Recomendaciones Para el Desarrollo de Aplicaciones P2P Utilizando el Protocolo SIP

ANEXO D
TECNOLOGÍAS PARA EL DESARROLLO DE APLICACIONES MÓVILES P2P CON EL PROTOCOLO SIP

Wilson Yecit Ortiz Sánchez

Director: Ing. Javier Alexander Hurtado

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telemática
Línea de Investigación en Ingeniería de Sistemas Telemáticos
Popayán, Enero de 2007

TABLA DE CONTENIDO

ANEXO D TECNOLOGÍAS PARA EL DESARROLLO DE APLICACIONES MÓVILES P2P CON EL PROTOCOLO SIP 1

D1 SIP PARA J2ME (JSR 180) 1

D1.1 Documentación 2
D1.2 Entornos de Desarrollo Soportados 2
D1.3 Adquisición de los SDKs SIP para J2ME 2
D1.4 Especificación Técnica 3
D1.5 Sumario de Clases 3
D1.7 Ejemplo de Aplicación SIP con J2ME 9

D2 PLUGIN SIP PARA SYMBIAN 12

D2.1 Características del Plugin 12
D2.2 Características de la Pila SIP 13
D2.3 Documentación 14
D2.4 Entornos de Desarrollo Soportados 14
D2.5 Adquisición de los SDKs y Plugins para Symbian 15
D2.6 Especificación Técnica 15
D2.7 API de Cliente SIP 16
D2.8 Uso de las Clases del API SIP para Symbian 25
D2.9 Ejemplo de Aplicación SIP con Symbian 34

BIBLIOGRAFÍA 41

LISTA DE FIGURAS

Figura 1. Diagrama de clases simplificado del API SIP. 3
Figura 2. Implementación de UAC y UAS..... 6
Figura 4. Clases relacionadas con el registro. 21
Figura 5. Clases relacionadas al dialogo. 22
Figura 6. Clases relacionadas con la transacción. 23
Figura 7. Clases relacionadas al mensaje SIP. 24
Figura 8. Clase CSIPRefresh. 25
Figura 9. Clase CSIPHttpDigest. 25
Figura 10. Iniciación del cliente SIP. 26
Figura 11. Orden recomendado para anular los objetos del API de Cliente SIP. 27
Figura 12. Enviando una solicitud SIP. 29
Figura 13. Recibiendo una solicitud SIP. 30
Figura 14. Registro con refresco. 32
Figura 15. Creación de un dialogo con el envío de un INVITE. 33

ANEXO D TECNOLOGÍAS PARA EL DESARROLLO DE APLICACIONES MÓVILES P2P CON EL PROTOCOLO SIP

El estudio realizado para conocer acerca de las tecnologías que se pueden utilizar para implementar aplicaciones P2P con el protocolo SIP, se hizo sobre diferentes plataformas como J2ME, Symbian, BREW y Waba. De las cuales solo las dos primeras cuentan con el soporte necesario para el desarrollo y simulación de aplicaciones móviles que hagan uso del protocolo SIP.

J2ME dispone de un API SIP (JSR 180) que permite a los dispositivos móviles que cuentan con una maquina virtual de Java enviar y recibir mensajes SIP. Symbian cuenta con un plugin para los SDKs de los teléfonos NOKIA de la serie 60 con Sistema Operativo Symbian, el cual permite a los desarrolladores crear aplicaciones en C++. También existen algunos SDKs de la serie 60 de Nokia que ya incluyen el soporte para SIP lo que evita la necesidad de instalar el plugin mencionado.

El presente capitulo se centrará principalmente en dar a conocer lo concerniente al API SIP para J2ME, el plugin SIP para la serie 60 de Nokia y el desarrollo aplicaciones móviles P2P con el protocolo SIP sobre estas dos plataformas.

D1 SIP PARA J2ME (JSR 180)

El JSR 180 es una especificación que define un paquete J2ME opcional que proporciona recursos necesarios para los dispositivos móviles para que puedan enviar y recibir mensajes SIP. El API disponible definida por el JSR 180 esta diseñada para ser compacta y genérica, y provee la funcionalidad del nivel de transacción, está

integrada en el Framework de conexión genérica definido en la Configuración (CLDC) y diseñada para que pueda ser usada con diferentes perfiles J2ME, pudiendo usarse también con la configuración CDC (Configuración de Dispositivos con Conexión). La plataforma mínima requerida por esta API es el J2ME CLDC v1.0.

D1.1 Documentación

EL SDK instalado suministra documentación precisa que especifica el uso de las clases, el protocolo SIP y proporciona diferentes ejemplos básicos de aplicación SIP para esta plataforma.

D1.2 Entornos de Desarrollo Soportados

El SDK para el desarrollo de aplicaciones J2ME con SIP pude ser utilizado en los siguientes entornos de desarrollo:

JBuilder

Eclipse

J2ME Wireless Toolkit

Nokia Developer's Suite para la plataforma Micro de Java 2

IBM Websphere Studio Device Developer.

Nota: En el anexo B se describe la configuración de algunas de estas herramientas, la creación de proyectos y la emulación de aplicaciones móviles P2P con SIP.

D1.3 Adquisición de los SDKs SIP para J2ME

Los SDKs para trabajar SIP en aplicaciones móviles pueden ser descargados de la página del Forum de Nokia (http://www.forum.nokia.com) con previo registro de usuario. Los SDKs son gratuitos, pero deben ser registrados en línea o por medio de una solicitud del código de registro para su utilización permanente.

D1.4 Especificación Técnica

La figura 1 muestra el diagrama de clases simplificado del API, las relaciones entre las clases y la herencia.

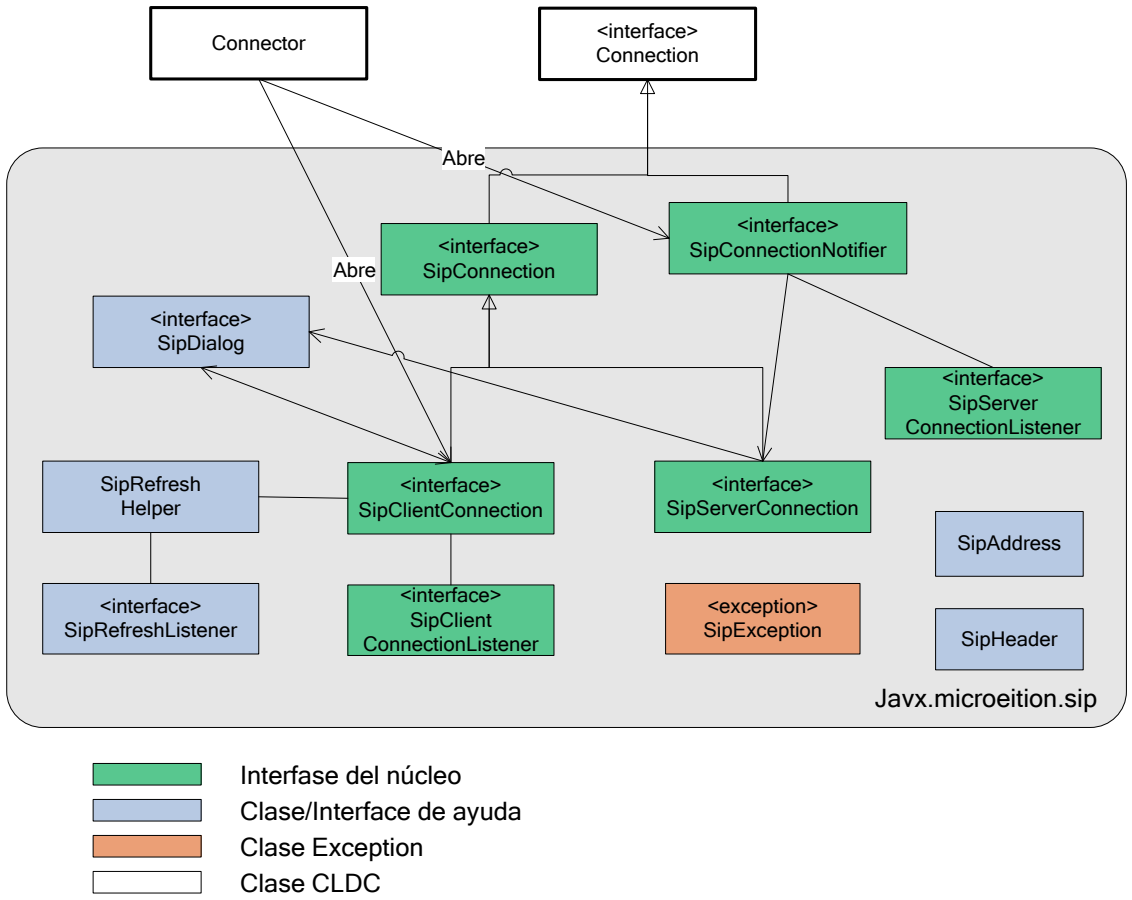

Figura 1. Diagrama de clases simplificado del API SIP.

D1.5 Sumario de Clases

A continuación se listan las clases e interfaces que componen el API con una breve explicación de cada una y la jerarquía de estas.

D1.5.1 Interfases

SipClientConnection: Interfase publica que representa una transacción de cliente SIP.

SipClientConnectionListener: Interfase Listener para respuesta SIP entrante. *SipConnection:* es la interfase base para las conexiones SIP.

SipConnectionNotifier: esta interfase define una notificación de conexión de servidor.

SipDialog: representa un Dialogo SIP.

SipRefreshListener: Interfase Listener para eventos RefreshHelper.

SipServerConnection: representa la transacción del servidor SIP.

SipServerConnectionListener: Interfase Listener para peticiones SIP entrantes.

D1.5.2 Clases

SipAddress: proporciona un analizador de direccionamiento SIP genérico.

SipHeader: proporciona una ayuda de análisis de direccionamiento SIP genérico.

