CLIENTE MÓVIL SIP PARA UN SISTEMA IVR BASADO EN LA ARQUITECTURA DE SERVICIOS IMS

JOSÉ FERNANDO MUÑOZ BERMEO
XIMENA VELASCO MELO

ANEXO B

ENTORNO DE DESARROLLO Y HERRAMIENTAS DE PROGRAMACIÓN PARA CLIENTES MÓVILES SIP

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telemática
Popayán
2006

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

B1. INTRODUCCIÓN

El presente anexo tiene la finalidad de exponer las tecnologías y herramientas que se utilizaron para implementar el cliente móvil SIP con capacidad de ejecución de aplicaciones que acceden a servicios IVR proveídos dentro de la arquitectura de servicio IMS.

Para el proceso de selección de las diferentes tecnologías y herramientas, se tuvo en cuenta que: permitiera el desarrollo de los requerimientos planteados, se tratara de software (SW) libre, se ejecutara en Windows XP, contara con una buena documentación de instalación y uso, facilitara ejemplos de aplicaciones previamente realizadas, utilizaran lenguajes de programación y metodologías de desarrollo para dispositivos móviles ya conocidas y en las cuales se tenga experiencia. Teniendo en cuenta este último parámetro, se seleccionó el lenguaje de programación Java para dispositivos móviles, es decir J2ME.

En la sección B2 se habla acerca del entorno de desarrollo Eclipse, en la sección B3 se detallan algunas herramientas de programación para dispositivos móviles, y en la sección B4 se habla acerca del API de SIP para J2ME.

B2. ENTORNO DE DESARROLLO: ECLIPSE [1]

Eclipse es un Entorno de Desarrollo Integrado (Integrated Development Enviroment, IDE) multiplataforma y de código abierto, que sirve para crear aplicaciones clientes de cualquier tipo.

El proyecto Eclipse se divide en tres subproyectos:

- El Núcleo de la aplicación, que incluye el subsistema de ayuda, la plataforma para trabajo colaborativo, el Área de Trabajo (Workbench) y el Espacio de Trabajo (Workspace) para gestionar proyectos.
- El conjunto de herramientas para el desarrollo Java (Java Development Toolkit, JDT).
- El Módulo de Entorno de Desarrollo (Plug-in Development Enviroment, PDE), que proporciona las herramientas para el desarrollo de nuevos módulos.

Eclipse fue creado originalmente por IBM. Ahora lo desarrolla la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

El IDE Eclipse emplea módulos para proporcionar toda su funcionalidad, a diferencia de otros entornos compactos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos permite que el entorno de desarrollo soporte otros lenguajes además de Java. Por ejemplo, existe un módulo para dar soporte a C/C++, para el desarrollo de aplicaciones para el sistema operativo Symbian.

En cuanto a las aplicaciones clientes, Eclipse provee al programador con frameworks muy ricos para el desarrollo de aplicaciones gráficas, definición y manipulación de modelos de software, aplicaciones Web, etc.

Características

- ✓ Editor de texto
- ✓ Resaltado de sintaxis
- ✓ Compilación en tiempo real
- ✓ Pruebas unitarias con JUnit
- ✓ Control de versiones con CVS
- ✓ Integración con la herramienta de construcción de Jakarta: Ant
- ✓ Asistentes: para creación de proyectos, clases, pruebas, etc.
- ✓ Refactorización de código
- ✓ Actualización/instalación automática de código (mediante Update Manager)
- ✓ Con respecto a las plataformas, Eclipse proporciona instalaciones binarias para Windows, Linux, Solaris, HP-UX, AIX, QNX, y Mac OS X.

Asimismo, a través de plug-ins, libremente disponibles, es posible añadir:

- ✓ Control de versiones con Subversion, vía Subclipse.
- ✓ Integración con Hibernate, vía Hibernate Tools.

EclipseME es un plugin para Eclipse que facilita la integración de los distintos componentes de desarrollo J2ME con este IDE, para de esta forma aprovechar la gestión de proyectos, herramientas de desarrollo y depuración dentro de emuladores J2ME.

B3. HERRAMIENTAS DE PROGRAMACIÓN PARA DISPOSITIVOS MÓVILES

B3.1. Carbide.j 1.5 [2]

Carbide.j 1.5 es un paquete de software dirigido a desarrolladores de aplicaciones MIDP (Mobile Information Device Profile) y de Perfil Personal (PP). Durante la instalación del Carbide .j también se instala y se configura automáticamente un SDK de Nokia, este es el Nokia Prototype SDK 4.0 for Java™ ME, el cual contiene siete simuladores para MIDP:

- Simuladores MIDP del Prototype 4.0 S40 (resoluciones: 128x128, 128x160, 208x208, 240x320)
- Simuladores MIDP del Prototype 4.0 S60, con tamaños variados de pantalla.
- Simulador MIDP del Prototype 4.0 S80 (resolución: 640x200)
- Simulador MIDP del Prototype 4.0 7710 (resolución: 640x320)

Este software se puede integrar con Borland JBuilder, NetBeans, Eclipse e IBM WebSphere Studio Device Developer 5.6 and 5.7.

B3.1.1. Recomendaciones del sistema

El Carbide .j y los SDK de Nokia que soporta se han evaluado completamente con los siguientes componentes HW y software:

Entorno básico:

- Microsoft Windows XP (SP 2 o posterior).
- Java™ 2 SDK, Standard Edition 1.4.2 o posterior (se ha evaluado con 1.4.2_06 y 5.0)
- Conexión a Internet para el proceso de registro.

Recomendaciones mínimas del sistema:

- 1 GHz o un procesador más rápido
- 512 Megabytes de RAM
- 500 Megabytes de espacio de disco libre

Recomendaciones sugeridas:

- 2 GHz o un procesador más rápido
- 1024 Megabytes de RAM

B3.1.2. Herramientas de instalación opcional

- Borland JBuilder X / 2005
- NetBeans 4.0 / 4.1 / 5.0
- IBM® WebSphere® Studio Device Developer 5.6 / 5.7

- Eclipse 3.0 o Eclipse 3.1
- Nokia PC Suite 6.8. Este software se requiere para la utilización de la herramienta de despliegue.
- S60 3rd Edition SDK para Symbian OS, para MIDP (necesario para la depuración en el dispositivo).

B3.2. SDKs [3]

B3.2.1. S60 3rd Edition SDK for Symbian OS for MIDP

Este SDK permite el desarrollo de aplicaciones MIDP para dispositivos móviles basados en la plataforma S60.

Características:

- Esta basado en la Plataforma de Desarrollo tercera Edición de la S60 y en Symbian OS v9.1.
- Incluye toda funcionalidad clave necesaria para el desarrollo de aplicaciones, en la cual se incluye la documentación, la información de referencia del API, y un simulador.
- Soporta MIDP 2.0, CLDC 1.1, Nokia UI API, API de Mensajería Inalámbrica (JSR-120 y extensiones WMA JSR-205), API Mobile Media (JSR-135), APIs de Java para Bluetooth (JSR-82), API Mobile 3D Graphics para J2ME (JSR-184), API FileConnection (JSR-75), API de Gestión de Infromación Personal (JSR-75), API de Servicios Web para J2ME (JSR-172), API de Seguridad y de Servicios Confiables para J2ME (JSR-177), API de Localización (JSR-179) y el API de SIP para J2ME (JSR-180).

B3.2.2. Series 60 2nd Edition SDK for Symbian OS Supporting Feature Pack 3 for MIDP

Este SDK permite el desarrollo de aplicaciones MIDP para dispositivos móviles basados en la plataforma S60, por ejemplo los equipos móviles Nokia N90 y N70.

