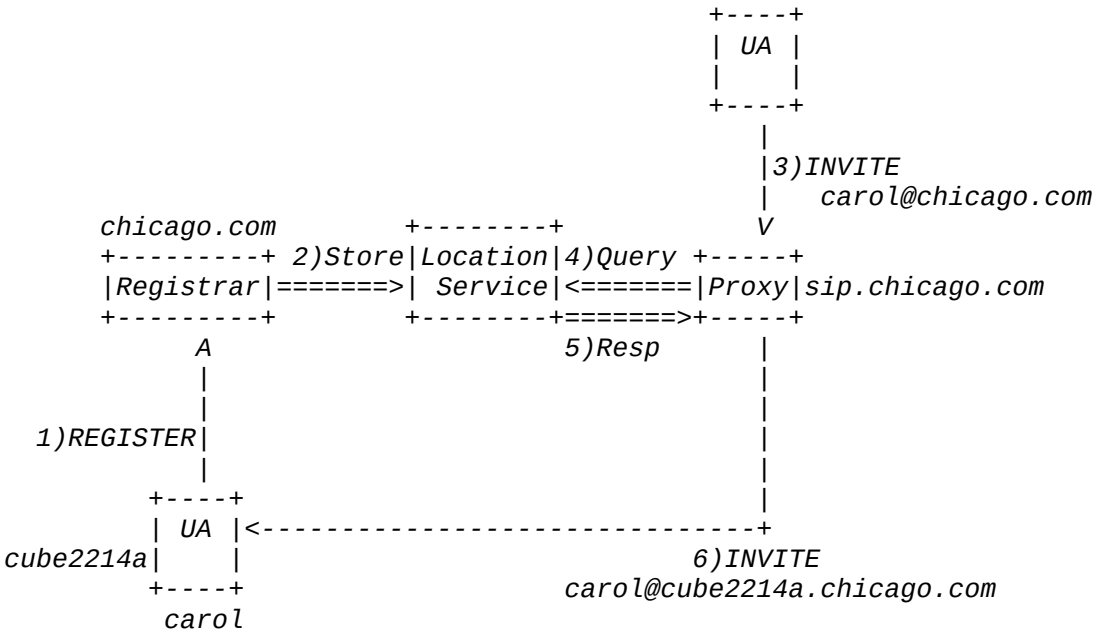
Agente de usuario y proxy SIP

El objetivo de esta primera práctica es el desarrollo de un agente de usuario y de un proxy SIP que permitan implementar el escenario siguiente:



En nuestro caso los servidores de registro y localización estarán ubicados en el Proxy (de esta forma simplificaremos la implementación y facilitaremos el desarrollo del servicio de localización que podrá limitarse a una tabla en el interior del servidor SIP implementado). El escenario de llamada tendrá un único proxy SIP de forma que se implementarán servicios de llamadas internas a un dominio (equivalente a una centralita telefónica para llamadas internas). De esta forma también evitaremos la necesidad de usar el servicio de DNS para la resolución del proxy SIP destino. También para simplificar el desarrollo de la práctica no permitiremos que el usuario se registre desde dos terminales distintos de cara a no tener que hacer "fork" de los mensajes de INVITE. Si un usuario intenta un registro desde un segundo terminal, esto provocará que se anule automáticamente el registro del terminal anterior en el proxy.

De cara a simplificar también la implementación de la pila de protocolos SIP, los únicos mensajes SIP a intercambiar serán: REGISTER, INVITE, ACK y BYE. Con estos mensajes será suficiente para implementar un servicio de establecimiento de sesiones SIP completo.

