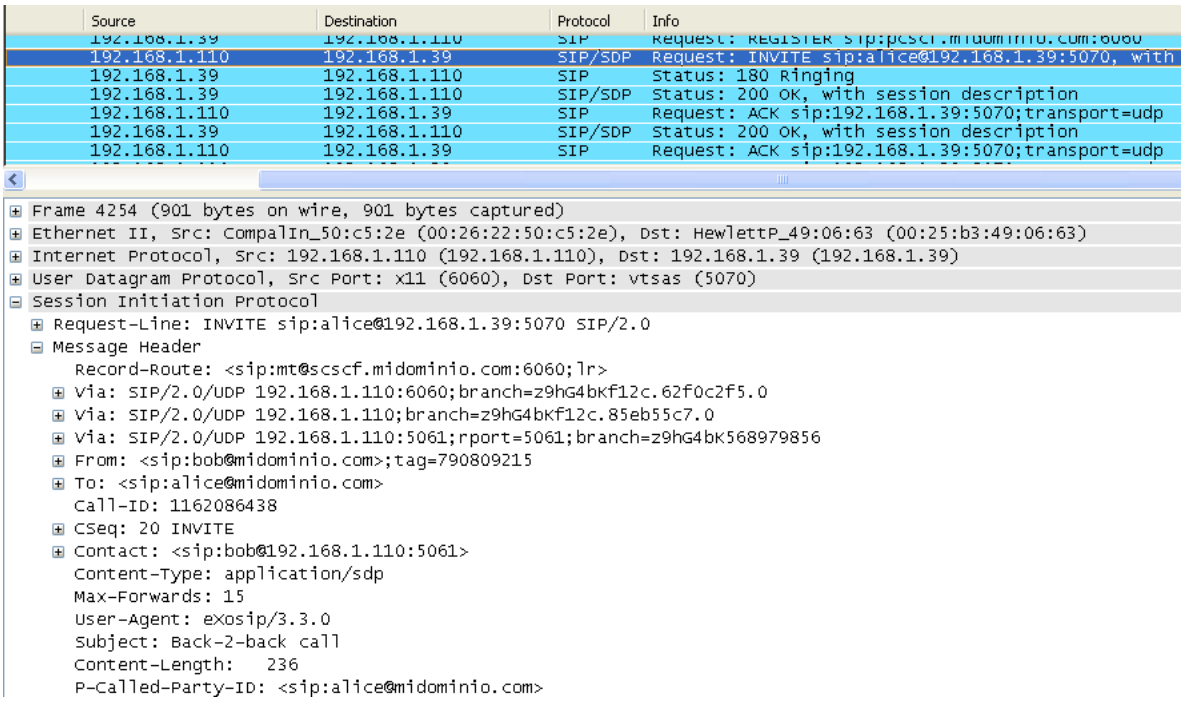

Ilustración 20: Captura SIP del proceso de inicio de sesión

En la [Ilustración 22] se ve como se manda el protocolo SDP anunciando información.