Características:

- Esta basado en la Plataforma de Desarrollo segunda Edición de la S60 y en Symbian OS v8.1a.
- Incluye toda funcionalidad clave necesaria para el desarrollo de aplicaciones, en la cual se incluye la documentación, la información de referencia del API, y un simulador.
- Soporta MIDP 2.0, CLDC 1.1, Nokia UI API, API de Mensajería Inalámbrica (JSR-120), API Mobile Media (JSR-135), APIs de Java para Bluetooth (JSR-82), API Mobile 3D Graphics para J2ME (JSR-184), API FileConnection (JSR-75), API de Gestión de Infromación Personal (JSR-75) y el API de Servicios Web para J2ME(JSR-172)

B3.3. Sun Java Wireless Toolkit 2.5 for CLDC, Beta Release [4]

El Conjunto de Herramientas Inalámbricas en lenguaje Java de Sun (WTK) (conocido antes como el Conjunto de Herramientas J2ME- Sun Java Wireless Toolkit) es un conjunto de herramientas para la creación de aplicaciones Java, que tienen por fin su ejecución en dispositivos que obedezcan la tecnología Java para la Industria Inalámbrica (Java Technology for the Wireless

Industry, JTWI - JSR 185) y a la especificación de la Arquitectura de Servicio Móvil (Mobile Service Architecture, MSA - JSR 248). El WTK 2.5 consta de herramientas para la construcción, diversas utilidades y un simulador.

El WTK implementa muy buenas capacidades, las cuales expone a través de APIs estándar que han sido definidas por medio de Procesos de la Comunidad Java (Java Community Process, JCP). Las APIs que soporta el WTK2.5 son las siguientes:

- ✓ Arquitectura de Servicio Móvil 1.0 (Mobile Service Architecture, MSA) (JSR 248) (implementación temprana, ya que su implementación aún no está completa)
- ✓ Tecnología Java para la Industria Inalámbrica (Java Technology for the Wireless Industry, JTWI) 1.0 (JSR 185)
- ✓ Connected Limited Device Configuration (CLDC) 1.1 (JSR 139)
- ✓ Mobile Information Device Profile (MIDP) 2.0 (JSR 118)
- ✓ Paquetes PDA opcionales, para la implementación en la plataforma J2ME (JSR 75)
- ✓ APIs de Java para Bluetooth (JSR 82)
- ✓ API Mobile Media (MMAPI) 1.1 (JSR 135)
- ✓ Especificación de Servicios Web J2ME (JSR 172)
- ✓ API de Seguridad y Servicios Confiables para J2ME (JSR 177)
- ✓ API de localización para J2ME (JSR 179)
- ✓ API de SIP para J2ME (JSR 180)
- ✓ API Mobile 3D Graphics para J2ME (JSR 184)
- ✓ API de Mensajería Inalámbrica (WMA) 2.0 (JSR 205)
- ✓ API Content Handler (JSR 211)
- ✓ API Scalable 2D Vector Graphics para J2ME (JSR 226)
- ✓ API de Tarificación (JSR 229)
- ✓ Suplementos Avanzados en Multimedia (JSR 234)
- ✓ API Mobile Internationalization (JSR 238)

También se pueden desarrollar aplicaciones para CLDC 1.0 y MIDP 1.0.

El WTK 2.5 Beta incluye todas las características de desarrollo avanzado que se encuentran en las versiones 2.2, 2.3 Beta, entre las cuales se tienen: firma de MIDlets, gestión de certificados, simulación a través del aire (OTA), simulación de push registry, etc.

Por tratarse de una versión Beta, presenta inconvenientes con el MMAPI en la reproducción continua de paquetes pequeños de audio, por lo cual fue necesario manejar paquetes de audio más grandes.

B3.3.1. Requerimientos software del sistema

- Microsoft Windows XP
- Plataforma JavaTM 2, Edición Estándar (Java SE SDK), version 1.5.0.
- Apple QuickTime player (se requiere para la reproducción de AMR)

B3.3.2. Requerimientos hardware mínimos del sistema

- 50 MB espacio en disco duro
- 128 MB RAM del sistema
- 800 MHz Pentium III CPU

B4. EL API DE SIP PARA J2ME (JSR-180)

B4.1. VISIÓN GENERAL [5][6][7]

- Interfaz Java estandarizada para permitir comunicaciones SIP en dispositivos J2ME.
- Paquete opcional para la plataforma J2ME, siendo MIDP el perfil principal objetivo.
- La especificación se basa en el CLDC1.0, pero también puede utilizarse con LDC.
- Proporciona acceso a los servicios terminales de SIP, a nivel de transacción con el usuario (Transaction User level access).
- Soporta una programación de SIP simple y confortable.
- Se trata de un API de SIP a nivel general, flexible, el cual abstrae el núcleo de la red (IETF SIP o 3GPP IMS SIP).

El API de SIP para J2ME se diseñó para el establecimiento y control de las sesiones multimedia en las redes de telecomunicaciones que se basan en el Protocolo de Internet (IP). SIP, también es un protocolo de señalización para todas las redes inalámbricas basadas en IP, conferencias por Internet, telefonía, notificación de eventos, y mensajería instantánea. El objetivo del API de SIP para J2ME es definir un API SIP, general, para clientes J2ME, para que de esta forma las aplicaciones SIP puedan funcionar en dispositivos de memoria limitada.

Esta API permite que dispositivos móviles Java puedan enviar y recibir mensajes SIP, ya que el MIDP 2.0 de J2ME incluye soporte para los sockets TCP/IP y los datagramas UDP/IP, que utiliza este protocolo para su transporte. Se trata de un API compacto y genérico lo cual permite que se utilice con cualquier perfil J2ME, no solamente con el MIDP. Como mínimo requiere la versión 1.0 del CLDC (Connected Limited Device Configuration), pero también se puede utilizar con el CDC (Connected Device Configuration). La API se integra en el CLDC's GCF (Generic Connection Framework).

La API de SIP para J2ME esta definida en el paquete: `javax.microedition.sip`. Este paquete contiene ocho (8) interfaces y cuatro (4) clases.

Jerarquía de clases

```
java.lang.Object
      javax.microedition.sip.SipAddress
      javax.microedition.sip.SipHeader
      javax.microedition.sip.SipRefreshHelper
      java.lang.Throwable
            java.lang.Exception
                  java.io.IOException
                        javax.microedition.sip.SipException
```

Jerarquía de Interfaces

```
javax.microedition.io.Connection
      javax.microedition.sip.SipConnection
            javax.microedition.sip.SipClientConnection
            javax.microedition.sip.SipServerConnection
      javax.microedition.sip.SipConnectionNotifier
```

```
javax.microedition.sip.SipClientConnectionListener
javax.microedition.sip.SipDialog
javax.microedition.sip.SipRefreshListener
javax.microedition.sip.SipServerConnectionListener
```

La Figura B1, muestra un diagrama de clases simplificado del API, la relación de las clases, la herencia de `javax.microedition.Connection` y la relación con `javax.microedition.Connector`.