SipRefreshHelper: esta clase implementa la funcionalidad que facilita el manejo de peticiones de refresco en nombre de la aplicación.

D1.5.3 Excepciones

SipException: esta es una clase de excepción para errores específicos SIP.

D1.5.4 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**

D1.5.5 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** [1]

D1.6 Uso de las Clases del API SIP para J2ME

Para implementar un Agente de Usuario en una aplicación, esta debe contener las funcionalidades de Cliente de Agente de Usuario y de Servidor de Agente de Usuario, los cuales crearan una conexión de UAC a UAS. Esto genera en realidad dos conexiones en el mismo terminal que las aplicaciones usarán durante el establecimiento y duración de la conexión, una conexión de cliente para enviar solicitudes y la otra de servidor para responder a las solicitudes de otros UAC. La figura 2 ilustra dos terminales que implementa las funcionalidades de UAC y UAS.

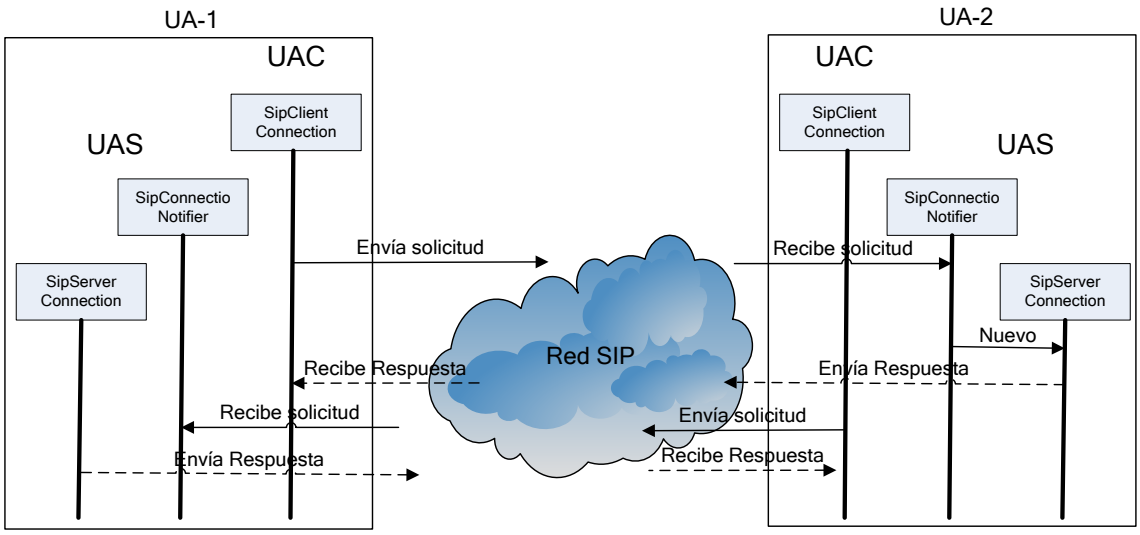

Figura 2. Implementación de UAC y UAS.

Para ilustrar mejor el uso de las clases del API JSR180 durante una transacción se describe a continuación la función que cada una realiza.

Suponiendo que un usuario (UA-1) decide enviar una invitación SIP a un conocido suyo (UA-2) y asumiendo que en ambos UAs se encuentran “observando” o “escuchando” solicitudes SIP entrantes a trabes de la interfaz SipConnectionNotifier que se inicializa con la ayuda del método “open” de la clase connector.

```
scn = (SipConnectionNotifier) Connector.open("sip:5060")
```

Cuando el usuario ordena enviar la solicitud, se abre una conexión de cliente hacia el destino nuevamente con la ayuda del método Open.

```
scc = (SipClientConnection)  Connector.open(dir_destino);
```

Sobre la conexión abierta se inicializa la recepción de respuestas entrantes.

```
scc.setListener(this)
```

En la conexión se establece el método a enviarse y todas las cabeceras necesarias para el mensaje.

```
scc.initRequest("INVITE", null);
scc.setHeader("From", dir_origen);
scc.setHeader("To", dir_destino);
```

Luego se abre un flujo hacia la conexión para depositar en él, el cuerpo del mensaje.

```
OutputStream os = scc.openContentOutputStream();
os.write(sdp.getBytes());
```

Por ultimo se cierra el flujo.

```
os.close();
```

En el UA-2 el mensaje es recibido a trabes del método de observación notifyRequest.

```
public void notifyRequest(SipConnectionNotifier sn)
```

En este se revisa si se trata de una solicitud soportada y se responde con una respuesta temporal.

```
if(ssc.getMethod().equals("INVITE")) {
        ssc.initResponse(180);
```

Se extraen las cabeceras y el cuerpo del mensaje para desplegar al usuario la solicitud recibida y esperar a que este determine si aceptar o no la solicitud.

```
String contentType = ssc.getHeader("Content-Type");
String contentLength = ssc.getHeader("Content-Length");
```

```
String origen = ssc.getHeader("From");
```

Si el usuario acepta, se envía una respuesta 200 con las cabeceras y el cuerpo del mensaje necesario.

```
ssc.initResponse(200);
ssc.setHeader("Content-Length", "" + sdp.length());
        :
os.close();
```

Se guarda el dialogo de la conexión, el cual servirá para enviar y recibir mensajes sin tener que establecer nuevas conexiones.

```
dialog = ssc.getDialog();
```

Ahora en el UA-1, se recibe la respuesta (200) en el método notifyResponse.

```
notifyResponse(SipClientConnection scc)
```

Si esta de acuerdo con las cabeceras y el cuerpo del mensaje recibido, guarda el dialogo, envía un ACK e inicia la sesión negociada.

```
dialog = scc.getDialog();
scc.initAck();
scc.send();
```

Ahora en el UA-2, se recibe el ACK que confirma la aceptación de lo negociado, por lo cual se le da inicio a la sesión.

```
else if(ssc.getMethod().equals("ACK")) {
```

```
ssc.close();
```

Finalmente se ha establecido la sesión que puede involucrar cualquier tipo de medio o recurso soportados por ambos UAs.

D1.7 Ejemplo de Aplicación SIP con J2ME

Las siguientes funciones son ejemplos de código acerca de cómo abrir conexiones SIP y el uso de las principales clases SipClientConnection, SipServerConnection y SipConnectionNotifier. Los ejemplos de las clases de ayuda: SipDialog, SipAddress, SipHeader y RefreshHelper se muestran en la documentación del API por lo que se recomienda su revisión. El siguiente ejemplo muestra como abrir una conexión de cliente SIP, enviar una solicitud y recibir una respuesta. La interfaz usada en este ejemplo de conexión de cliente es SipClientConnection. [2]

```
public void sendTextMessage(String msg) {
    SipClientConnection sc = null;
    try {
        // abre la conexión SIP
        sc = (SipClientConnection)
    Connector.open(“sip:sippy.tester@host.com:5060”);
        // inicializar la solicitud SIP MESSAGE
        sc.initRequest(“MESSAGE”, null);
        // establecer las cabeceras
        sc.setHeader(“From”, “sip:user@host.com”);
        sc.setHeader(“Subject”, “testing...”);
```

```
            // Escribir el cuerpo del mensaje
            sc.setHeader(“Content-Type”, “text/plain”);
            sc.setHeader(“Content-Length”, Integer.toString(msg.length()));
            OutputStream os = sc.openContentOutputStream();
            os.write(msg.getBytes());
            os.close(); // cerrar el flujo y enviar el mensaje a la red
            // Esperar máximo 15 segundos por la respuesta
            sc.receive(15000);
            // respuesta recivida
            if(sc.getStatusCode() == 200) {
            // manejo de la respuesta 200 OK
            } else {
            // manejo de otras respuestas
            }
            sc.close();
        } catch(Exception ex)
        // Manejo de excepciones
        }
    }
```

El siguiente ejemplo de código muestra como abrir una conexión a un servidor SIP, recibir una solicitud y enviar una respuesta. Las interfaces usadas en el ejemplo de conexión del servidor son SipConnectionNotifier y SipServerConnection.

```
public receiveMessage() {
    SipConnectionNotifier scn = null;
    SipServerConnection ssc = null;
    String method = null;
    try {
        //Abrir la conexión de servidor SIP y escuchar las  solicitudes entrantes
        scn = (SipConnectionNotifier) Connector.open(“sip:”);
        // Bloquea y espera por solicitudes entrantes.
        // SipServerConnection is established and returned
        // Cuando una nueva solicitud es recibida.
        ssc = scn.acceptAndOpen();
        // Se revisa que solicitud llegó
        method = ssc.getMethod();
        if(method.equals(“MESSAGE”)) {
        // leer el contenido de MESSAGE
        String contentType = ssc.getHeader(“Content-Type”);
        if((contentType != null) && contentType.equals(“text/plain”)) {
        InputStream is = ssc.openContentInputStream();
        int ch;
        // Leer el contenido
        while ((ch = is.read()) != -1) {
        ...
        }
        }
        // Inicializa una respuesta SIP 200 OK y la devuelve
```

```
            ssc.initResponse(200);
            ssc.send();
            }
            ssc.close();
        } catch(Exception ex) {
        // manejo de excepciones
        }
    }
```

D2 PLUGIN SIP PARA SYMBIAN

El plugin para el SDK de esta plataforma permite el desarrollo de aplicaciones con SIP en lenguaje C++. Este amplía las características de la Plataforma de la serie 60, agregando las funcionalidades SIP que no están incluidas en los dispositivos o no están especificados para un cierto dispositivo en su SDK original, es decir habilita a los dispositivos que no contienen el soporte nativo para la tecnología SIP, incluyéndoles en las aplicaciones un archivo mediante el cual les permite manejarlo. El plugin SIP permite la prueba y depuración de aplicaciones SIP en C++, sin necesidad de tener a mano un dispositivo real. Adicionalmente proporciona un emulador de Servidor SIP, el cual facilita probar y depurar aplicaciones sin necesidad de acceder a un servidor SIP verdadero (en el anexo B se presenta una descripción de este Servidor Proxy SIP).