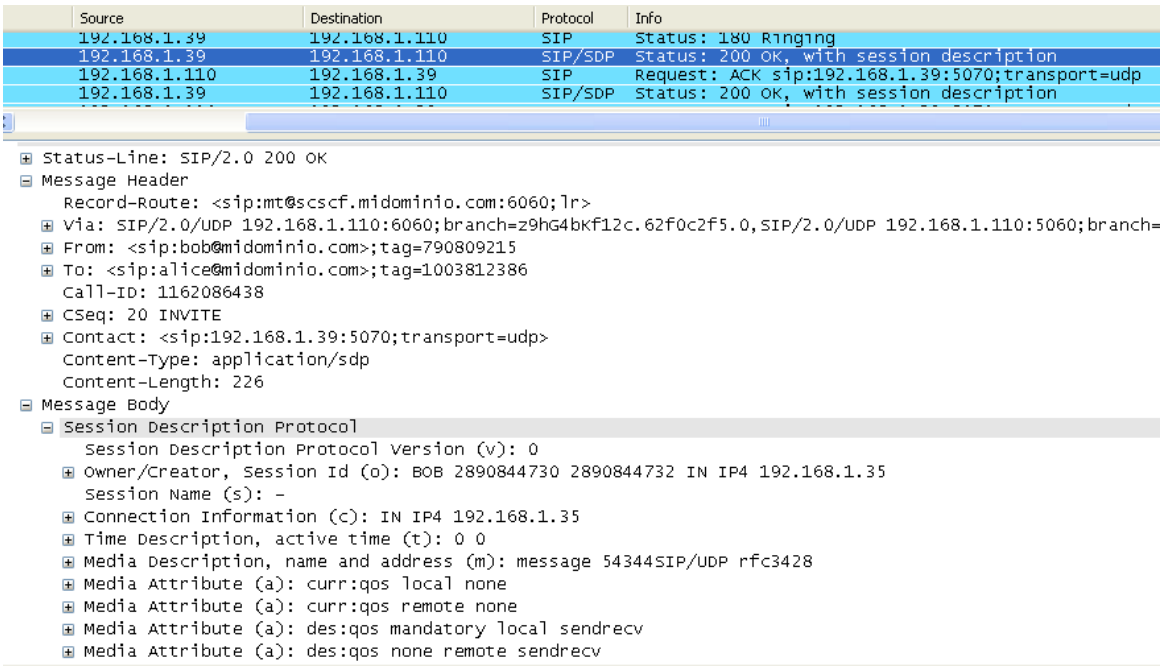

Ilustración 21: Captura de envío de OK con SDP

En la [Ilustración 23] vemos que se lanza la petición MESSAGE la cual hace entrega del mensaje de texto, en este caso el cliente responde con un 200-OK que indica que el mensaje le llego correctamente.