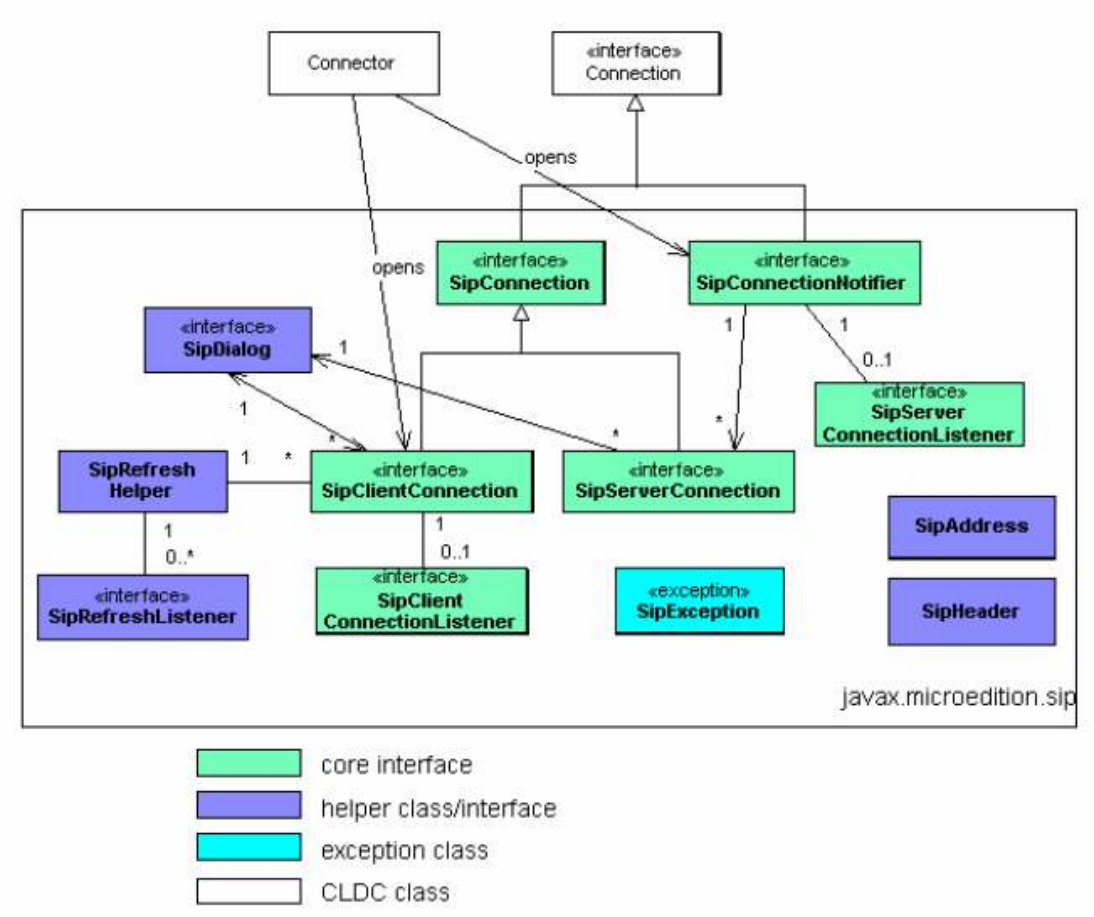

Figura B1: Diagrama de Clases simplificado del API de SIP para J2ME

A continuación se describirán las interfaces y clases de manera resumida.

Interfaces	Descripción
`SipConnection`	La interfaz básica para las conexiones SIP; mantiene las propiedades y métodos comunes para las sub-interfaces: `SipClientConnection` y `SipServerConnection`.
`SipClientConnection`	Representa una transacción con un cliente SIP. Una aplicación puede

	crear una nueva `SipClientConnection` utilizando `javax.microedition.io.Connector` o `SipDialog`.
`SipServerConnection`	Representa una transacción con un servidor; esta se crea por medio del `SipConnectionNotifier` cuando se recibe una nueva petición.
`SipConnectionNotifier`	Define un elemento de notificación de conexión con el servidor SIP, el cual va ingresando en una cola los mensajes entrantes. Para recibir las peticiones de entrada las aplicaciones utilizan el método `acceptAndOpen()`, el cual acepta y abre una nueva `SipServerConnection`. Si no hay mensajes en la cola el método se bloquea hasta que se reciba una nueva petición.
`SipServerConnectionListener`	Una interfaz de escucha para peticiones SIP de entrada.
`SipClientConnectionListener`	Una interfaz de escucha para respuestas SIP de entrada.
`SipDialog`	Representa un diálogo SIP. El `SipDialog` puede recuperarse de un objeto `SipConnection`, cuando este se encuentre disponible, lo cual sucede después de una petición de `INVITE` o `SUBSCRIBE`.
`SipRefreshListener`	Una interfaz de escucha para la clase `RefreshHelper`; declara un método `refreshEvent()` que toma un `refreshID` para identificar la tarea correspondiente de actualización, y un `statusCOde` que representa el resultado de la actualización: `0` para cancelación, `200` para éxito, y cualquier otra cosa para identificar un fallo.

Tabla B1: Interfaces del API de SIP para J2ME

Clases	**Descripción**
`SipAddress`	Provee un analizador sintáctico y genérico de direcciones SIP. Puede utilizarse para analizar una dirección SIP, o para construir una.
`SipHeader`	Provee un analizador sintáctico y genérico, para ayuda, de encabezados SIP. Puede utilizarse para analizar valores de cadenas de los encabezados que se obtienen de los mensajes SIP, o para construir encabezados SIP.
`SipRefreshHelper`	Implementa la funcionalidad que facilita el manejo de peticiones de actualización, en nombre de la aplicación.
`SipException`	Una clase de excepciones para errores específicos SIP.

Tabla B2: Clases del API de SIP para J2ME

El API de SIP para J2ME esta hecho para usarse en aplicaciones que necesiten implementar funcionalidades del tipo agente de usuario. Estas pueden utilizar el `SipClientConnection` para implementar un agente de usuario cliente, y para implementar el agente de usuario servidor pueden utilizar las interfaces `SipConnectionNotifier` y `SipServerConnection`. Esto de puede ver en la Figura B2.

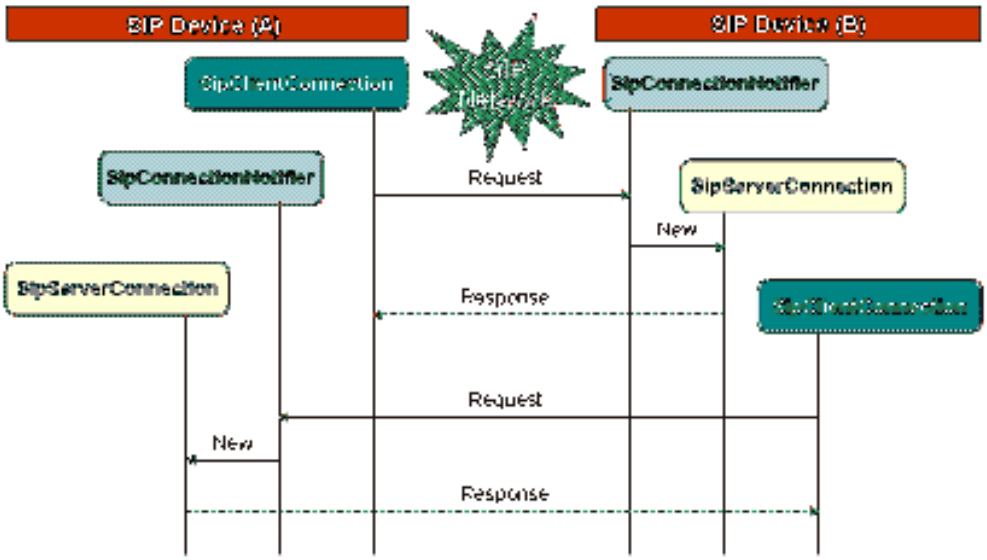

Figura B2: Flujo general de peticiones y respuestas entre dos clientes móviles SIP

Un dispositivo móvil debe actuar como un cliente SIP al enviar peticiones a los servidores SIP, y también como un servidor SIP al aceptar peticiones de otros clientes SIP.