Igualmente, realizaremos una implementación simplificada de las posibles causas que originen posibles respuestas a los mensajes anteriores SIP. Las únicas respuestas a estos mensajes serán en nuestra implementación:

- *100 Trying*
- *180 Ringing*
- *200 OK*
- *404 Not Found (el usuario no está registrado)*
- *408 Request Timeout (el usuario no contesta a la llamada al cabo de 10 segundos de hacer sonar el terminal a través del que se ha registrado el mismo)*
- *486 Busy Here (usuario se encuentra ya en una llamada activa o rechaza la nueva llamada)*

- *503 Service Unavailable. Este será el único error de servidor y se usará para notar que nuestro proxy SIP solo admite una llamada concurrentemente. Si en el procesamiento de un INVITE llegara un segundo INVITE con diferente Call-ID, se devolverá este error. Como veremos más adelante, si se usa el mecanismo de loose routing, el servidor proxy estará en una llamada desde que recibe el mensaje de INVITE hasta que se confirma con un 200 OK el BYE. Si no se usa el mecanismo de loose routing, el servidor quedará libre para atender nuevas llamadas finalizado el flujo de procesamiento del INVITE con un ACK en caso de error o con un 200 OK en caso de llamada exitosa.*

Usaremos como lenguaje de programación para todas las prácticas de esta parte de la asignatura Java. Podéis usar la versión de J2SE instalada en los laboratorios u otra distinta pero tened en cuenta que la evaluación se hará sobre las máquinas del laboratorio. Las máquinas del laboratorio tienen instalado también Eclipse/Netbeans para Java que proporciona un entorno gráfico e intuitivo con el que desarrollar, compilar, ejecutar y depurar que puede ser usado tanto como entorno para la realización de las prácticas como para su defensa el día de la evaluación.

Entre las opciones a implementar se contará con un modo de depuración que muestre por consola de usuario las trazas de los mensajes enviados y los mensajes recibidos con lo que podéis validar el correcto funcionamiento de vuestra aplicación y podéis mostrar el funcionamiento de vuestra práctica el día de la evaluación. Para detectar los puntos de error de vuestro código a la hora de realizar la práctica, podéis usar también el entorno de depuración que ofrece el IDE elegido. En cualquier caso y de cara a poder comprobar que la práctica implementada cumple con las especificaciones de este enunciado, tanto el servidor proxy como el agente de usuario a implementar ofrecerán por línea de comando la posibilidad de establecer un modo debug como se especifica más abajo.

Aunque el puerto de escucha estándar (por defecto) para comunicaciones no seguras de SIP es el 5060, podéis jugar con los puertos para permitir que tanto los agentes de usuario como el proxy se ejecuten en una misma máquina (por ejemplo, el proxy puede correr en el puerto estándar mientras que los agentes de usuario especificarán un puerto distinto en el mensaje de registro). Como sabéis SIP utiliza de forma estándar los puertos 5060 para comunicaciones no seguras y 5061 para comunicaciones seguras. Aunque la señalización SIP podría usar TCP, para la práctica nosotros solamente usaremos UDP.

El agente de usuario será una aplicación java UA.java con la siguiente línea de comando:

java UA usuarioSIP puertoescuchaUA IPProxy puertoescuchaProxy debug(true/false)

Donde la opción de debug servirá para que los mensajes que se intercambian aparezcan con todas las cabeceras en pantalla o solo con la línea de petición/respuesta. El usuarioSIP que se introduce por línea de comandos tendrá el formato:

nombreUsuario@dominio

de esta forma, el agente de usuario conocerá tanto el detalle de qué usuario ha arrancado la aplicación así como el dominio al que responderá el proxy SIP.

Una vez arrancado el agente de usuario creará un DatagramSocket asociado al puertoescuchaUA y lanzará un hilo que escuche DatagramPackets en dicho DatagramSocket. El hilo principal de ejecución atenderá a la interfaz de usuario (System.in.read). Al arrancar el agente de usuario registrará automáticamente el usuario en el proxy. En caso de recibir un mensaje de OK del proxy se permitirá hacer llamadas al usuario. Si se recibe un mensaje 404 del proxy porque éste no autoriza el registro del usuario la aplicación de agente de usuario finalizará. En caso que tras 2 segundos de haber enviado el mensaje REGISTER al proxy no hayamos obtenido respuesta de éste, volveremos a enviar el mensaje de REGISTER y así de forma indefinida hasta que el proxy nos responda. Una vez que se ha conseguido registrar el usuario en el proxy el agente de usuario tendrá dos líneas de ejecución concurrentes de las

que atenderá el teclado y los mensajes SIP que llegan por la red. Para poder hacer una llamada se introducirá por teclado:

INVITE usuarioSIPDestino

donde usuarioSIPDestino solo contendrá el nombre del llamado, sin incluir el dominio del mismo pues para la presente práctica todas las llamadas serán dentro de un mismo dominio.