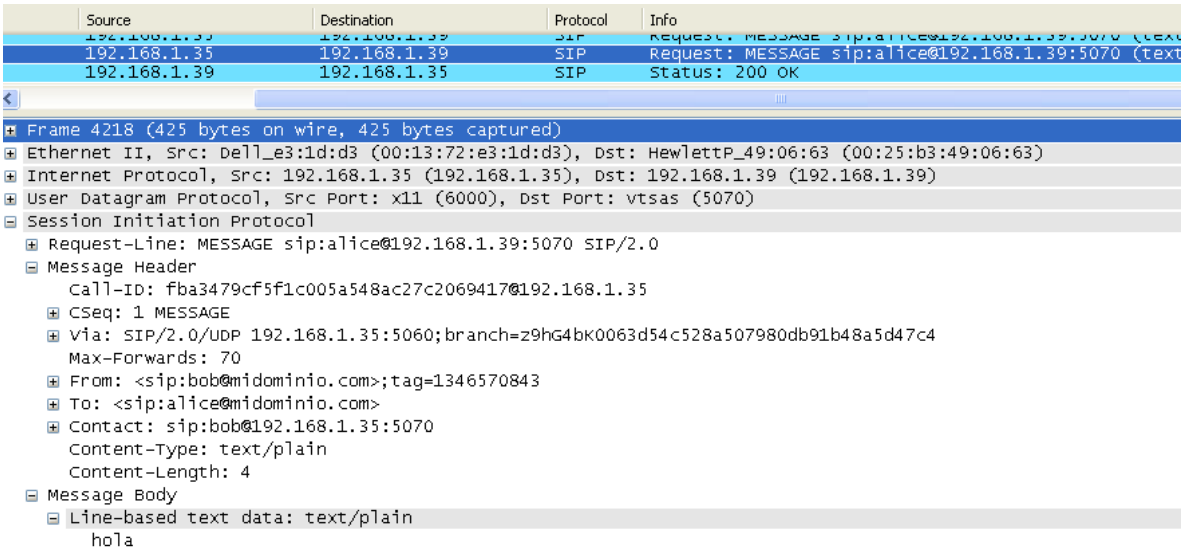

Ilustración 22: Captura del MESSAGE

Problemas encontrados durante el desarrollo

1. Según la documentación de Open IMS Core, la petición de REGISTRO se tiene que hacer a la dirección sip:midominio.com, cuando el midlet java realiza la conexión e intenta abrir la dirección sip:midominio.com le agrega un puerto de destino (5060) quedando sip:midominio.com:5060 , se le pregunto vía mail a las personas que dan soporte a los bugs de Open IMS Core y comentan que el hecho de que la SIP API de java agregue ese puerto de destino hace que el cliente no esté en norma, por lo que no se puede realizar un registro satisfactorio en el proxy del core, nosotros caímos en ésta situación, en donde se intento registrar en el core para obtener la funcionalidad total del mismo sin tener éxito, por eso, es que en este trabajo se tomo la ayuda en de un servidor B2B para poder establecer una sesión entre los dos clientes IMS.

2. El comportamiento del core debido a este problema era extraño ya que se analizo el tráfico de red y se observo que cuando intentaba rutear para hacer la entrega de los paquetes, se hacia un loop y el ttl (*time to live*) del paquete se acababa mandando error de Server Time Out, se les pregunto nuevamente a las personas que dan soporte a los bugs de Open IMS Core

y con su ayuda, analizamos el flujo de paquetes y el comportamiento era como si uno de los módulos del core no se ejecutara.

Conclusiones

Los objetivos de este proyecto terminal se cumplieron exitosamente ya que se pudo transmitir texto plano entre los dos clientes IMS. Las desventajas de trabajar con herramientas FOSS radican es su documentación variada (ésta puede ser buena o mala sin precedente alguno) y en garantizar la estabilidad de sus componentes.

En cambio, las ventajas radican en los siguientes puntos:

- Se obtiene un conocimiento profundo del funcionamiento de sus procesos, permitiendo realizar un mejor diseño al conocer las vulnerabilidades.

- Son herramientas gratuitas que no detienen su desarrollo por medio de licencias.

El desarrollo del cliente y la integración con la red IMS, son un el primer punto de partida para seguir explorando esta tecnología, ya que muy pronto estaremos inmersos en una red de siguiente generación en donde se tenga la total convergencia digital.

A pesar de todos los obstáculos presentes en el desarrollo de los clientes y la instalación del dominio IMS, se puede considerar que el alcance que tuvo este trabajo cumplió las expectativas esperadas. Finalmente, queda mencionar que los clientes IMS quedan a disposición de la UAM para su uso en futuros proyectos.

Líneas Futuras

Esta Sección hace un breve listado de las futuras líneas de trabajo.

1. Darle mayor funcionalidad a los clientes intentando asegurar el registro con la entidad correcta.

2. Migrar todos los componentes a IPv6.

3. hacer que los clientes no sean emulados sino ejecutarlos en equipos reales.

Bibliografía

[1] Miikka Poikselkä and Georg Mayer, The IMS IP Multimedia Concepts and Services, 3th edition, John Wiley & Sons, West Sussex, 2009.

[2] G. Camarillo, and M.A. Garcia-Martin, The 3G Multimedia Subsystem (IMS), Mergin the Internet an the Cellular Worlds, 2nd edition, John Wiley and Sons, Chichester, 2006.

[3] Moshe Timothy Masons, Marc Gird-Genet, Olufemi James Oyedapo and Damien Chatelain, Light Weight IP Multimedia Subsystem (IMS) Client for Mobile Devices, CCECE/CCGEI May

5-7 2008 Niagara Falls. Canadá 978-1-4244-1643-1 IEEE

[4] Alton Kenneth Macdonald Hernandez, Servicios Personalizados de Multimedia Ofrecidos Sobre la Plataforma IP Multimedia Subsystem, ITAM 2009.

[5] Javier Castillo Ruiz, Desarrollo de B2BUA-Filter Criteria para S-CSCF en IMS, Universidad Politécnica de Catalunya España, 2006.

[6] Rodrigo Lizana M, Subsistemas Multimedia IP (IMS) en 3GPP y 3GPP2, Universidad de Santiago de Chile, 2008

[7] The Evolving IPTV Service Architecture, Cisco Systems.