Por ejemplo, una aplicación llama al método `Connector.open()`, mediante el cual se crea una instancia de una conexión SIP nueva, la cual retornará una `SipClientConnection` o una `SipServerConnection`, dependiendo de la URI SIP que se le entregue al método `open()`.

La aplicación deberá llamar al método `Connector.close()`, cuando la conexión finalice.

Una URI SIP tiene la siguiente sintáxis:

```
{scheme}:[{target}][{params}]
```

Donde:

- `scheme` es el esquema SIP que soporta el sistema, puede ser `sip` o `sips`. En el protocolo `sips` las conexiones que acepta el servidor son aquellas que corresponden a peticiones que se encuentren bajo transporte seguro, tal cual como se define en el estándar de la RFC 3261.

- `target` es la dirección del usuario, la cual puede ser algo como `userName@target-host.com:portNumber` o puede ser un número telefónico simplemente.

- `params` representa los parámetros adicionales; por ejemplo, `transport=udp` para especificar que UDP va a ser el protocolo de transporte.

Si la dirección del servidor se incluye, significa que se abre una conexión nueva con el cliente, sino, significa que se abre una conexión nueva con el servidor. A continuación se muestran algunos ejemplos:

Conexión con un cliente:

```
...
// conexión con un cliente
SipClientConnection scc = (SipClientConnection)
    Connector.open("sip:max@company.com");
SipClientConnection scc2 = (SipClientConnection)
    Connector.open("sip:jack@128.11.1.2:5100");
...
```

Conexión con un servidor:

```
...
// conexión con un servidor
SipConnectionNotifier scn = (SipConnectionNotifier)
    Connector.open("sip:5500");
...
SipConnectionNotifier scn2 = (SipConnectionNotifier)
    Connector.open("sip:");
...
```

El método siguiente demuestra como abrir una conexión entre un cliente SIP y un servidor que se encuentra corriendo en la dirección 102.12.1.1 y que escucha por el puerto 5050.

```
private void sendMessage(String message) {
    SipClientConnection sc = null;
    try {
        // se abre una conexión SIP con el servidor 102.12.1.1 puerto
        // 5050
        sc = (SipClientConnection)
            Connector.open(sip:user@102.12.1.1:5050);
        // se inicializa la petición SIP con el método MESSAGE
        sc.initRequest("MESSAGE", null);
        // se colocan los encabezados del mensaje SIP
        sc.setHeader("Subject", subject.getString());
        sc.setHeader("Content-Type", "text/plain");
        sc.setHeader("Content-Length", ""+text.length());
        // se envía el mensaje SIP a la red
        OutputStream os = sc.openContentOutputStream();
        os.write(message.getBytes());
        os.close(); // se cierra la conexión
    } catch(Exception ex) {
        ex.printStackTrace();
    }
}
```

El segmento de código siguiente implementa el método `notifyRequest()` de la interfaz `SipServerConnectionListener`. Cuando el servidor recibe el mensaje de petición SIP para la inicialización, este recupera los encabezados del mensaje y su contenido, luego retorna al cliente una respuesta.

```
public void notifyRequest(SipConnectionNotifier scn) {
   try {
```

```
      ssc = scn.acceptAndOpen();
      if(ssc.getMethod().equals("MESSAGE")) {
         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);
            String tmp = new String(testBuffer, 0, i);
         }
         ssc.initResponse(200);
         ssc.send();
      }
   } catch(IOException ex) {
   }
}
```

B4.2. TÉRMINOS IMPORTANTES [7]

A continuación se definen algunos términos que se utilizan en la especificación.

- **Conexión cliente**: en esta API de SIP la conexión cliente (clase `SipClientConnection`) es una interfaz para la transacción SIP con el cliente, la cual envía una solicitud original y recibe respuestas hasta la respuesta final. Ver también Transacción SIP y Conexión servidor.

- **Conexión servidor**: en esta API de SIP la conexión servidor (clase `SipServerConnection`) es una interfaz para la transacción SIP con el servidor, la cual recibe una petición original y envía respuesta(s) hasta la respuesta final. Ver también Transacción SIP y Conexión cliente.

- **Transacción SIP**: una transacción SIP ocurre entre un cliente y un servidor, y comprende todos los mensajes desde la primera petición que se envió desde el cliente hacia el servidor, hasta una respuesta final (no `1xx`) enviada desde el servidor hacia el cliente. Si la petición es `INVITE` y la respuesta final no es un `2xx`, la transacción incluye también un `ACK` para la respuesta.

- **Diálogo SIP**: en esta API de SIP, la clase `SipDialog` representa la información acerca de un diálogo SIP. Al utilizarse esa interfaz, el usuario está en la capacidad de enviar peticiones subsiguientes dentro de un diálogo. Un diálogo es una relación SIP para a par, que persiste durante algún tiempo, entre dos agentes de usuario. Un diálogo se establece mediante mensajes SIP, tales como respuestas `2xx` a una petición `INVITE`. Un diálogo se identifica por medio de un identificador de llamada, una etiqueta local, y una etiqueta remota.

- **Agente de Usuario (UA, User Agent)**: una entidad lógica que puede actuar tanto como un agente de usuario cliente y un agente de usuario servidor. Típicamente una aplicación (por ejemplo, una MIDlet) va a implementar la funcionalidad de UA utilizando el API de SIP para J2ME. Ver también UAC y AUS.

- **Agente de Usuario Cliente (User Agent Client, UAC)**: un agente de usuario cliente es una entidad lógica que crea una nueva petición, y luego utiliza los elementos del estado de la transacción del cliente para enviarla. En otras palabras, si un elemento de software inicia una petición, este actúa como un UAC, por cuanto dure la transacción. Si luego, éste recibe una petición, en este caso asumirá el papel de agente de usuario servidor para el procesamiento de la transacción.

- **Agente de Usuario Servidor (User Agent Server, UAS)**: un agente de usuario servidor es una entidad lógica que genera una respuesta a una petición SIP. La respuesta puede aceptar, rechazar, o redireccionar la petición. Este papel dura solamente por lo que dure la transacción. En otras palabras, si un elemento de software responde a una petición, este actúa como un UAS por el tiempo que dure la transacción. Si luego éste genera una petición, asumirá el papel de agente de usuario cliente para el procesamiento de la transacción.

- **Método**: el método es la función principal que se supone que una petición va a invocar en el servidor. El método se lleva en el mismo mensaje de petición. Métodos de ejemplo son el `INVITE`, `MESSAGE` y el `BYE`.

- **Respuesta Provisional**: es una respuesta que utiliza el servidor para indicar el progreso, pero ésta no termina la transacción SIP. Las respuestas `1xx` son provisionales, las otras respuestas se consideran finales.

- **Respuesta Final**: una respuesta que termina una transacción SIP, al contrario de una respuesta provisional la cual no termina la transacción. Todas las respuestas `2xx`, `3xx`, `4xx`, `5xx` y `6xx` son finales.

B4.3. CASOS DE USO PRINCIPALES [8]

Los casos de uso del API de SIP se pueden dividir en tres categorías de alto nivel: no relacionadas con el UAC (User Agent Client) o con el UAS (User Agent Server), funcionalidad del lado del UAC y funcionalidad del lado del UAS.