D2.1 Características del Plugin

Soporte de Extensiones SIP, el plugin proporciona características SIP al emulador de la serie 60, permitiendo la comprobación y puesta a punto de las aplicaciones SIP en C++, sin necesidad de un dispositivo real.

Servidor SIP, el plugin incluye un emulador de servidor que proporciona la funcionalidad de un servidor de registro y Proxy acorde con la IETF. Esto permite probar y poner a punto las aplicaciones sin necesidad de contar con un servidor real.

Emulador de Soporte de Conectividad, el plugin permite comunicaciones entre emuladores de la Serie 60 y un Emulador de Servidor SIP en un solo PC o en múltiples PCs con una conexión TCP/IP. Este emulador permite comprobar las interacciones SIP realizadas entre los UAs sin necesidad de contar con los dispositivos, una red SIP real, o un servidor SIP.

Soporte del SDK, el plugin SIP 1.0 es compatible con la 2 Edición del SDK de la Serie 60.

El plugin SIP 2.0 es compatible con los SDKs de la serie 60 segunda edición paquete de características 1.

El plugin SIP 4.0 es compatible con los SDKs de la serie 60 segunda edición paquetes de características 2 y 3.

El SDK Nokia_Prototype_SDK_4_0_Beta incluye el plugin SIP 4.0 por lo cual tiene las mismas especificaciones.

D2.2 Características de la Pila SIP

Soporte para la plataforma de la serie 60 segunda edición y para la plataforma de la serie 60 segunda edición paquete de características 1, los dispositivos compatibles con estas plataformas de la serie 60 no incluyen soporte nativo para SIP. Este plugin proporciona un archivo *.sis en el que contiene la pila SIP que será incluida en las aplicaciones desarrolladas.

Soporte para la plataforma de la serie 60 segunda edición paquete de características 2

Los dispositivos compatibles con la plataforma de la serie 60 segunda edición paquete de características 2 incluyen el soporte SIP nativo.

Actualización de la pila SIP

El plugin SIP 4.0 para el SDK contiene una actualización de la pila SIP. Esta actualización es para ser usada con los dispositivos Nokia 6680. [3]

D2.3 Documentación

El plugin es proporcionado con una amplia documentación que contiene la especificación del protocolo SIP, instalación y configuración del plugin, configuración de algunos IDES, guía de desarrollo, guía de referencia de las APIs, librería de desarrollo Symbian, herramientas y utilidades, introducción a la tercera edición de la serie 60 y una guía de inicio entre otros. Esta documentación se puede encontrar en el directorio: [directorio de instalación del SDK o plugin]\Symbian\9.1\S60_3rd\S60Doc. El subdirectorio 9.1 puede variar dependiendo del SDK y plugin instalado.

Adicionalmente contiene un ejemplo de aplicación SIP, el cual puede ser encontrado en la carpeta: \Symbian\9.1\S60_3rd\S60Ex\.

D2.4 Entornos de Desarrollo Soportados

El plugin es compatible con los SDKs soportados por los siguientes IDEs:

Eclipse.

Netbeans.

Borland C++ Mobile Edition

CodeWarrior para Symbian OS.

Microsoft Visual C++ 6.0

Microsoft Visual Estudio.NET 2003 (requiere Carbide.vs)

D2.5 Adquisición de los SDKs y Plugins para Symbian

Cualquiera de las versiones del plugin, los SDKs y algunas de las herramientas mencionadas puede ser descargadas de la pagina del Forum Nokia (http://www.forum.nokia.com) con previo registro de usuario. Los plugins al igual que la mayoría de las herramientas que en este sitio se encuentran son gratuitas, pero deben ser registradas en línea o por medio de una solicitud de código de registro para su utilización permanente.

D2.6 Especificación Técnica

El SDK proporciona un manejo muy completo del entorno SIP mediante cuatro APIs que se describen brevemente a continuación.

El API de Codec SIP, que proporciona servicios de codificación y descodificación de cabeceras y campos SIP. También proporciona medios para convertir mensajes SIP del formato UTF-8 ASCII al formato de implementación interna del API y viceversa.

El API SIP Profile, que proporciona un servicio de registro para aplicaciones SIP.

El API Client Resolver, define una arquitectura para aplicaciones que realizan la resolución de posibles clientes designados para una solicitud SIP entrante.

El API SIP Client, proporciona el acceso a los servicios básicos de la pila SIP. Los servicios más esenciales son el envío y recepción de mensajes SIP, creación de registro y formación y negociación de diálogos iniciados por INVITE, REFER y SUBSCRIBE.

De estas APIs la más involucrada con el desarrollo de aplicaciones móviles P2P con SIP es el API de Cliente SIP, por lo cual se realizara una descripción mas detallada de

sus características. Si se desea obtener información precisa acerca de las anteriores APIs se recomienda revisar el documento “s60_cpp_sdk_3rd_apirefguide.chm” que aparece adjunto a los archivos de instalación de cualquier SDK de la serie 60 con soporte SIP, en la carpeta “S60Doc” o revisar los documentos electrónicos “SIP_Codec_API_Specification_v1_0_en”, “SIP_Profile_API_Specification_v1_0_en” y “SIP_Client_API_Specification_v1_0_en” que se encuentran disponibles en el sitio del forum de Nokia (www.forum.nokia.com).

D2.7 API de Cliente SIP

El API de cliente SIP es un API servidor de cliente, que permite a múltiples clientes usar la pila SIP simultáneamente, estos clientes son las diferentes aplicaciones que pueden estar ejecutándose al mismo tiempo en un terminal. Como requisito de implementación solicita que una aplicación implemente un conjunto de funciones de retorno, las cuales pueden usar el API de cliente SIP para pasar los eventos generados a la aplicación. Es decir se requiere de la implementación de algunas funciones que SIP pueda invocar cuando necesite realizar alguna notificación a la aplicación.

D2.7.1 Sumario de Clases

Los servicios ofrecidos por el API de Cliente SIP son proporcionados como una librería dinámicamente enlazada llamada sipclient.DLL. La lista de clases es presentada en la tabla 3.

Paquete	Clase
sip.H	CSIP
sipclienttransaction.H	CSIPClientTransaction
sipconnection.H	CSIPConnection
sipconnectionobserver.H	MSIPConnectionObserver
sipdialog.H	CSIPDialog
sipdialogassocbase.H	CSIPDialogAssocBase

sipinvitedialogassoc.H	CSIPInviteDialogAssoc
sipmessageelements.H	CSIPMessageElements
sipobserver.H	MSIPObserver
siprefresh.H	CSIPRefresh
sipnotifydialogassoc.H	CSIPNotifyDialogAssoc
sipreferdialogassoc.H	CSIPReferDialogAssoc
sipregistrationbinding.H	CSIPRegistrationBinding
sipregistrationcontext.H	MSIPRegistrationContext
siprequestelements.H	CSIPRequestElements
sipresponseelements.H	CSIPResponseElements
sipservertransaction.H	CSIPServerTransaction
sipsubscribedialogassoc.H	CSIPSubscribeDialogAssoc
siptransactionbase.H	CSIPTransactionBase
siphttpdigest.H	CSIPHttpDigest
siphttpdigestchallengeobserver.H	MSIPHttpDigestChallengeObserver

Tabla 1 listado de clases del API de cliente SIP

D2.7.2 Descripción de las Clases

CSIP: Esta clase proporciona la conexión al servidor SIP y proporciona funciones para solicitar los mecanismos de seguridad SIP soportados y el soporte de comprensión de señalización.

CSIPClientTransaction: Clase para administrar las transacciones de cliente SIP. Proporciona servicios para terminar y obtener los parámetros de transacción de cliente SIP.

CSIPConnection: Clase para supervisar el estado de la conexión y enviar solicitudes SIP asociadas a un dialogo.

CSIPDialog: Clase para manejar los diálogos SIP. Proporciona servicios que preguntan el estado del diálogo, obteniendo el diálogo relacionado con las cabeceras SIP y todas las asociaciones del diálogo.

CSIPDialogAssocBase: Clase base para las asociaciones del diálogo SIP. Proporciona servicios para obtener el dialogo SIP asociado, para obtener el tipo de asociación del diálogo actual y para enviar solicitudes de refresco sin destinatario asociadas a un dialogo.

CSIPHttpDigest: Clase para administrar establecimientos SIP de seguridad. La clase proporciona funciones para establecer y eliminar parámetros relacionados con los mecanismos de seguridad.

CSIPInviteDialogAssoc: Clase para administrar asociaciones de diálogo SIP creadas con un INVITE. Proporciona servicios para la creación, uso y terminación de una invitación SIP asociada a un diálogo.

CSIPMessageElements: Clase para la creación y manipulación de elementos opcionales en un mensaje SIP.

CSIPNotifyDialogAssoc: Clase para manejar asociaciones SIP NOTIFY del diálogo. Proporciona los servicios de creación, uso y terminación de asociaciones SIP NOTIFY del diálogo.

CSIPReferDialogAssoc: Clase para manejar asociaciones REFER del diálogo. Proporciona servicios para la creación, uso y terminación de asociaciones REFER del diálogo SIP.

CSIPRefresh: Clase para manejar el refresco SIP. Proporciona funciones para adquirir transacciones SIP asociadas y su estado. La clase también proporciona funciones para actualizar y terminar refrescos automáticos.

CSIPRegistrationBinding: Clase para manejar el registro SIP. La clase proporciona las funciones para la creación y actualización del registro SIP.

CSIPRequestElements: Esta clase proporciona las funciones de creación y manipulación de las direcciones de origen y destino en una solicitud SIP. También proporciona funciones para la manipulación de métodos SIP en solicitudes SIP desconocidas.