[8] R. S. Cruz, M. S. Nunes, L. Menezes and J. Domingues, SIP Based IPTV Architecture for Heterogeneous Networks.

[9] Manuel Jesus Prieto Martin, Desarrollo de Juegos con J2ME (Java 2 Micro Edition), primera edición, Alfa omega Ra-Ma, México, 2005.

[10] SIP API for Java ME API JSR 180 Version 1.2, Java Community Process/ JSR 180 Expert Group.

[11] http://www.openimscore.org/

[12] http://uctimsclient.berlios.de/

[13] http://www.wireshark.org/

[14] http://developers.sun.com/mobility/apis/articles/sip/

Apéndice

A continuación se muestra el código fuente del cliente IMS.

```
package example.sip;

import java.io.*;
import javax.microedition.io.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.sip.*;

public class ClienteIMS extends MIDlet implements CommandListener,
        SipClientConnectionListener, SipServerConnectionListener {

    //displpayables
    private Display display;
    private Displayable currentDisplay;
    private Displayable backDisplay;
    //variables del radiobutton
    private ChoiceGroup usuario;
    private int seleccion;
    //forms usados
    private Form form;
    protected Form waitScreen = null;
    protected Form failFrm = null;
    protected Form registeredFrm = null;
    protected Form deregisteredFrm = null;
    protected Form proxyFrm = null;
    protected Form ringingFrm = null;
    protected Form sendFrm = null;
    //direccion del proxy
    private TextField proxyAddress;
    //comandos utilizados
    private Command exitCmd = new Command("Exit", Command.EXIT,
1);
    private Command registerCmd = new Command("Registrar",
Command.OK, 1);
    private Command backCmd = new Command("Atras", Command.BACK,
1);
    private Command sendCmd = new Command("Enviar", Command.OK,
1);
    private Command deregisterCmd = new Command("De-Registrar",
Command.OK, 1);
    private Command byeCmd = new Command("Cancelar", Command.BACK,
1);
    //variable del dialog
    private SipDialog dialog;
```

```
    //estados del cliente
    private final short S_OFFLINE = 0;
    private final short S_REGISTERING = 1;
    private final short S_REGISTERED = 2;
    private final short S_DEREGISTERING = 3;
    private final short S_DEREGISTERED = 4;
    private final short S_RINGING = 5;
    private short state = S_OFFLINE;
    //interfaces de conexion sip
    private SipClientConnection scc = null;
    private SipConnectionNotifier scn = null;
    private SipServerConnection ssc = null;
    private SipServerConnection ORIGssc = null;
    //valores para la identidad del usuario
    public String myName = "";
    public String myDomain = "";
    public int mySipPort;
    public String myIp = "";
    public String myURI = "";
    //valores para la identidad del usuario2
    public String friendName = "";
    public String friendDomain = "";
    public int friendSipPort;
    public String friendIp = "";
    public String friendURI = "";
    //variable sdp
    private String sdp = "";

    //determinamos la identidad del usuario1
    public void setUserIdentity(String name, String domain, int
port, String ip) {
        myName = name;
        myDomain = domain;
        mySipPort = port;
        myIp = ip;
        myURI = "sip:" + myName + "@" + myDomain;
    }

    //determinamos la identidad del usuario2
    public void setFriendIdentity(String name, String domain, int
port, String ip) {
        friendName = name;
        friendDomain = domain;
        friendSipPort = port;
        friendIp = ip;
        friendURI = "sip:" + friendName + "@" + friendDomain;
    }

    //formulario inicial
```

```
    private Form getProxyFrm() {
        if (proxyFrm == null) {
            proxyFrm = new Form("INICIO");

            proxyFrm.append("Tienes que Registrarte en el CORE\n"
                    + "Proporciona la IP asignada al usuario y
presiona REGISTER para registrarte\n");

            //se muestra la direccion del proxy
            proxyAddress = new TextField("\nDireccion Proxy: ",
"sip:pcscf.midominio.com:6060", 50, TextField.ANY);
            proxyFrm.append(proxyAddress);

            proxyFrm.append(" \n");

            //generamos la seleccion de los usuarios
            usuario = new ChoiceGroup("Elige un Usuario",
ChoiceGroup.EXCLUSIVE);
            usuario.append("Bob          192.168.1.35", null);
            usuario.append("Alice        192.168.1.39", null);