El procesamiento de mensajes que lleguen al DatagramSocket distinguirá si es una llamada entrante o saliente y seguirá el diagrama de estados simplificado que se recoge en el siguiente apartado. Es obligatorio saber si el agente de usuario se encuentra en una llamada o no, si está en una llamada si es llamante o llamado y además, dentro del procesamiento del mensaje INVITE es necesario saber en qué estado se encuentra de cara al correcto tratamiento de los mensajes SIP. ***El agente de usuario mostrará mensajes por pantalla durante el flujo de procesamiento del mensaje INVITE cada vez que cambie el estado dentro del procesamiento de la transacción****.*

Una llamada entrante (recepción de un mensaje INVITE desde el Proxy) avisará al usuario mediante un mensaje en pantalla y podrá ser respondida o rechazada por el usuario. Es decir, al recibir un INVITE se responderá con un 200 OK si se descuelga la llamada o con un 408 si pasados 10 segundos no se responde la llamada o con un 486 si se rechaza la llamada o llega un INVITE con un nuevo Call-ID cuando se está procesando el INVITE de una llamada previa o dicha llamada está en curso y no ha sido aún colgada.

El agente de usuario deberá mostrar por pantalla (System.out.println) todos los mensajes recibidos. Si se activa el modo debug se debe mostrar el detalle con todas las cabeceras de los mensajes. Si no se activa el modo debug será suficiente con mostrar la primera línea de cada mensaje. La activación del modo debug se pasará por línea de comando como se ha comentado anteriormente.

El proxy será también una aplicación Java que se lanzará:

java Proxy puertoescuchaProxy loose-routing(true/false) debug(true/false)

Donde la opción de loose-routing (que solo podrá valer true o false) se usará para especificar si se quiere que el proxy añada la cabecera Record-route o no y la opción de debug servirá para que los mensajes que se intercambian aparezcan con todas las cabeceras en pantalla o solo con la línea de petición/respuesta.

Se permitirá arrancar el Proxy sin parar opciones por línea de comandos. En este caso, el comportamiento por defecto será que el puertoescuchaProxy será el estándar de SIP (5060) y se activará tanto loose-routing como debug. Por simplicidad, solo se atenderá el establecimiento de una llamada simultáneamente (de forma que si se está procesando un INVITE y llega un mensaje con diferente call-ID, se contestará con un mensaje 503 Service Unavailable). En el caso de tener el loose-routing activo, se responderá con un 503 Service Unavailable ante un nuevo INVITE con diferente Call-ID también durante el curso de la llamada hasta procesar el mensaje de 200 OK como respuesta al BYE.

El proxy siempre aceptará la llamada si ambos usuarios están registrados y contestará con un 404 si alguno de los usuarios (llamante o llamada) no están registrados. De cara a gestionar el control de acceso al sistema, el Proxy tendrá configurada una lista de usuarios válidos que pueden registrarse en el sistema. Si se intenta el registro de un usuario válido, responderá con un 200 OK pero si el registro lo intenta un usuario no válido, se responderá con un 404.