CSIPResponseElements: La clase proporciona los servicios de creación y manipulación de respuestas SIP. Se usa para crear y manipular respuestas SIP incluso el código de estado, *reason phrase* y elementos opcionales como cabeceras de usuario, contenido y su tipo.

CSIPServerTransaction: Clase para manejar las transacciones de servidor SIP. Proporciona servicios para la creación, terminación y toma de los parámetros de transacción SIP. El cliente no puede instanciar esta clase.

CSIPSubscribeDialogAssoc: Clase para manejar asociaciones del diálogo SUBSCRIBE SIP. Proporciona servicios de creación, uso y terminación de asociaciones de dialogo SIP SUBSCRIBE.

CSIPTransactionBase: Clase base para el manejo de transacciones SIP. Proporciona servicios para solicitar tipos de transacciones y sus estados.

MSIPConnectionObserver: Interfaz a ser implementada por los clientes de CSIPConnection. La interfaz permite recibir solicitudes, respuestas, notificaciones del estado de la conexión y notificaciones de error de la pila SIP.

MSIPHttpDigestChallengeObserver: El cliente debe implementar esta interfaz si desea proporcionar credenciales de seguridad en las negociaciones recibidas de los servidores SIP.

MSIPObserver: Interfaz de CSIP a ser implementada por el usuario. La interfaz permite recibir solicitudes SIP de una conexión que no ha sido inicializada por el cliente.

MSIPRegistrationContext: interfaz que proporciona información de solicitudes SIP y creación de diálogos.

D2.7.3 Descripción de la Implementación de Clases

Una aplicación que usa el API de cliente SIP crea una instancia de CSIP. La aplicación debe implementar las funciones de observación MSIPobserver, las cuales son usadas por CSIP. Con CSIP es posible solicitar los mecanismos de seguridad soportados y averiguar si se tiene un CSIPConnection para un IAP dado.

Una vez se tiene CSIP, es posible crear una instancia de CSIPConnection. La aplicación debe implementar las funciones de observación MSIPConnectionObserver. Con CSIPConnection, es posible enviar de forma independiente las solicitudes SIP y pedir el estado de la conexión de la red. Las respuestas se reciben a través de MSIPConnectionObserver. La aplicación puede tener múltiples objetos CSIPConnection, pero cada uno de ellos debe tener un IAP diferente. La figura 3 muestra la clase CSIP, CSIPConnection y sus clases de observación

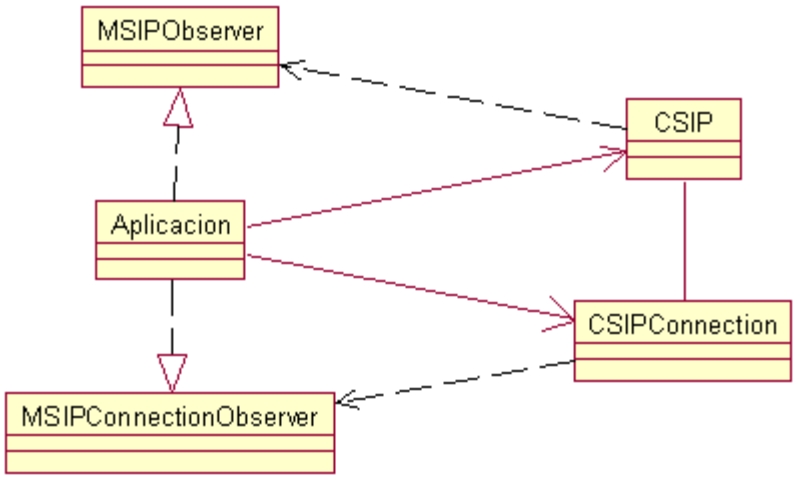

Figura 3. CSIP, CSIPConnection y sus clases de observación.

La clase CSIPRegistrationBinding proporciona los servicios de registro, actualización y eliminación de registro. También es posible configurar la pila SIP para que automáticamente refresque el registro y solicite y establezca el Proxy de salida usado para el registro. La Creación de una instancia de CSIPRegistrationBinding requiere de

la existencia de un CSIPConnection. La figura 4 muestra las clases relacionadas con el registro.

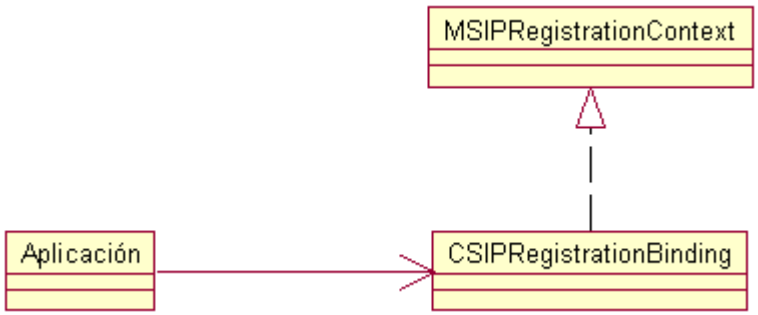

Figura 4. Clases relacionadas con el registro.

CSIPDialogAssocBase es la clase base para CSIPInviteDialogAssoc, CSIPSubscribeDialogAssoc, CSIPReferDialogAssoc y CSIPNotifyDialogAssoc, proporciona una manera para localizar la instancia de CSIPDialog asociada y envía una solicitud de refresco sin destinatario dentro de un diálogo. CSIPDialog es instanciada por el API de Cliente SIP, no puede ser creada por la aplicación.

Cuando un CSIPInviteDialogAssoc, CSIPSubscribeDialogAssoc, CSIPReferDialogAssoc y CSIPNotifyDialogAssoc es asociado, puede ser pasado a una instancia de CSIPDialog existente, o puede usarse para crear un nuevo CSIPDialog.

La clase CSIPInviteDialogAssoc permite a la aplicación iniciar un diálogo y enviar una solicitud INVITE, PRACK, ACK, UPDATE o BYE relacionada al diálogo. Con CSIPSubscribeDialogAssoc, la aplicación puede subscribirse a diferentes eventos enviando solicitudes SUBSCRIBE y de eliminación de suscripción. También es posible pedir a la pila SIP que refresque automáticamente la solicitud SUBSCRIBE. Con CSIPReferDialogAssoc, la aplicación puede definir y enviar solicitudes REFER. Con

CSIPNotifyDialogAssoc, la aplicación puede definir y puede enviar solicitudes NOTIFY. La figura 5 muestra las clases relacionadas con el dilogo.

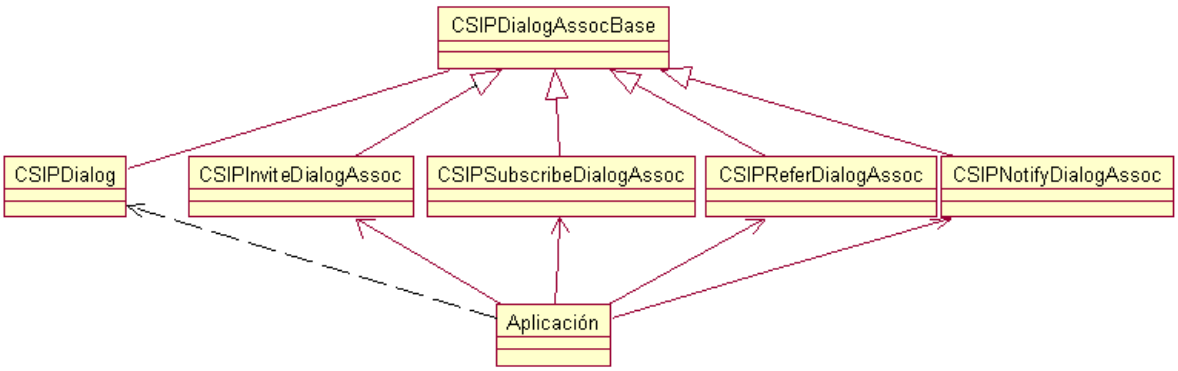

Figura 5. Clases relacionadas al dialogo.

Cuando la aplicación invoca una función que genera una solicitud SIP a ser enviada a la red, la implementación de Cliente SIP instancia un nuevo objeto CSIPClientTransaction. CSIPClientTransaction representa la transacción SIP que consiste en la solicitud y respuesta SIP. Una vez se ha recibido una respuesta, la aplicación puede obtener un CSIPResponseElements de CSIPClientTransaction. También pueden cancelarse ciertas transacciones de cliente o se pueden ser refrescadas.

Cuando se recibe una solicitud SIP de la red, se pasa a la aplicación como una CSIPServerTransaction. La aplicación puede obtener CSIPRequestElements, que describe qué tipo de solicitud SIP fue recibida, de CSIPServerTransaction. Una vez la aplicación ha decidido qué tipo de respuesta(s) va a regresar, usa CSIPServerTransaction para enviarlas.

CSIPTransactionBase proporciona funciones para obtener el estado actual de la transacción y el tipo de la transacción. Algunas operaciones sólo se permiten cuando la transacción está en un cierto estado. CSIPClientTransaction y CSIPServerTransaction

son instanciadas por el API de Cliente SIP; estas no pueden ser creadas por la aplicación. La figura 6 muestra las clases relacionadas con la transacción

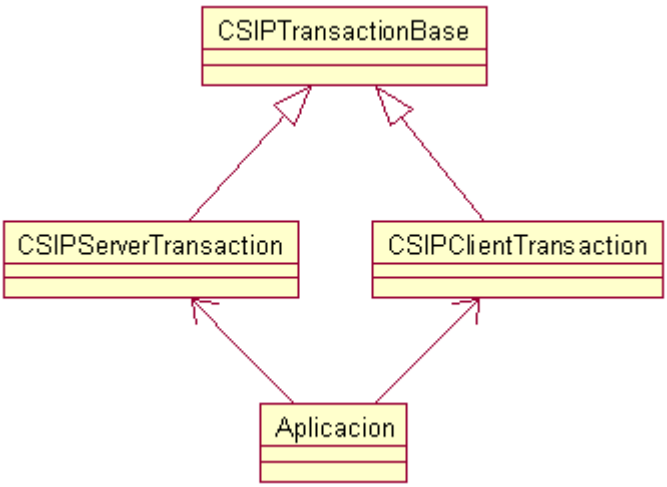

Figura 6. Clases relacionadas con la transacción.