            proxyFrm.append(usuario);
            seleccion = usuario.getSelectedIndex();

            if (seleccion == 0) {// si es bob
                //tomamos la identidad
                setUserIdentity("bob", "midominio.com", 5070,
"192.168.1.35");
                setFriendIdentity("alice", "midominio.com", 5070,
"192.168.1.39");

                sdp = "v=0\n"
                        + "o=" + myName.toUpperCase() + "
2890844730 2890844732 IN IP4 " + myIp + "\n" + ""
                        + "s=-\n"
                        + "c=IN IP4 " + myIp + "\n"
                        + "t=0 0\n"
                        + "m=message 54344" + "SIP/UDP rfc3428\n"
                        + "a=curr:qos local none\n"
                        + "a=curr:qos remote none\n"
                        + "a=des:qos mandatory local sendrecv\n"
                        + "a=des:qos none remote sendrecv\n";

            } else if (seleccion == 1) {//si es alice
                //tomamos la identidad
                setUserIdentity("alice", "midominio.com", 5070,
"192.168.1.39");
                setFriendIdentity("bob", "midominio.com", 5070,
"192.168.1.35");
                sdp = "v=0\n"
```

```
                        + "o=" + friendName.toUpperCase() + "
2890844730 2890844732 IN IP4 " + friendIp + "\n" + ""
                        + "s=-\n"
                        + "c=IN IP4 " + friendIp + "\n"
                        + "t=0 0\n"
                        + "m=message 54344" + "SIP/UDP rfc3428\n"
                        + "a=curr:qos local none\n"
                        + "a=curr:qos remote none\n"
                        + "a=des:qos mandatory local sendrecv\n"
                        + "a=des:qos none remote sendrecv\n";
            }

            //agrega los comandos y escucha
            proxyFrm.addCommand(registerCmd);
            proxyFrm.addCommand(exitCmd);
            proxyFrm.setCommandListener(this);
        }

        return proxyFrm;
    }

    //formulario que se muestra cuando hay un error
    private Form getFailFrm(String msg) {
        if (failFrm == null) {
            failFrm = new Form("Error");
            failFrm.addCommand(exitCmd);
            failFrm.setCommandListener(this);
            failFrm.append(new StringItem("", ""));
        }
        StringItem si = (StringItem) failFrm.get(0);
        si.setText(msg);

        return failFrm;
    }

    //pantalla que se muestra cuando esta sonando
    private Form getRingingFrm(String message) {
        if (ringingFrm == null) {
            ringingFrm = new Form("Ringing ...");
            ringingFrm.append(new StringItem("Mensaje:",
message));
            //ringingFrm.addCommand(denyCmd);
            //ringingFrm.addCommand(answerCmd);
            ringingFrm.setCommandListener(this);
        }

        StringItem si = (StringItem) ringingFrm.get(0);
        si.setText(message);
```

```
        return ringingFrm;
    }

    //pantalla mostrada al registrar al usuario
    private Form getRegisteredFrm() {
        if (registeredFrm == null) {
            registeredFrm = new Form("Usuario Registrado",
                    new Item[]{
                        new StringItem(null, "Espera la llamada
del server B2B "
                        + "para iniciar comunicacion.")
                    });
            registeredFrm.addCommand(exitCmd);
            registeredFrm.addCommand(deregisterCmd);
            registeredFrm.setCommandListener(this);
        }
        return registeredFrm;
    }

    //pantalla qque se muestra cuando el usario no esta registrado
    private Form getDeRegisteredFrm(String message) {
        if (deregisteredFrm == null) {
            deregisteredFrm = new Form("Informacion");
            deregisteredFrm.append(new StringItem("Mensaje:",
message));

            //agrega comandos al form
            deregisteredFrm.addCommand(registerCmd);
            deregisteredFrm.addCommand(exitCmd);

            deregisteredFrm.setCommandListener(this);
        }
        StringItem si = (StringItem) deregisteredFrm.get(0);
        si.setText(message);
        return deregisteredFrm;
    }

    //mensaje que se muestra alchatear
    private Form getSendForm() {
        if (sendFrm == null) {
            sendFrm = new Form("Envia",
                    new Item[]{new TextField(" Mensaje:", "", 255,
TextField.ANY)});
            sendFrm.addCommand(sendCmd);