La implementación del API de SIP de la S60 de Nokia desarrolla los siguientes casos de uso, de nivel principal. Algunos de esos casos de uso se han dividido en casos de uso más pequeños.

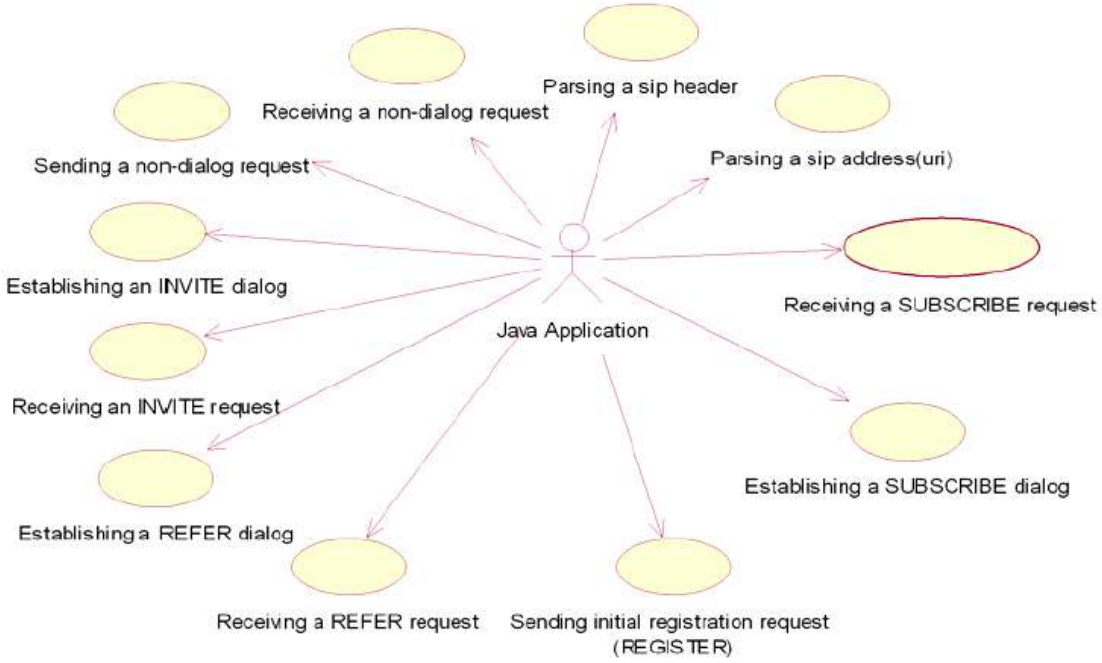

Figura B3: Casos de Uso principales de SIP

Los siguientes casos de uso son aplicables tanto al lado UAC como al UAS:

- El análisis de un encabezado SIP.
- El análisis de una dirección SIP (URI).
- El envío de una nueva petición dentro de un diálogo existente. Esta funcionalidad se incluye dentro de los casos de uso de 'Establecimiento de un diálogo INVITE', 'Establecimiento de un diálogo SUBSCRIBE' y 'Establecimiento de un diálogo REFER'.

Los siguientes casos de uso pertenecen a la funcionalidad del lado de UAC. Estas funcionalidades incluyen el envío de una petición y la recepción de respuestas a la petición que se envió.

- Establecimiento de un diálogo, INVITE, SUBSCRIBE y REFER. Ver los casos de uso 'Establecimiento de un diálogo INVITE', 'Establecimiento de un diálogo SUBSCRIBE' y 'Establecimiento de un diálogo REFER'.
- Envío de una petición de no diálogo. Por ejemplo, un mensaje.
- Gestión del registro.

Los siguientes casos de uso pertenecen a la funcionalidad del lado de UAS. Estas funcionalidades incluyen la recepción de una petición y el envío de respuestas a la petición que se recibió.

- Recepción de una petición que crea un diálogo (INVITE, SUBSCRIBE o REFER). Ver 'Recepción de una petición de INVITE', 'Recepción de una petición de SUBSCRIBE' y 'Recepción de una petición de REFER'. Nota: el caso de uso de 'Recepción de una petición

de INVITE' incluye la recepción del ACK, más otras peticiones que el UAC puede enviar antes que el lado del UAS halla enviado 200 OK a la petición.
- Recepción de una petición de no diálogo, enviada por fuera de cualquier diálogo. Ver el caso de uso 'Recepción de una petición de no diálogo'.

Los casos de uso específicos del INVITE se ilustran más detalladamente en la siguiente figura:

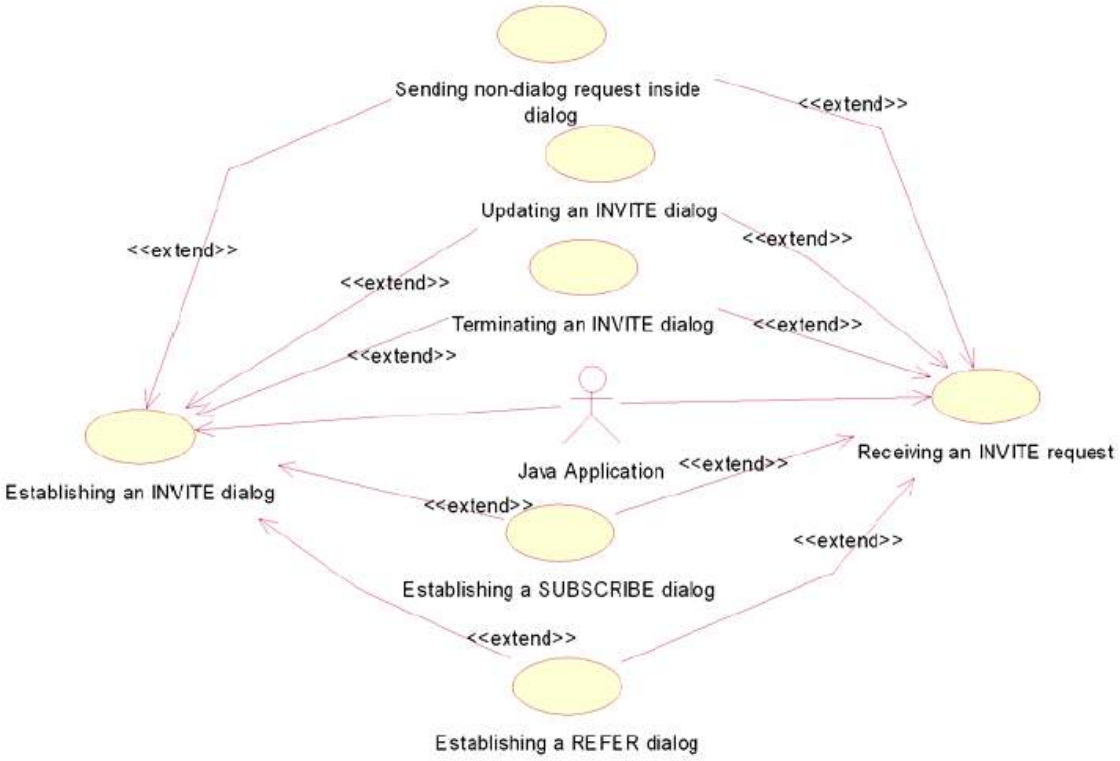

Figura B4: Casos de uso del INVITE de SIP

B4.4. MÉTODOS DE PETICIÓN SIP [8]

Hay varios métodos de petición en SIP y se están creando métodos nuevos todo el tiempo (por ejemplo `PRACK`) por parte de la IETF. La RFC 2543 (SIP básico) de la IETF, que se encuentra en http://www.ietf.org/rfc/rfc2543.txt?number=2543 define:

- `INVITE`: indica que un usuario o servicio está siendo invitado para participar en una sesión de llamada.
- `ACK`: confirma que un cliente recibió una respuesta final a una petición de `INVITE`.
- `BYE`: termina una llamada y puede enviarse por el llamante o el llamado.
- `CANCEL`: cancela cualquier búsqueda pendiente, pero no termina una llamada que ya haya sido aceptada.
- `OPTIONS`: cuestiona (o comprueba) las capacidades de los servidores.