La clase CSIPMessageElements guarda las partes de los mensajes SIP que son comunes a las solicitudes y respuestas SIP. Proporciona funciones para obtener y establecer aquéllas cabeceras SIP que son visibles a la aplicación. El contenido del mensaje SIP también puede obtenerse y puede establecerse. La aplicación generalmente necesita instanciar un objeto CSIPMessageElements cuando va a enviar una solicitud SIP.

CSIPRequestElements contiene CSIPMessageElements y las partes obligatorias de cualquier solicitud SIP. Este proporciona lo necesario para manipular cabeceras To, From, URI Remoto, métodos de solicitud y CSIPMessageElements. La aplicación necesita instanciar un objeto CSIPRequestElements cuando va a enviar una solicitud SIP.

CSIPResponseElements contiene CSIPMessageElements y las partes obligatorias de cualquier respuesta SIP. Proporciona elementos para manipular los códigos de estado y *reason phrase*, y elementos para obtener cabeceras From, To, CSeq y CSIPMessageElements. La figura 7 muestra las clases relacionadas con el mensaje SIP-

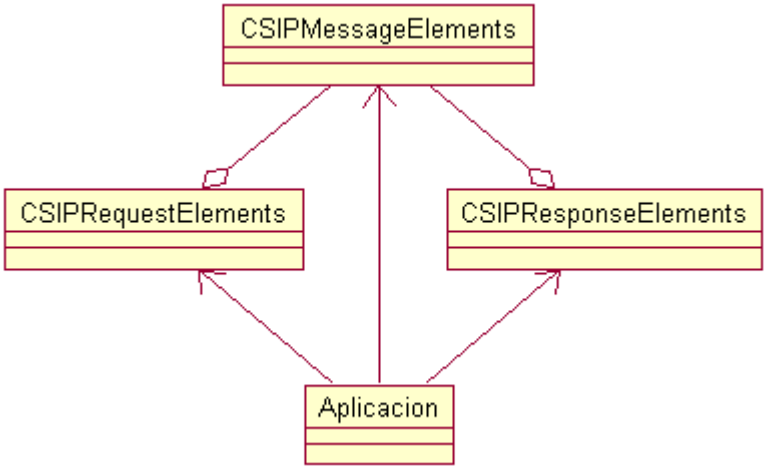

Figura 7. Clases relacionadas al mensaje SIP.

La clase CSIPRefresh se usa cuando la pila SIP debe refrescar una solicitud SIP automáticamente. REGISTER, SUBSCRIBE y una solicitud fuera de un diálogo pueden ser refrescadas mediante esta clase. Una vez se ha enviado una solicitud de refresco, la aplicación puede solicitar el estado de refresco CSIPRefresh. Para solicitudes fuera de un diálogo, CSIPRefresh proporciona funciones de actualización y terminación del refresco. La actualización y terminación de un REGISTER o SUBSCRIBE se realiza a trabes de las clases CSIPRegistrationBinding y CSIPSubscribeDialogAssoc en lugar de usar CSIPRefresh. La figura 8 muestra la clase CSIPRefresh.

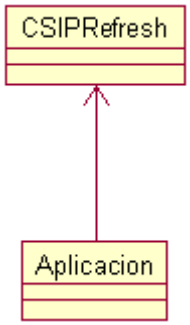

Figura 8. Clase CSIPRefresh.

La clase CSIPHttpDigest contiene sólo funciones estáticas. Proporciona lo necesario para manejar escenarios de seguridad. La figura 9 muestra la clase CSIPHttpDigest.

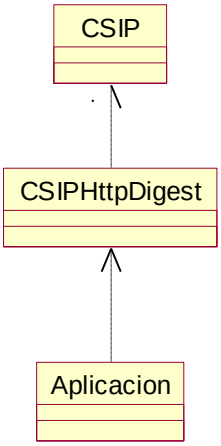

Figura 9. Clase CSIPHttpDigest.

D2.8 Uso de las Clases del API SIP para Symbian

D2.8.1 Iniciación de un Cliente SIP

Una aplicación que usa el API de Cliente SIP crea primero una instancia de CSIP. Cuando CSIP se crea, el cliente se conecta a un servidor. La aplicación crea una instancia de CSIPConnection para poder usar un IAP. La aplicación tiene que implementar los métodos de observación de MSIPConnectionObserver. Cuando

CSIPConnection se crea, una sesión alterna es formada entre el cliente y el servidor. La función CSIPConnection::NewL () es asíncrona. Cuando la conexión es creada y la conexión es activada, las funciones de observación MSIPConnectionObserver::ConnectionStateChanged(CSIPConnection:: EActive) son llamadas. Cuando el estado es CSIPConnection::EActive, el objeto puede usarse para enviar mensajes SIP, crear un registro y diálogos. El estado de la conexión puede ser obtenido mediante CSIPConnection::State ().

La figura 10 muestra el diagrama de secuencia para la Iniciación del cliente SIP.

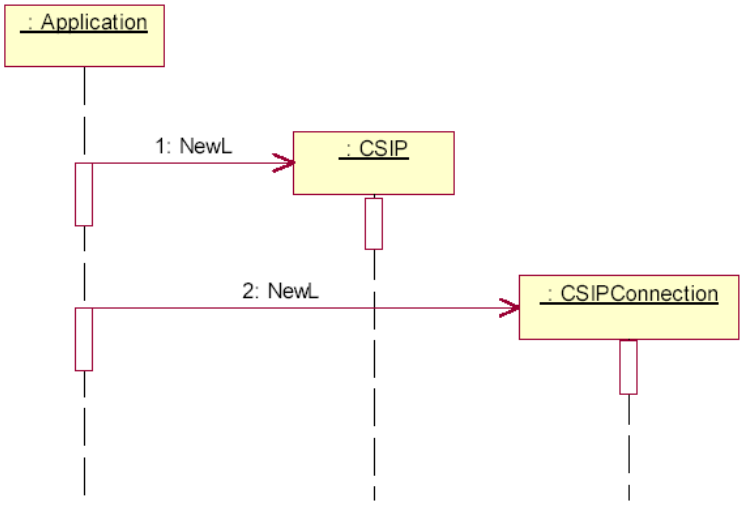

Figura 10. Iniciación del cliente SIP.

D2.8.2 Cerrar un Cliente SIP

La aplicación puede eliminar cuando quiera cualquier objeto que tenga del API.

Cuando se elimina la instancia de CSIP, la conexión al servidor SIP se cierra. El servidor puede ser detenido en ese punto. Eliminando CSIPConnection causa que la sub-sesión entre el cliente y el servidor sea cerrada.

Si la aplicación elimina un objeto con el cual existen dependencias a otro objeto, puede inutilizar a los otros objetos. Los objetos que están en un estado inutilizable retornaran el código de error KErrSIPResourceNotAvailable, indicando que no pueden realizar la operación solicitada por que ya no existe acceso al objeto eliminado.

Un ejemplo del escenario anterior es cuando una aplicación tiene objetos CSIPConnection y CSIPRegistrationBinding y se elimina CSIPConnection. Usando CSIPRegistrationBinding después de eso en la mayoría de los casos se produce una falla, porque sin CSIPConnection, no hay manera alguna para que CSIPRegistrationBinding se comunique con el lado del servidor de la pila SIP. La figura 11 muestra el orden recomendado para anular los objetos del API de Cliente SIP.

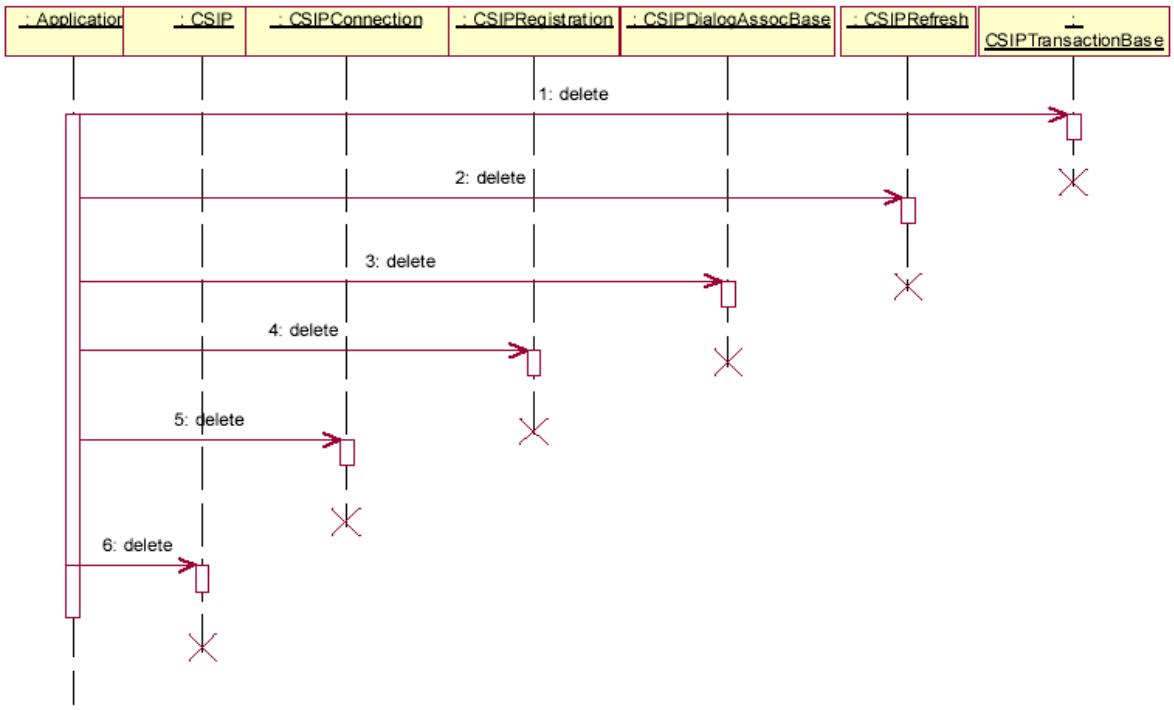

Figura 11. Orden recomendado para anular los objetos del API de Cliente SIP.