            sendFrm.addCommand(byeCmd);
            sendFrm.addCommand(exitCmd);
            sendFrm.setCommandListener(this);
        }
```

```
        TextField tfield = (TextField) sendFrm.get(0);

        if (tfield != null) {
            tfield.setString("");
        }

        return sendFrm;
    }

    //toma los eventos de los comandos
    public void commandAction(Command c, Displayable d) {
        if ((c == registerCmd) && (d == proxyFrm)) {
            Thread t = new Thread() {//si no esta en un hilo falla
la ejecucion

                public void run() {
                    register();//hace el registro
                }
            };
            t.start();
            return;
        } else if (c == backCmd && (d == failFrm)) {
            setDisplay(getProxyFrm());
        } else if (c == exitCmd && (d == failFrm)) {
            destroyApp(false);
        } else if (c == deregisterCmd && (d == registeredFrm)) {
            Thread t = new Thread() {

                public void run() {
                    deregister();//hace el deregistro
                }
            };
            t.start();
            return;
        } else if (c == exitCmd && (d == deregisteredFrm)) {
            destroyApp(false);
        } else if (c == registerCmd && (d == registeredFrm)) {
            setDisplay(getProxyFrm());
        } else if (c == sendCmd && (d == sendFrm)) {
            Thread t = new Thread() {

                public void run() {
                    //caja de texto de la pantalla de chateo
                    TextField txtField = (TextField)
sendFrm.get(0);
                    //pone el texto de la caja en el mensaje
                    if ((txtField != null) &&
(txtField.getString().length() > 0)) {
                        sendMessage(txtField.getString());//se
```

```
manda el mensaje
                        txtField.setString("");//se limpia la caja
de texto
                    }
                }
            };
            t.start();
            return;

        } else if (c == byeCmd && (d == sendFrm)) {
            sendBYE();//manda un bye
        } else if (c == exitCmd && (d == sendFrm)) {
            destroyApp(false);//sale de la aplicacion
        }
    }

    //metodo que sirve para mostrar la pantalla que necesitamos
    private void setDisplay(Displayable d) {
        if (currentDisplay != waitScreen) {
            backDisplay = currentDisplay;
        }

        display.setCurrent(d);
        currentDisplay = d;
    }

    public void startApp() {
        display = Display.getDisplay(this);
        //llamamos al la pantalla inicial
        setDisplay(getProxyFrm());
    }

    //metodo que realiza el registro
    private void register() {
        try {
            state = S_REGISTERING;

            startListener();//pongo en escucha en el puerto
indicado
            // envio la peticion al proxy del core
            scc = (SipClientConnection)
Connector.open(proxyAddress.getString());
            scc.setListener(this);

            // build the contact URI
            String contact = "sip:" + myName + "@" +
scn.getLocalAddress()
                    + ":" + scn.getLocalPort() + ";expires=3600";
```

```
            // initialize INVITE request agregando los headers
            scc.initRequest("REGISTER", scn);

            scc.setHeader("Max-Forwards", "7");
            scc.setHeader("Require", "sec-agree");


            scc.setHeader("From", myURI);
            scc.setHeader("To", myURI);

            scc.setHeader("Contact", contact);
            scc.setHeader("Allow", "INVITE, ACK, CANCEL, BYE,
PRACK, UPDATE, REFER, MESSAGE");

            scc.setHeader("User-Agent", "Sun WTK 2.5 Emulator");
            scc.setHeader("Private", "none");
            scc.setHeader("Content-Length", "0");

            scc.setHeader("Supported", "path");

            scc.send();//envia el register

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    //metodo que derregistra a un usuario
    private void deregister() {
        try {
            state = S_DEREGISTERING;
            startListener();//pongo en escucha en el puerto
indicado
            // envio la peticion al proxy del core
            scc = (SipClientConnection)
Connector.open(proxyAddress.getString());
            scc.setListener(this);

            // build the contact URI
            String contact = "sip:" + myName + "@" +