- `REGISTER`: registra la dirección listada en el campo `To` con el servidor SIP.

La petición `PRACK` juega el mismo papel de `ACK`, pero es para respuestas provisionales. `PRACK` es un mensaje de SIP normal, sin embargo, como `BYE`, tiene su propia respuesta, a diferencia del `ACK`. Para más información acerca del método `PRACK` visitar la RFC pertinente en http://www.ietf.org/rfc/rfc3262.txt.

Junto con los métodos mencionados se encuentran otros métodos de petición estándares. Estos se enumeran a continuación (con enlaces para las descripciones individuales).

- `REFER`: http://www.ietf.org/rfc/rfc3515.txt.
- `PUBLISH`: http://www.ietf.org/rfc/rfc3903.txt.
- `SUBSCRIBE`: http://www.ietf.org/rfc/rfc3265.txt.
- `NOTIFY`: http://www.ietf.org/rfc/rfc3515.txt.
- `MESSAGE`: http://www.ietf.org/rfc/rfc3428.txt.

También es posible enviar métodos no estándares, pero esos están fuera del alcance de este documento.

Los tipos de respuestas se listan a continuación. Tener cuenta que son similares a las de HTTP:

- `1xx`: Provisional, petición recibida, se procede a procesar la petición.
- `2xx`: Éxito, la acción se recibió exitosamente, es entendida y aceptada.
- `3xx`: Redirección, se necesita que se haga una acción adicional para completar la petición.
- `4xx`: Error de Cliente, la petición contiene una sintaxis mala o no puede ser procesada por el servidor.
- `5xx`: Error del Servidor, el servidor falló en la realización de una petición aparentemente válida.
- `6xx`: Falla Global, la petición no se puede procesar por ningún servidor.

B4.5. DESCRIPCIÓN DETALLADA DE CLASES Y DE INTERFACES [9]

- **javax.microedition.io.Connector**

`Connection open(String name)`

En la señalización SIP, UDP es el protocolo de conexión por defecto. La implementación de la S60 cambia el protocolo de conexión a TCP cuando se adiciona el parámetro `transport=tcp`, a la URI que se pasa como argumento en la operación `Connector.open()`. Por ejemplo: `sip:sippy@operator.com:5070;transport=tcp`.

Cuando el mensaje que se va a enviar es mayor de 1300 bytes, aproximadamente, el mensaje se envía automáticamente vía TCP. Tener en cuenta que, por ejemplo, el protocolo no puede cambiarse en medio de un dialogo de UDP a TCP.

Restricciones concernientes a la creación de objetos `SipClientConnection` y `SipConnectionNotifier`:

- No se soporta la conexión `sips`
- Solamente se soporta el encabezado `Accept-Contact` en la creación del objeto `SipConnectionNotifier`. Por ejemplo:

```
SipConnectionNotifier scn = (SipConnectionNotifier)
Connector.open("sip:*;type=\"application/vnd.company.x-game\"");
```

- La implementación de la S60 no soporta al `SipConnectionNotifier` cuando actúa en modo dedicado.

- **javax.microedition.sip.SipAddress**

Esta interfaz se implementó como se especifica en el API de SIP para J2ME (JSR-180).

- **javax.microedition.sip.SipClientConnection**

`int enableRefresh(SipRefreshListener srl)`

Este método solo se puede llamar por las peticiones `REGISTER` y `SUBSCRIBE`.

`SipClientConnection initCancel()`

El usuario no puede poner encabezados o contenido a una petición `CANCEL`.

`void initRequest(java.lang.String method, SipConnectionNotifier scn)`

Los encabezados del sistema '`CSeq`' y '`Call-ID`' no están disponibles inmediatamente después de la llamada a esta operación. Luego de la llamada a esta operación, los encabezados '`To`', '`From`' y '`Contact`' están disponibles.

Una petición `<EXTENSION>` significa cualquier petición arbitraria propia del usuario.

Las siguientes peticiones pueden ponerse como argumento `method`: `REGISTER`, `INVITE`, `OPTIONS`, `MESSAGE`, `SUBSCRIBE`, `PUBLISH`, `UPDATE`, `REFER` y `<EXTENSION>`.
Las siguientes peticiones no se pueden poner como argumento `method`: `ACK`, `CANCEL`, `BYE`, `PRACK`, `UPDATE` y `NOTIFY`.

`boolean receive(long timeout)`

Esta operación arroja una `IOException`, y si un error ocurre en el stack de SIP, durante la transacción, el estado del objeto se pone como `TERMINATED`.

`void setCredentials(java.lang.String username, java.lang.String passoftwareord, java.lang.String realm)`

La implementación de la S60 no le pasa a la MIDlet las respuestas `401` o `407`. La implementación de la S60 arroja una `IOException` si el servidor de registro regresa estos valores y las credenciales no han sido puestas por la MIDlet. Al usuario tampoco se le pregunta por las credenciales, en el caso en que el servidor envíe una respuesta `401` o `407`. Lo que esto implica es que la MIDlet tiene que poner las credenciales antes de que se envíe una petición.

`void setRequestURI(java.lang.String URI)`

La implementación de la S60 no soporta esta operación. Si la operación se llama en el estado `INITIALIZED` esta operación arroja una `SipException` con un `INVALID_OPERATION`.

- **javax.microedition.sip.SipConnection**

`void addHeader(java.lang.String name, java.lang.String value)`

Adiciona un encabezado al mensaje SIP.

Ejemplo 1: adición de una fila de encabezado única. El mensaje ya contiene la `Route` del encabezado: `<sip:carol@chicago.com>`.

```
addHeader("Route", "<sip:alice@atlanta.com>");
```
El resultado será:
```
Route:<sip:alice@atlanta.com>
Route:<sip:carol@chicago.com>
```

Ejemplo 2: adición de filas de encabezado múltiples, como una lista separada por comas. El mensaje ya contiene la `Route` de encabezado: `<sip:carol@chicago.com>`.

```
addHeader("Route", "<sip:alice@atlanta.com>,<sip:bob@biloxi.com>");
```
El resultado será:
```
Route: <sip:alice@atlanta.com>
Route: <sip:bob@biloxi.com>
Route: <sip:carol@chicago.com>
```

`java.lang.String getHeader()`

Retorna el valor más alto del campo encabezado, o `null` si el mensaje actual no tiene un encabezado o por alguna otra razón el encabezado no se encuentra disponible (por ejemplo, mensaje no inicializado). Los encabezados se almacenan como objetos separados y el encabezado más alto se retorna.

Ejemplo 1: obtener el valor más alto del encabezado `Route`, de un mensaje que contiene 3 encabezados `Route`.

```
Route: <sip:alice@atlanta.com>
Route: <sip:carol@chicago.com>
```

```
Route: <sip:bob@biloxi.com>
getHeader("Route");
```
El resultado es:
```
<sip:alice@atlanta.com>
```

java.lang.String[ ] getHeaders(java.lang.String name)

Obtiene el valor, o valores, del encabezado, de un tipo de encabezado específico. El método retorna los valores del encabezado, separados en un arreglo, sin considerar como hayan sido almacenados en el mensaje.