D2.8.3 Envío de Solicitudes SIP

Para enviar una solicitud SIP, la aplicación crea una instancia de CSIPRequestElements, y establece las partes deseadas de la solicitud. Una vez CSIPRequestElements se ha llenado, la aplicación invoca la función CSIPConnection::SendRequestL(CSIPRequestElements). CSIPConnection toma la propiedad de CSIPRequestElements y comunica la solicitud al servidor de la pila SIP. CSIPConnection instancia un CSIPClientTransaction. SendRequestL comienza el procesamiento del servidor de envío del mensaje, pero por el tiempo que consumen las tareas de resolución de dirección y las operaciones de conexión, este solo es realizado después de que la llamada de SendRequestL ha retornado. Si un error ocurriera después de que SendRequestL ha vuelto, por ejemplo en la resolución de dirección, el error se comunica a la aplicación por medio de MSIPConnectionObserver::ErrorOccured. Es importante entender que cuando SendRequestL retorna, el mensaje no ha sido enviado todavía a la red pero ha estado procesándose en el lado del servidor de la pila SIP.

Cuando el mensaje de respuesta SIP es recibido del lado del servidor, CSIPConnection forma un objeto CSIPResponseElements para representar la respuesta SIP, lo enlaza a CSIPClientTransaction y lo notifica a la aplicación con MSIPConnectionObserver:: IncomingResponse. La aplicación puede determinar qué tipo de respuesta se recibió accediendo a CSIPResponseElements de CSIPClientTransaction. Si la respuesta era una respuesta provisional, la aplicación no anula el CSIPClientTransaction todavía. Después de recibir una respuesta 200 la aplicación elimina el CSIPClientTransaction quedando completa la transacción.

La figura 12 ilustra los pasos para enviar una solicitud de MESSAJE y recibir dos respuestas a la solicitud.

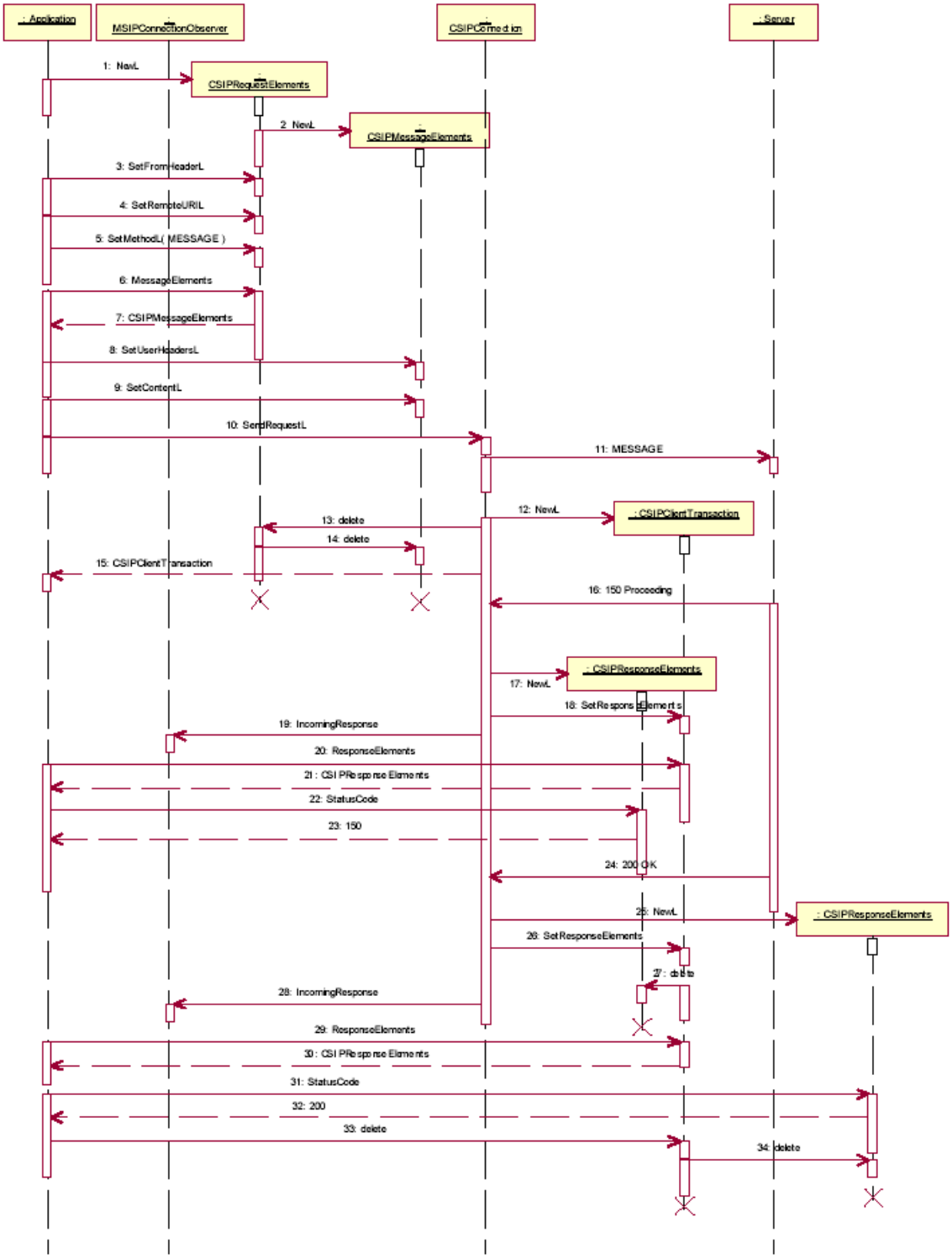

Figura 12. Enviando una solicitud SIP.

D2.8.4 Recibir una Solicitud SIP

En este ejemplo, una solicitud SIP se recibe a través de un CSIPConnection existente. CSIPConnection crea CSIPRequestElements y un CSIPServerTransaction para representar la solicitud SIP entrante, antes de notificar a la aplicación con un evento de IncomingRequest a MSIPConnectionObserver. La aplicación consigue información de la solicitud extrayendo CSIPRequestElements de CSIPServerTransaction. Cuando una respuesta es enviada, CSIPResponseElements necesita ser instanciado y pasada a CSIPServerTransaction con la función de SendResponseL. Esto produce que la respuesta sea manejada por el servidor para futuros procesos. Cuando SendResponseL regresa, la aplicación no necesitara más de CSIPServerTransaction y la elimina. La figura 13 muestra los pasos de recepción de un MESSAGE y envío de una respuesta.

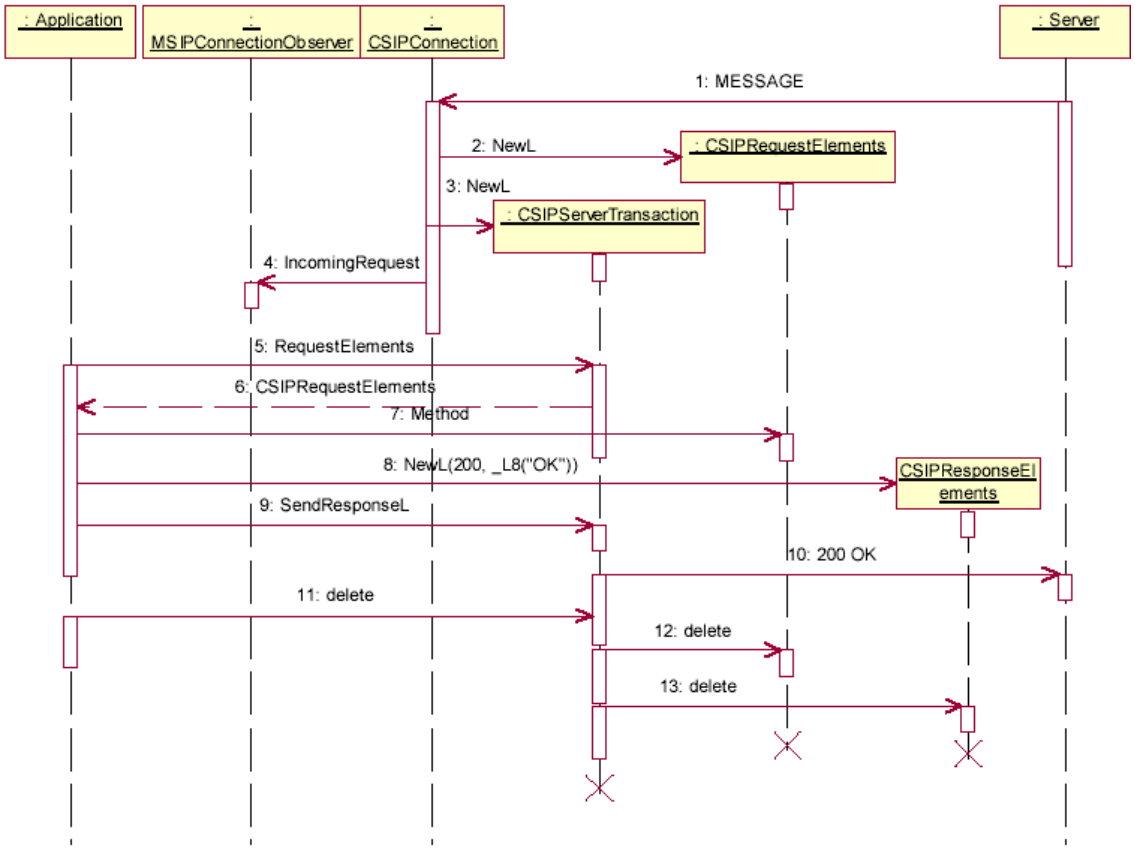

Figura 13. Recibiendo una solicitud SIP.