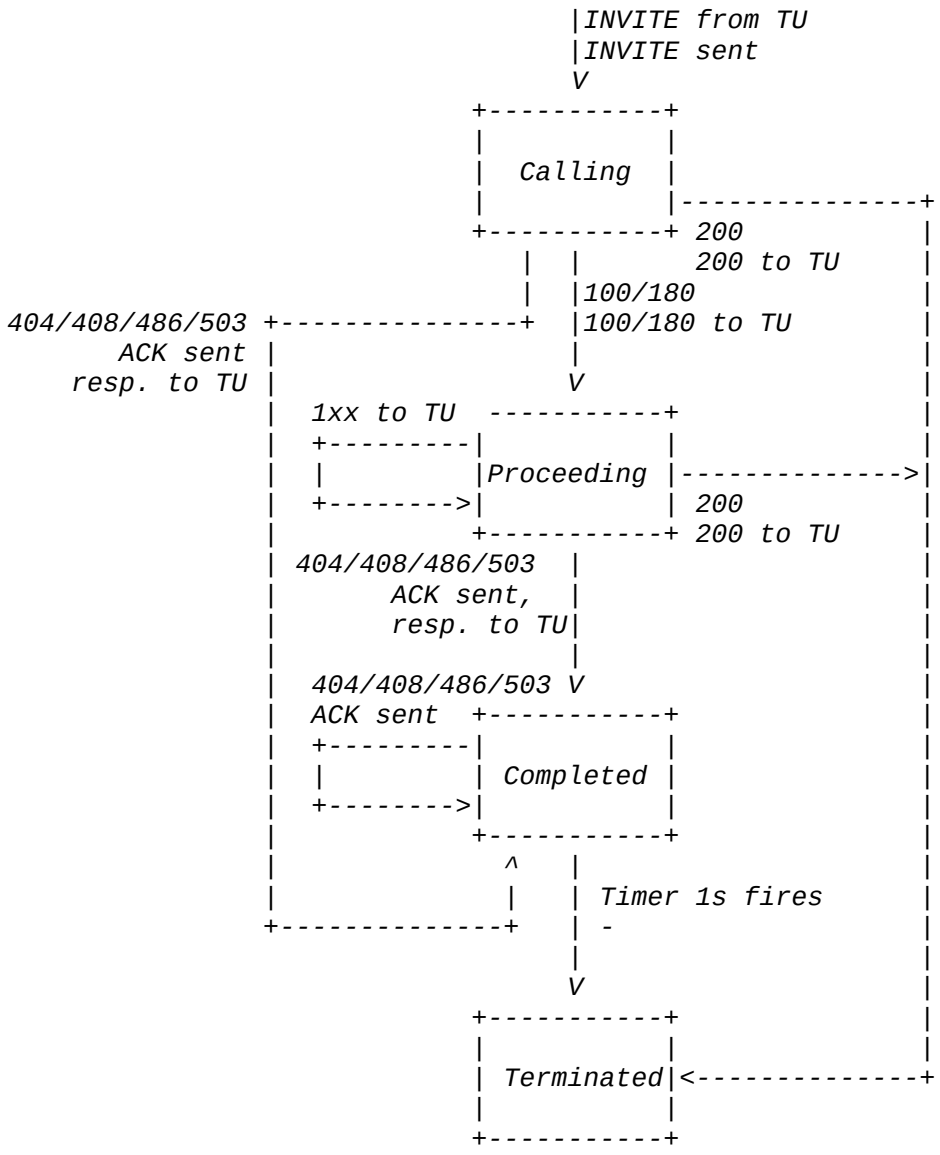
Si la opción de loose-routing se ha especificado a false, el proxy no procesará el BYE pues irá extremo a extremo. En caso contrario, el BYE pasará por el proxy. En ningún caso la voz deberá ir por el proxy, solo los mensajes de señalización SIP.

Nótese que los mensajes INVITE llevarán como carga útil SDP para describir el contenido de la sesión de voz. El mensaje 200 OK de respuesta al INVITE llevará también carga SDP. El resto de mensajes SIP no llevarán carga SDP. El objetivo será poder especificar el puerto al que enviar la voz desde el otro extremo de cara a poder unir la señalización SIP de esta práctica con el software desarrollado en la primera parte de la asignatura.