scn.getLocalAddress()
                    + ":" + scn.getLocalPort();

            // initialize INVITE request
            scc.initRequest("REGISTER", scn);

            scc.setHeader("Max-Forwards", "7");
            scc.setHeader("Require", "sec-agree");
            scc.setHeader("Expires", "0");//valor clave para el
```

```
deregistro

            scc.setHeader("From", myURI);
            scc.setHeader("To", myURI);

            scc.setHeader("Contact", contact);
            scc.setHeader("Allow", "INVITE, ACK, CANCEL, BYE,
PRACK, UPDATE, REFER, MESSAGE");

            scc.setHeader("User-Agent", "Sun WTK 2.5 Emulator");
            scc.setHeader("Private", "none");
            scc.setHeader("Content-Length", "0");

            scc.setHeader("Supported", "path");

            scc.send();//envia el derregister

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    //metodo que envia el mensaje
    private void sendMessage(String message) {
        SipClientConnection cn1 = null;
        OutputStream output = null;
        try {
            // construye el contact URI
            String contact = "sip:" + myName + "@" +
scn.getLocalAddress()
                    + ":" + scn.getLocalPort();

            startListener();//pone en escucha en el puerto
            cn1 = (SipClientConnection)
Connector.open("sip:alice@192.168.1.39:5070");
            cn1.setListener(this);

            cn1.initRequest("MESSAGE", null);
            cn1.setHeader("Content-Type", "text/plain");
            cn1.setHeader("Content-Length",
Integer.toString(message.length()));
            cn1.setHeader("From", "sip:bob@midominio.com");
            cn1.setHeader("To", "sip:alice@midominio.com");
            cn1.setHeader("Contact", contact);

            //manda el contenido a la red
            output = cn1.openContentOutputStream();
            output.write(message.getBytes());
            output.close();
```

```
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    private void sendBYE() {
        try {
            scc = dialog.getNewClientConnection("BYE");
            scc.send();//envia el bye

            boolean gotit = scc.receive(10000);//espera una
respuesta
            if (gotit) {
                if (scc.getStatusCode() == 200) {//si le responden
que si sale de sesion
                    setDisplay(getDeRegisteredFrm("Sesion
cerrada....."));
                } else {
                    setDisplay(getFailFrm("Error " +
scc.getReasonPhrase()));
                }
            }

        } catch (Exception ex) {
            ex.printStackTrace();
        }

    }

    public void notifyResponse(SipClientConnection scc) {
        try {
            boolean ok = scc.receive(1000);//escucha que metodos
llegan

            if (!ok) {
                //si no se recibe respuesta manda pantalla de
error
                setDisplay(getFailFrm("Respuesta no recibida"));
            }

            int code = scc.getStatusCode();
            System.out.println("Estatus recibido: " + code);

            if (state == S_REGISTERING) {//si la bandera es
registrada
                if (code == 200) {//respuesta es correcta
                    System.out.println("Received OK after
REGISTER");
```

```
                    state = S_REGISTERED;
                    setDisplay(getRegisteredFrm());//muestra la
pantalla de registrado

                } else {//si falla el registro muestra pantalla de
error
                    System.out.println("Respuesta no conocida " +
scc.getStatusCode() + scc.getReasonPhrase());
                    setDisplay(getFailFrm("Respuesta no conocida "
+ scc.getStatusCode() + scc.getReasonPhrase()));
                }
            } else if (state == S_DEREGISTERING) {//si la bandera
es deregistrada
                if (code == 200) {//respuesta es correcta
                    System.out.println("Received OK after
REGISTER");
                    state = S_DEREGISTERED;
                    //muestra la pantalla de deregistrado
                    setDisplay(getDeRegisteredFrm("Usuario De-
registrado con exito"
                            + " presiona \"Registrar\" para
iniciar comunicacion."));

                } else {//si falla el registro muestra pantalla de
error
                    System.out.println("Respuesta no conocida " +
scc.getStatusCode() + scc.getReasonPhrase());