D2.8.5 Recibir una Solicitud SIP y Crear una Sesión

El UAS recibe una solicitud INVITE de la red y crea una transacción de servidor INVITE para la solicitud recibida. Seguido a esto se envía una respuesta SIP 100 al UA remoto. Después de comparar la solicitud recibida con las capacidades de aplicación almacenadas, la pila SIP enruta la solicitud INVITE recibida a la aplicación escogida. La aplicación crea una instancia de CSIPInviteDialogAssoc y crea una instancia de CSIPDialog. La aplicación envía una respuesta SIP 200 al UA remoto. El UA remoto reconoce la respuesta SIP 200 y responde con un ACK, luego una sesión SIP se crea entre el UA local y el remoto.

D2.8.6 Proceso de Registro SIP

Para comenzar un registro, la aplicación crea primero un objeto CSIPRefresh, y luego un CSIPRegistrationBinding, pasándole el CSIPRefresh. No es obligatorio crear un CSIPRefresh; sólo se necesita si el registro será refrescado por la pila SIP. Una vez CSIPRegistrationBinding existe, la aplicación puede comenzar el registro llamando a RegisterL. CSIPRegistrationBinding forma una solicitud REGISTER y la comunica al servidor de la pila SIP, se instancia CSIPClientTransaction y se lo retorna a la aplicación.

Cuando la respuesta 200 es recibida por el servidor, CSIPConnection crea CSIPResponseElements para mantener la respuesta y enruta la respuesta a CSIPRegistrationBinding. CSIPRegistrationBinding enlaza la respuesta a CSIPClientTransaction. La aplicación es notificada acerca de la respuesta con MSIPConnectionObserver::IncomingResponse. La aplicación ya no necesita de CSIPClientTransaction y lo elimina. Finalmente se usa IsContextActive para averiguar si el registro ha tenido éxito.

La figura 14 muestra los pasos de registro con un refresco y recepción de una respuesta 200 (OK).

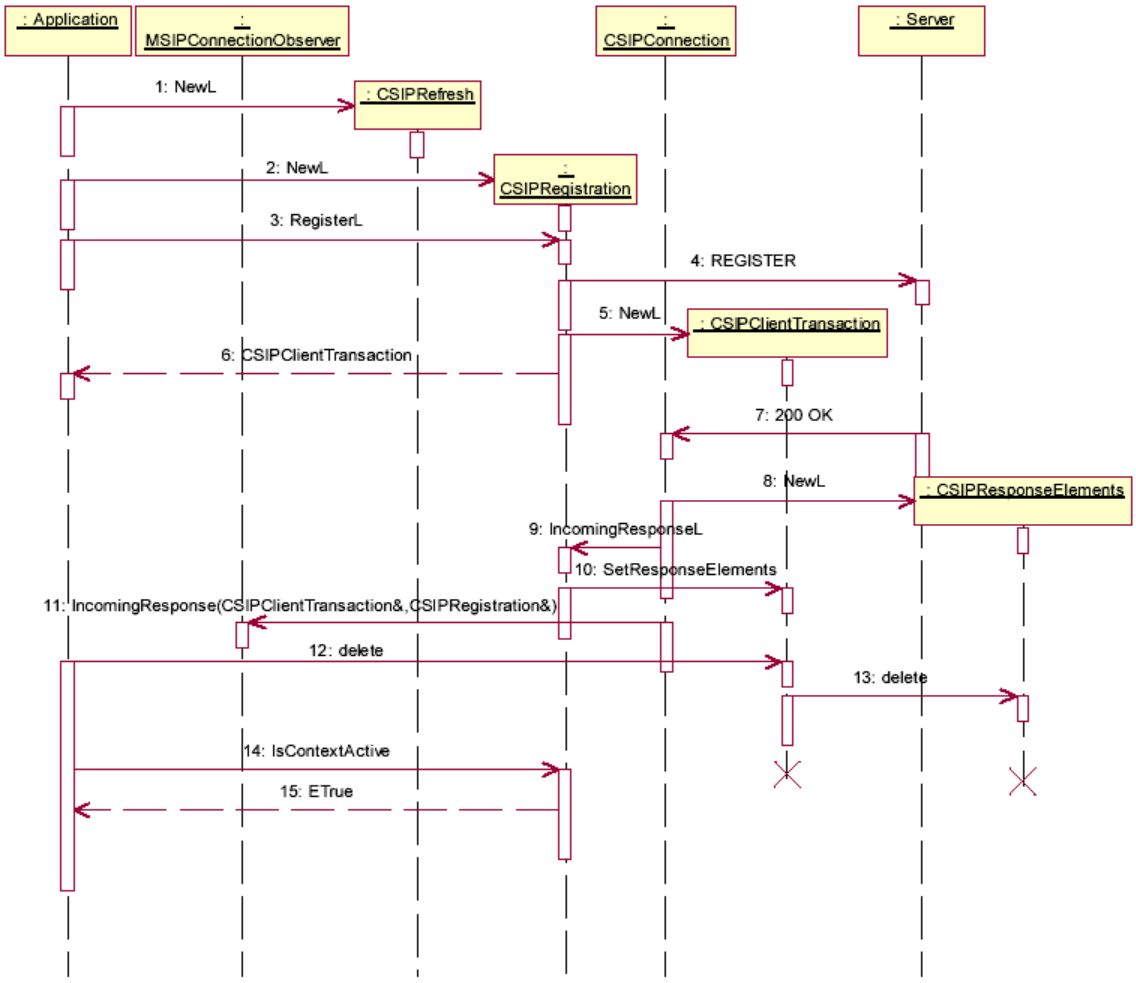

Figura 14. Registro con refresco.

D2.8.7 Creación de un Dialogo con INVITE

Para iniciar un diálogo, la aplicación crea los objetos CSIPInviteDialogAssoc y CSIPMessageElements: Luego llena las cabeceras SIP pertinentes en CSIPMessageElements antes de pasarlo a CSIPInviteDialogAssoc usando la función SendInviteL. Con CSIPInviteDialogAssoc forma la solicitud INVITE y la comunica al servidor de la pila SIP, instancia CSIPClientTransaction y lo retorna a la aplicación.

Cuando se recibe una respuesta 180 en el servidor, CSIPConnection crea un CSIPResponseElements para mantener la respuesta y enruta la respuesta a CSIPDialog. CSIPDialog enlaza la respuesta a CSIPClientTransaction. La aplicación es

informada sobre la recepción de la respuesta 180 con MSIPConnectionObserver:: IncomingResponse.

Cuando se recibe la respuesta 200, un proceso similar ocurre como con la respuesta 180. A estas alturas, se espera que la aplicación responda enviando una solicitud de ACK. Esto se realiza invocando la función CSIPInviteDialogAssoc::SendAckL que genera un ACK y lo envía servidor de la pila SIP. La figura 15 muestra los pasos de enviar un INVITE, recibir una respuesta y enviar un ACK.