Ejemplo 1: obtener los encabezados `Route`, del mensaje que contiene 2 encabezados `Route`, en filas de encabezado, separadas.

```
Route: <sip:alice@atlanta.com>
Route: <sip:carol@chicago.com>
getHeaders("Route");
```
El resultado es un arreglo de cadenas:
```
{"<sip:alice@atlanta.com>", "<sip:carol@chicago.com>"}
```

Ejemplo 2: obtener los encabezados `Route`, de un mensaje que contiene 3 encabezados `Route`, en un valor de encabezado, separado por comas.
```
Route:
<sip:alice@atlanta.com>,<sip:carol@chicago.com>,<sip:bob@biloxi.com>
getHeaders("Route");
```
El resultado es un arreglo de cadenas:
```
{"<sip:alice@atlanta.com>", "<sip:carol@chicago.com>",
"<sip:bob@biloxi.com>"}
```

void setHeader(java.lang.String name, java.lang.String value)

Pone el valor del encabezado en el mensaje SIP. Si el encabezado no existe, este se adiciona al mensaje, de lo contrario el encabezado se sobreescribe. Sí existen valores múltiples del campo de encabezado, el más alto se sobreescribe.

Ejemplo 1: reemplazo de una fila única del encabezado. El mensaje ya contiene los siguientes encabezados:

```
Route: <sip:alice@atlanta.com>
Route: <sip:carol@chicago.com>
setHeader("Route", "<sip:bob@biloxi.com>");
```
El resultado será:
```
Route: <sip:bob@biloxi.com>
Route: <sip:carol@chicago.com>
```

Ejemplo 2: poner filas múltiples del encabezado, como una lista separada por comas. Este mensaje ya contiene un encabezado:

```
Route: <sip:carol@chicago.com>
setHeader("Route", "<sip:alice@atlanta.com>,<sip:bob@biloxi.com>");
```

El resultado será:
```
Route: <sip:alice@atlanta.com>
Route: <sip:bob@biloxi.com>
```

`void removeHeader(java.lang.String name)`

Retira el encabezado del mensaje. Si existen múltiples valores del campo encabezado, el más alto es el que se retira. Si el encabezado que se pone no se encuentra, este método no hace nada.

Ejemplo 1: remover el encabezado de un mensaje que contiene dos encabezados `Route`, en filas de encabezados diferentes.

```
Route: <sip:alice@atlanta.com>
Route: <sip:carol@chicago.com>
removeHeader("Route");
el resultado es:
Route: <sip:carol@chicago.com>
```

`java.lang.String getRequestURI()`

La implementación de la S60 no soporta esta operación. Esta operación siempre regresa `null`.

`void send()`

La implementación de la S60 trata de enviar una petición o una respuesta durante 30 segundos. Esto significa que la operación `send()` bloquea la aplicación durante 30 segundos, en el peor de los casos. Una `IOException` se arroja si después de los 30 segundos el envío de la petición o respuesta aún falla.

Las siguientes características y restricciones se aplican en el lado del UAS:

- Una respuesta exitosa (<=299) para las peticiones `SUBSCRIBE` y `REFER`, no es enviada hasta que la MIDlet envíe la primera petición `NOTIFY` al UAC. Por ejemplo, cuando se ha enviado `NOTIFY`, primero se envía la respuesta exitosa e inmediatamente luego la petición `NOTIFY`.
- Una `IOException` se arroja si algún error ocurre (por ejemplo, en el stack SIP) luego de la creación del objeto `SipServerConnection` y antes de que se llame la operación `send()`. En esta situación, el estado del objeto `SipServerConnection` se pone en `TERMINATED`.

- **javax.microeditions.sip.SipConnectionNotifier**

Esta interfaz se implementó y especificó en el API de SIP para J2ME (JSR-180).

- **javax.microedition.sip.SipDialog**

`java.lang.String getDialogID()`

En los casos de `SUBSCRIBE` y `REFER`, el ID del diálogo está disponible en el UAS luego de enviar la primera petición `NOTIFY`.

`SipClientConnection getNewClientConnection(java.lang.String method)`

- **javax.microedition.sip.SipException**

Esta interfaz se implementó y especificó en el API de SIP para J2ME (JSR-180).

- **javax.microedition.sip.SipHeader**

Esta interfaz se implementó y especificó en el API de SIP para J2ME (JSR-180).

- **javax.microedition.sip.SipRefreshHelper**

`java.io.OutputStream update(int refreshID, java.lang.String[ ] contact, java.lang.String type, int length, int expires)`

Las restricciones de la implementación de la S60 para la operación `update()` son:

- Solo se permite la actualización del encabezado '`Contact`' en el caso `REGISTER`.
- Solo se permite un encabezado '`Contact`' para poner el argumento '`contact`'.

- **javax.microedition.sip.SipRefreshListener**

`void refreshEvent(int refreshID, int statusCode, java.lang.String reasonPhrase)`

Cuando ocurre una actualización esta operación de "callback" no se llama por la implementación de la S60. Esta operación de callback se llama cuando la tarea de actualización empieza, se actualiza (por la MIDlet), se cancela o falla.

- **javax.microedition.sip.SipServerConnection**

La funcionalidad de `REGISTER` no soporta para el lado del UAS.

`void initResponse(int code)`

Una respuesta de `100` a una petición de `INVITE` se envía automáticamente por el stack de SIP, en la implementación de la S60. Poner y enviar una respuesta `100` a un `INVITE` es algo que ignora el

lado UAS. Una respuesta de `100` no es permitida para otros métodos de petición y en estos casos se arroja una `IllegalArgumentException`.

- **javax.microedition.sip.SipServerConnectionListener**

`void notifyRequest(SipConnectionNotifier scn)`

Esta operación de "callback" no se llama si la MIDlet se bloquea, al mismo tiempo, en la operación `SipConnectionNotifier.acceptAndOpen()`. La razón de esta funcionalidad es que la operación `acceptAndOpen()` retorna un objeto `SipServerConnection` y la llamada a la operación `notifyRequest()` es innecesaria en esta situación, porque la operación `acceptAndOpen()` ya proporcionó un objeto `SipServerConnection` a la MIDlet.

B4.6. IMPLEMENTACIÓN DE REFERENCIA DE LA JSR-180 [6]

Una implementación de referencia para la JSR-180 se encuentra disponible a través de Nokia. Se trata de una implementación en Java, construida con la CLDC 1.1 y la MIDP 2.0. Soporta mensajería SIP real a través de una red de área local, lo cual le permite a las aplicaciones el envío de mensajes SIP, la recepción de mensajes SIP, de forma sincronizada, y la recepción no sincronizada de mensajes de notificación SIP por parte de las interfaces de escucha.

Para descargar la implementación de referencia de la JSR 180 se debe ser miembro del Foro de Nokia, y el proceso de registro es gratis. Una vez se es miembro se puede descargar y desempaquetar el archivo. La estructura de directorios con la que se encontrará será la siguiente:

```
/sip/sip1_0-ri-bin
    /examples (aplicaciones de ejemplo)
    /lib (archivo JAR que contiene las APIs de SIP)
    /midp (archivos ejecutables MIDP 2.0 y CLDC 1.1 que incluyen las APIs
de SIP)
```

El directorio `midp` contiene a su vez subdirectorios, uno de los cuales es el directorio `bin`, este contiene varias herramientas, dentro de las que se encuentra `sipa-midp.exe` quien es la encargada de lanzar el emulador para SIP. Se puede utilizar el comando `-help` para que se muestre la lista de todas las opciones.