                    setDisplay(getFailFrm("Respuesta no conocida "
+ scc.getStatusCode() + scc.getReasonPhrase()));
                }
            }

        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    public void notifyRequest(SipConnectionNotifier scn) {
        try {
            ssc = scn.acceptAndOpen(); // blocking

            if (ssc.getMethod().equals("INVITE")) {
                ORIGssc = ssc;
                state = S_RINGING;

                String contentLength = ssc.getHeader("Content-
Length");//checa el tamaÃ±o
                String subject = ssc.getHeader("Subject");//extrae
contenido subject
```

```
                String contact = ssc.getHeader("Contact");
                int length = Integer.parseInt(contentLength);
                if (length == 0) {//si es vacio
                    setDisplay(getRingingFrm(subject));
//despliega sig pantalla

                    ssc.initResponse(200);//setea ok


                    //System.out.println("el sdp es: "+sdp);
                    ssc.setHeader("Content-Length", "" +
sdp.length());//anuncia el tamaÃ±o del sdp
                    ssc.setHeader("Content-Type",
"application/sdp");//anuncia el tipo
                    OutputStream os =
ssc.openContentOutputStream();
                    os.write(sdp.getBytes());//manda el sdp a la
red
                    os.close(); // close and send

                    ssc.send();//manda ok
                    //guarda el dialog
                    dialog = ssc.getDialog();
                    ringingFrm.append("\nEstado del dialogo:" +
dialog.getState());
                    ringingFrm.append("\n\nEsperando ACK.......");
                    ssc.close();
                }
            } else if (ssc.getMethod().equals("ACK")) {
                System.out.println("sesion establecida " +
ssc.getHeader("CALL-ID"));
                setDisplay(getSendForm());//muestra la pantalla en
donde se manda msj
                System.out.println("Dialog state: " +
dialog.getState() + "\n");
                //dialog = scc.getDialog();
                ssc.close();

            } else if (ssc.getMethod().equals("BYE")) {
                ssc.initResponse(200);
                ssc.send();
                //muestra la pantalla de deregistrado
                setDisplay(getDeRegisteredFrm("Sesion cerrada"));
                ssc.close();

            } else if (ssc.getMethod().equals("MESSAGE")) {
                System.out.println("se ha recibido mensaje");

                //checa el contenido y el tipo del mensaje
```

```
                String contentType = ssc.getHeader("Content-
Type");
                String contentLength = ssc.getHeader("Content-
Length");
                int length = Integer.parseInt(contentLength);
                if ((contentType != null) &&
contentType.equals("text/plain")) {
                    InputStream is = ssc.openContentInputStream();
                    int i = 0;
                    byte testBuffer[] = new byte[length];
                    i = is.read(testBuffer);//lee lo que esta en
el buffer

                    String temp = new String(testBuffer, 0,
i);//almacena el contenido
                    StringItem st = new
StringItem(friendName.toUpperCase() + " dice: ", temp);

                    sendFrm.append(st);//imprime en pantalla


                }
                ssc.initResponse(200);
                ssc.send();


            } else {//si falla el registro muestra pantalla de
error
                setDisplay(getFailFrm("Metodo no conocido " +
scc.getMethod()));
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    public void pauseApp() {
// pause
    }

    public void destroyApp(boolean b) {
        notifyDestroyed();
    }

    public void shutdown() {
        destroyApp(false);
    }

    //inicio la escucha en el puerto
```

```
    private void startListener() {
        try {
            if (scn != null) {
                scn.close();
            }
// escucha en el puerto dado
            scn = (SipConnectionNotifier)
Connector.open("sip:5070");
            scn.setListener(this);
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    //paramos la escucha en el puerto
    private void stopListener() {
        try {
            if (scn != null) {
                scn.close();
            }
            scn = null;
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```