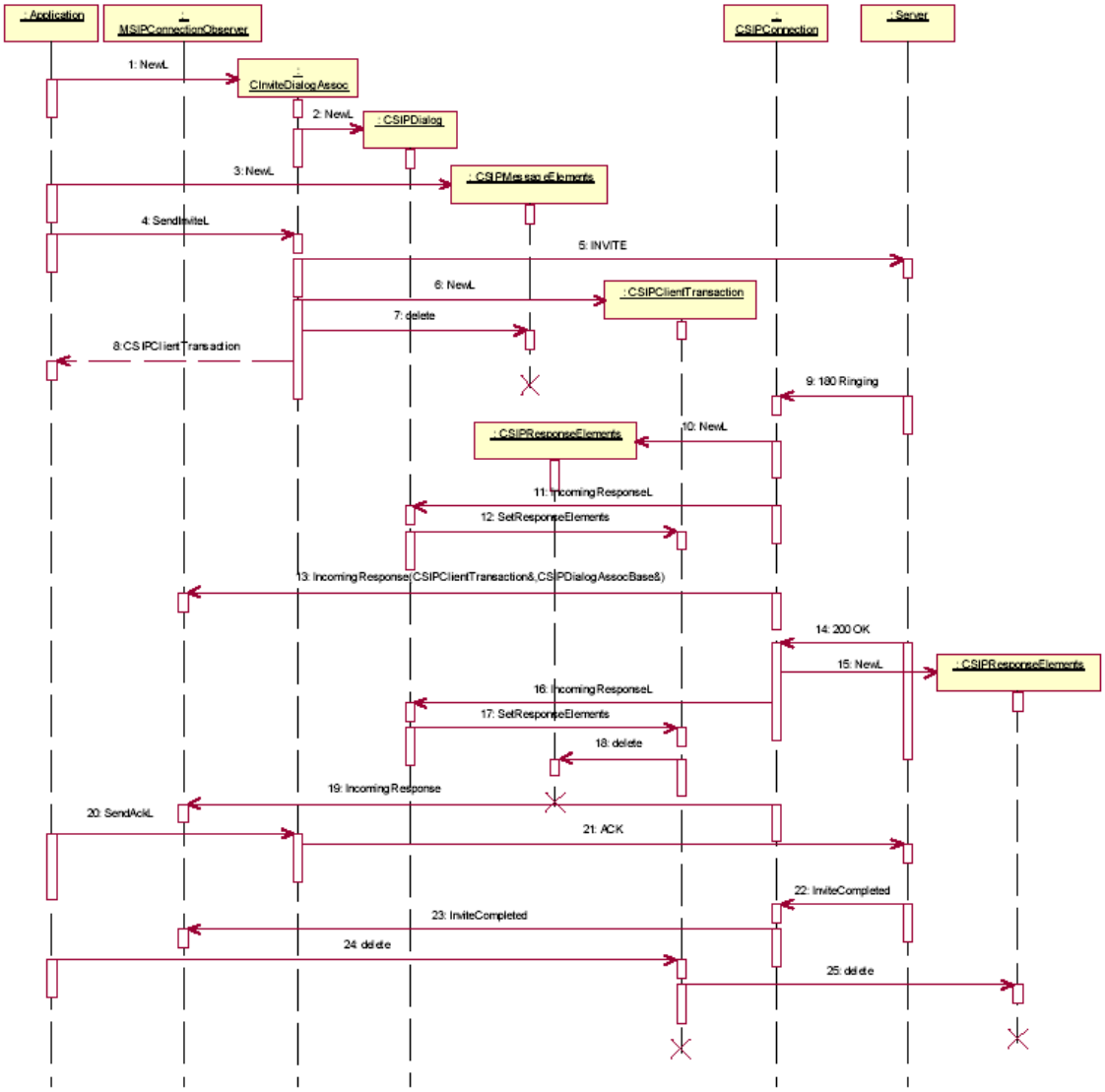

Figura 15. Creación de un dialogo con el envío de un INVITE.

D2.8.8 Generación de Respuestas 100

No se permiten a las aplicaciones enviar respuestas 100. La pila SIP genera la respuesta 100 automáticamente cuando recibe una solicitud INVITE de la red.

D2.8.9 Respuesta a una Solicitud SIP CANCEL

Cuando la pila SIP recibe una solicitud CANCEL de la red, automáticamente responde a ella. Nunca se pasan las solicitudes de cancelación a la aplicación. Si la pila SIP no encuentra una transacción concordante para la cancelación, genera automáticamente una respuesta 481 para el CANCEL. Si la pila SIP encuentra una transacción que coincide para la cancelación, genera una respuesta 200 para el CANCEL. Si la pila SIP encuentra una transacción coincidente que está en proceso (no se ha enviado aun una respuesta final), genera una respuesta 487 también automáticamente para el INVITE. Luego la pila SIP informa de esto a la aplicación usando el método MSIPConnectionObserver::InviteCanceled.

D2.9 Ejemplo de Aplicación SIP con Symbian

El API de Cliente SIP inicia creando los objetos CSIP y CSIPConnection:

```
iSIP = CSIP::NewL(KAppUid, *iMySIPObserver);
iConnection = CSIPConnection::NewL(*iSIP, KiapIdentifier,
                                   *iMyConnectionObserver);
```

Para publicar la dirección de contacto SIP de un usuario, se crea un registro SIP. Esto requiere de un AOR, un contacto a ser registrado y si se desea como en este ejemplo, de un CSIPRefresh para tener un registro que sea refrescado automáticamente por la pila SIP.

En el siguiente ejemplo, “localhost” debe ser reemplazado con la dirección IP local.

```
CSIPToHeader* aor =
CSIPToHeader::DecodeL(_L8("sip:user@remote.registrar"));
CleanupStack::PushL(aor);
CSIPAddress* addr = CSIPAddress::DecodeL(_L8("sip:user@LOCALHOST"));
CleanupStack::PushL(addr);
CSIPContactHeader* contact = CSIPContactHeader::NewL(addr);
CleanupStack::Pop(addr);
CleanupStack::PushL(contact);
CSIPRefresh* refresh = CSIPRefresh::NewLC();
iRegistration = CSIPRegistrationBinding::NewL(*iConnection, aor, contact,
refresh);
CleanupStack::Pop(3); //refresco, contacto, aor
```

Una vez CSIPRegistrationBinding se ha creado, una transacción de REGISTRO puede ser iniciada.

```
iRegisterTransaction = iRegistration->RegisterL();
```

Cuando se recibe una respuesta 2xx, indica que el registro ha tenido éxito. Cuando se reciba una respuesta final, la transacción puede ser eliminada.

```
Void CmyConnectionObserver::IncomingResponse(CSIPClientTransaction&
aTransaction,
                    CSIPRegistrationBinding& aRegistration)
  {
  if (aRegistration.IsContextActive())
    {
    // El registro ha tenido éxito
```

```
    }
  if (aTransaction.ResponseElements()->StatusCode() >= 200)
    {
    // La respuesta final es recibida, la transacción ya no se necesita
    if (aTransaction == *iRegisterTransaction)
      {
      delete iRegisterTransaction;
      iRegisterTransaction = NULL;
      }
    }
```

Una solicitud INVITE es recibida de la red. Una instancia de CSIPConnection se crea para este IAP.

```
Void CMySIPObserver::IncomingRequest(Tuint32 aIapId,
CSIPServerTransaction* aTransaction)
  {
  // Se crea un observador para el nuevo CSIPConnection
  TRAPD(err, iMyOtherConnectionObserver =
CmyConnectionObserver::NewL());
  // Se crea una nueva conexión para el IAP
  TRAP(err, iOtherConnection = CSIPConnection::NewL(*iSIP, aIapId,
                    *iMyOtherConnectionObserver));
  // La aplicación posee la transacción
  iInviteServerTransaction = aTransaction;
  }
```

Como la solicitud recibida es un INVITE, y la aplicación desea enviar una respuesta (101-299) que creará un diálogo SIP, CSIPInviteDialogAssoc debe ser creado antes de enviar la respuesta.

```
iInviteAssoc = CSIPInviteDialogAssoc::NewL(*iInviteServerTransaction);

CSIPResponseElements* response = CSIPResponseElements::NewLC(180,
“Ringing”);

iInviteServerTransaction->SendResponseL(response);

CleanupStack::Pop(response);
```

Cuando el usuario ha aceptado el INVITE, una respuesta 200 que usa el mismo objeto de la transacción es enviado. El envió de una respuesta 2xx a un INVITE causa una transacción entrando en el estado CSIPTransactionBase::Eterminated, ahora la transacción puede ser eliminada.

```
CSIPResponseElements* response = CSIPResponseElements::NewLC(200,
“OK”);

iInviteServerTransaction->SendResponseL(response);

CleanupStack::Pop(response);

delete iInviteServerTransaction;

iInviteServerTransaction = NULL;
```

Cuando la solicitud de ACK llega, el API de Cliente SIP crea un nuevo CSIPServerTransaction y pasa el objeto de la transacción a la aplicación que usa MSIPConnectionObserver del IAP a través del ACK que fue recibido. Ninguna respuesta se envía a un ACK, para que el objeto de la transacción pueda ser eliminado.

```
Void CmyConnectionObserver::IncomingRequest(CSIPServerTransaction*
aTransaction,
                                CSIPDialog& aDialog)
  {
  if (aTransaction.Type() == SIPStrings::StringF(SipStrConsts::Eack))
    {
    // Puede inspeccionar la parte del contenido SDP del ACK.
    // No se enviara respuesta al ACK
    delete aTransaction;
    }
  else
    {
    // Manejo de otras solicitudes
    }
  }
```

El siguiente es un ejemplo de definición de cabeceras de extensión con el API de Cliente SIP.

```
Void CmyClass::ExtractHeadersL( CSIPMessageElements& aElements )
  {
  const RpointerArray<CSIPHeaderBase>& userHeaders =
    aElements.UserHeaders();
  for ( Tint i = 0; i < userHeaders.Count(); i++ )
    {
     const CSIPHeaderBase* header = userHeaders[ i ];
```

```
// Un ejemplo de una cabecera ya soportada conocida.
// Éste puede ser moldeado a la clase actual.
// También será soportada en el futuro y no causará los problemas de
// SC/BC.
If ( header->Name() == SIPStrings::StringF( SipStrConsts::ERAckHeader))
   {
   iRAckHeader = static_cast<CSIPRAckHeader*>(header->CloneL());
   }

// Un ejemplo de una cabecera SIP que es actualmente soportada como
// una extensión.
// La aplicación no debe hacer un lanzamiento estático a
// CSIPExtensionHeader.
// En cambio debe usar CSIPHeaderBase::ToTextValueL y debe analizar el
//  resultado.
RstringF extensionName = SIPStrings::Pool().OpenFStringL(
L8("Extension") );
CleanupClosePushL( extensionName );
if ( header->Name() == extensionName )
   {
   HbufC8* headerValue = header->ToTextValueLC();

   // Hacer el análisis gramatical específico y el manejo para la cabecera

   CleanupStack::PopAndDestroy( headerValue );
```

```
        }
    CleanupStack::PopAndDestroy( 1 ); // extensionName
    }
} [3][4]
```

BIBLIOGRAFÍA

[1] SIP API for Java™ 2 Micro Edition Version 1.0.1 JSR 180 Expert Group. Diciembre de 2004. http://www.microjava.com/nokia/documents?content_id=7848

[2] Nokia. SIP API for Java™ 2 Micro Edition (JSR-180): Implementation Notes Version 1.0. Junio de 2006. http://www.forum.nokia.com/ME_Developers_Library/GUID-9D6A8CE9-C288-463B-AF37-CB9B42C39636.pdf

[3] Nokia. S60 3rd Edition SDK for Symbian OS API Reference Guide [Directorio de instalación del SDK 3.0]\Symbian\9.1\S60_3rd\S60Doc\ s60_cpp_sdk_3rd_apirefguide.chm

[4] Nokia. S60 3rd Edition SDK for Symbian OS API Reference Guide. [Directorio de instalación del SDK3.0]\Symbian\9.1\S60_3rd\S60Doc\S60 3rd Edition API Referente.