Para empezar a experimentar con los ejemplos que trae la JSR-180, se deberá correr `sipa-midp`. Como resultado de la ejecución se verán 4 MIDlets de ejemplo, como en la Figura B5.

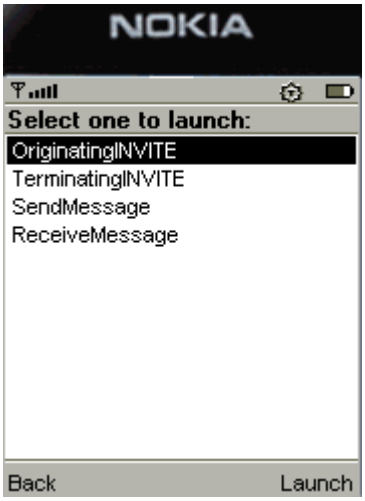

Figura B5: Implementación de referencia de la JSR-180

Se pueden probar `SendMessage` y `ReceiveMessage` juntos, ver Figura B6. Para ello se empiezan cada una en una ventana de emulador diferente. Los emuladores pueden correr en el mismo servidor o en diferentes servidores, en este caso se deberá conocer la dirección IP o el nombre del servidor donde se ponga a correr el `ReceiveMessage`. Las siguientes figuras muestran el ejemplo en funcionamiento.

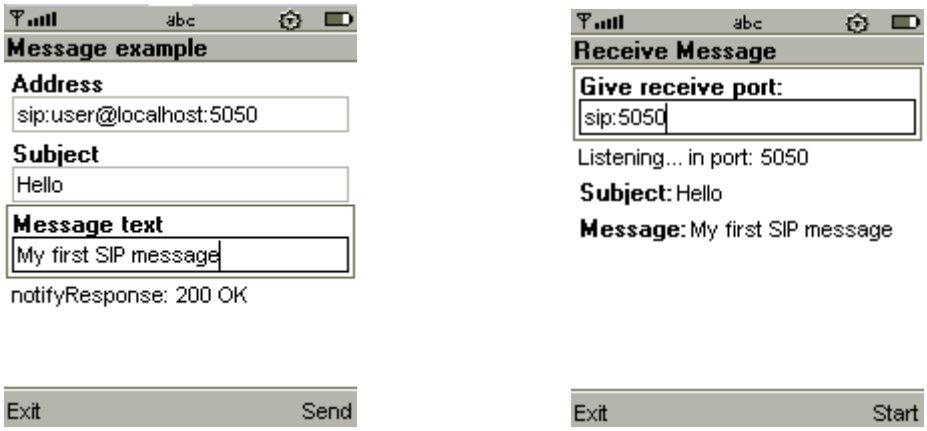

Figura B6: MIDlets SendMessage y ReceiveMessage

Para probar las MIDlets `OriginatingINVITE` y `TerminatingINVITE`, estas también se deben abrir en ventanas de emulador diferentes, y de igual forma pueden estar en el mismo o en diferentes servidores, ver Figura B7. Una vez que las MIDlets estén corriendo, se deberán introducir los datos solicitados y se verá algo como en las siguientes figuras. En este ejemplo `OriginatingINVITE` envió un `INVITE` y un `ACK`, y `TerminatingINVITE` le respondió con `180` para indicar el timbre y con `200` para indicar que todo salió bien (`OK`).

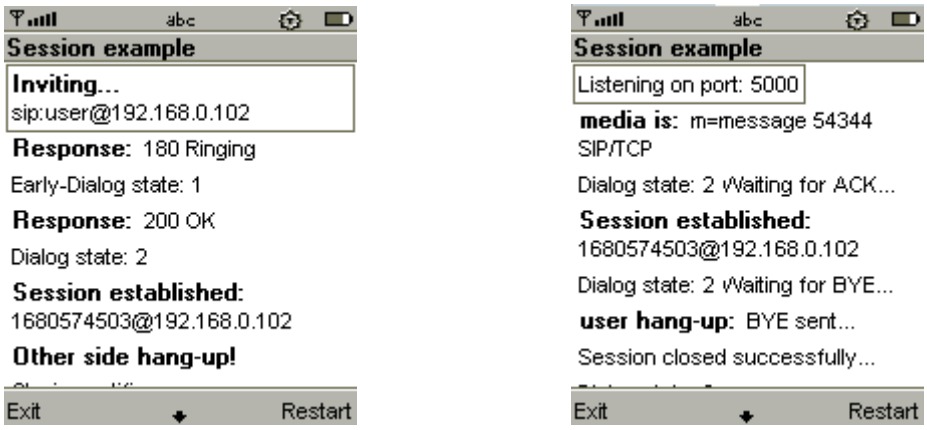

Figura B7: MIDlets OriginatingINVITE y TerminatingINVITE

B5. BIBLIOGRAFÍA

[1] Eclipse. Disponible en web: <http://www.eclipse.org/>

[2] Nokia Forum. *Carbide.j 1.5*. [En línea]. Disponible en web: <http://forum.nokia.com/info/sw.nokia.com/id/d9f7e9b2-3932-4358-9e8e-aa5cd26be54e.html> [Consulta: Marzo de 2006]

[3] Nokia Forum. *S60 Platform SDKs for Java*. [En línea]. Disponible en web: <http://forum.nokia.com/info/sw.nokia.com/id/6e772b17-604b-4081-999c-31f1f0dc2dbb/S60_Platform_SDKs_for_Symbian_OS_for_Java.html> [Consulta: Marzo de 2006]

[4] SDN – Sun Developer Network. *Sun Java Wireless Toolkit 2.5 for CLDC, Beta 2 Release*. [En línea]. Disponible en web: <http://java.sun.com/products/sjwtoolkit/download-2_5.html> [Consulta: Mayo de 2006]

[5] Mikko Lönnfors, Jari Kinnunen. *"SIP for Mobile Phones"*. [En línea]. Mayo 2004. Nokia Research Center, Helsinki. Disponible en web: <http://phoenix.labri.fr/documentation/sip/Documentation/Papers/Programming_SIP/Presentation/Jain/SIP_for_J2ME.pdf> [Consulta: Agosto de 2006]

[6] Qusay H. Mahmoud. *"Getting Started with SIP API for J2ME (JSR 180)"*. [En línea]. Noviembre 2004. Disponible en web: <http://developers.sun.com/techtopics/mobility/apis/articles/sip/ Getting Started with SIP API for J2MEJSR 180.doc> [Consulta: Julio de 2006]

[7] The Java Community Process. *"SIP API (JSR 180) for Java™ 2 Micro Edition"*. [En línea]. Diciembre 2004. Disponible en web: <http://www.microjava.com/nokia/documents?content_id=7848> [Consulta: Enero de 2005]

[8] Nokia Forum. *"MIDP: SIP API Developer's Guide"*. [En línea]. Junio 2006. Disponible en web: <http://sw.nokia.com/id/753ce82b-58fc-4879-a944-d3d239d34784/MIDP_SIP_API_Developers_Guide_v1_0_en.pdf > [Consulta: Octubre de 2006]

[9] Nokia Corporation. *"SIP API for Java™ 2 Micro Edition (JSR-180): Implementation Notes" Version 1*. [En línea]. Enero 2006. Disponible en web: <http://www.forum.nokia.com/> [Consulta: Marzo de 2006]