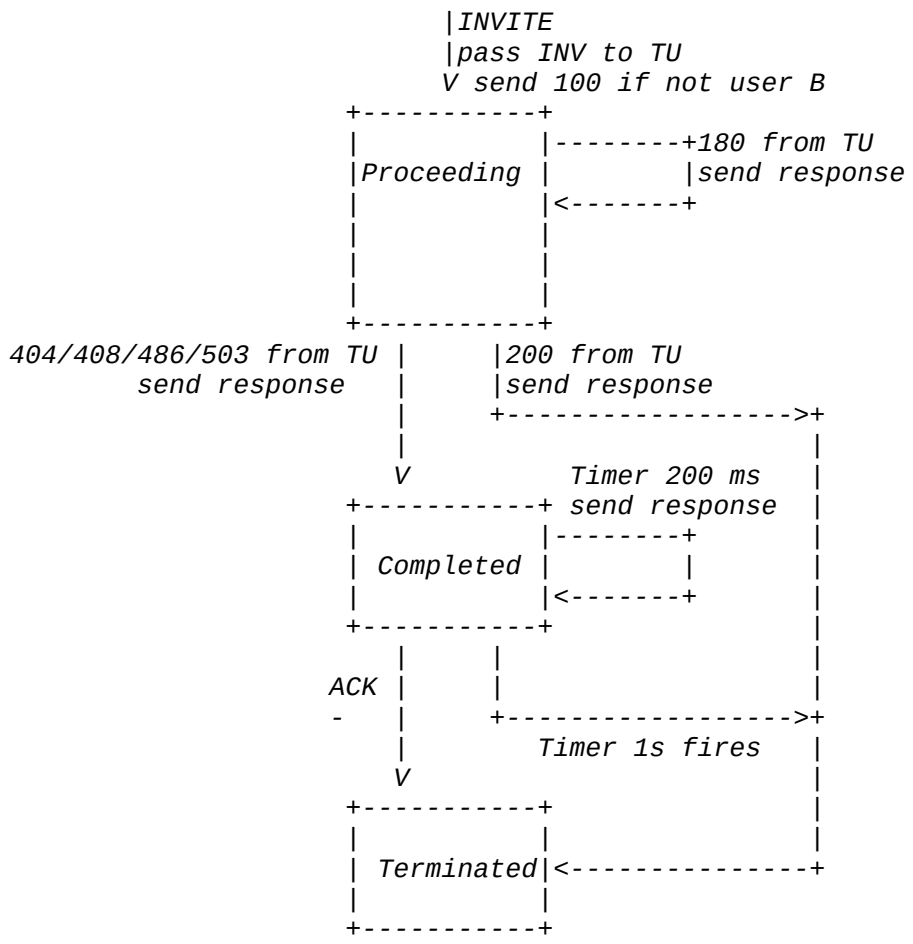
De cara a simplificar el trabajo de desarrollo de la práctica, ***se proporciona un API de manejo de mensajes SIP de obligatorio uso****. Este API se describe más adelante en este enunciado. El API implementa las simplificaciones en el formato de los mensajes SIP comentadas en clase de cara a facilitar la implementación de la práctica (por ejemplo, no se tratan las etiquetas que SIP añade a las Vias, direcciones del llamado y llamante, etc).*

Estados y temporizadores

De cara a simplificar la lógica de control de llamada, para el caso del procesamiento del INVITE solo se deben implementar los siguientes códigos de respuesta con los siguientes temporizadores. Para la máquina de estados de procesamiento de la transacción del llamante (donde TU significa Transaction User en su comunicación con la capa de transacción definida en la RFC 3261):



Para el llamado se tendrá (ahora el flujo lo origina la recepción de un nuevo mensaje de INVITE con un nuevo Call-ID, lo cual desencadena que se notifique de los mensajes que se van recibiendo al usuario de la transacción al tiempo que se generen los mensajes SIP apropiados y se gestione el cambio de estado en la capa de transacción):



Para la gestión de los temporizadores se puede usar Timer y TimerTask del paquete java.util o bien la clase javax.swing.Timer. Un ejemplo sencillo con esta última clase podría ser:

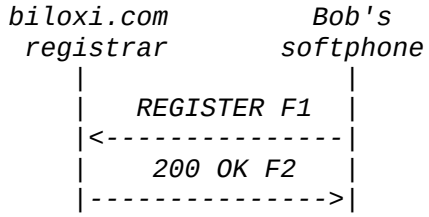
```
Timer timer = new Timer (tiempoEsperaEnMilisegundos, new ActionListener ()
{
   public void actionPerformed(ActionEvent e)
   {
      // Aquí el código que queramos ejecutar.
    }
});
...
timer.start();
```

Los temporizadores se pueden parar si antes de que venzan llega el mensaje que estamos esperando.

La RFC también detalla las diferentes cabeceras, su uso, si son de implementación obligatoria, etc. Todos estos detalles los dejaremos de lado en esta asignatura y solo requeriremos una implementación limitada del protocolo. En las siguientes secciones se enuncian algunos ejemplos de los mensajes a intercambiar de cara a facilitar la implementación de la práctica.

Registro

Como se recoge en la RFC se tiene:



F1 REGISTER Bob -> Registrar

```
REGISTER sip:registrar.biloxi.com SIP/2.0
Via: SIP/2.0/UDP bobspc.biloxi.com:5060
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Bob <sip:bob@biloxi.com>
Call-ID: 843817637684230@998sdasdh09
CSeq: 1826 REGISTER
Contact: <sip:bob@192.0.2.4>
Expires: 7200
Content-Length: 0
```

F2 200 OK Registrar -> Bob

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP bobspc.biloxi.com:5060
To: Bob <sip:bob@biloxi.com>
From: Bob <sip:bob@biloxi.com>
Call-ID: 843817637684230@998sdasdh09
CSeq: 1826 REGISTER
Contact: <sip:bob@192.0.2.4>
Expires: 7200
Content-Length: 0
```

O bien la respuesta puede ser de no encontrar al usuario entre los clientes del dominio:

```
SIP/2.0 404 Not Found
Via: SIP/2.0/UDP bobspc.biloxi.com:5060
To: Bob <sip:bob@biloxi.com>
From: Bob <sip:bob@biloxi.com>
Call-ID: 843817637684230@998sdasdh09
CSeq: 1826 REGISTER
Contact: <sip:bob@192.0.2.4>
Expires: 7200
Content-Length: 0
```

Notad que para implementar lo número de identificación de llamada y transacción podéis usar un simple contador de forma que se vayan incrementando a cada llamada y a cada transacción dentro de la llamada. El Call-ID según la RFC 3261 debería llevar a la derecha de la @ la dirección del llamado de cara a desambiguar identificadores generados por diferentes agentes de usuario pero este comportamiento no será requerido en la implementación de esta práctica. Las direcciones de contacto son direcciones sip que llevan a la derecha de la @ la IP o nombre de la máquina donde se esperan recibir los mensajes SIP. Se recomienda usar este formato para la práctica. De cara a facilitar la implementación para máquinas que no tienen asociadas registros de recurso tipo A en el DNS se pueden

usar directamente direcciones IP. El From y el To llevarán obligatoriamente la dirección SIP de llamante y llamado asociada al dominio, sin incluir la IP de la máquina. Podrán opcionalmente llevar también el nombre de usuario a la izquierda de la dirección SIP. Todos los mensajes irán sobre UDP. El campo expires, aunque recogido en este ejemplo no será de obligatorio procesamiento. En caso que se procese, el registro de un usuario caducará según se indica en la cabecera expires (en el ejemplo la caducidad es de 2 horas).

El agente de usuario registrará automáticamente al usuario pasado por línea de comandos en el proxy al arrancar. Si obtiene como respuesta un 200 OK se mantendrá a la espera de que el usuario inicie una llamada (por línea de comandos) o a recibir una llamada de la red. Si recibe un 404 la aplicación de agente de usuario finalizará. Si el proxy no contesta al mensaje de REGISTER, el agente de usuario hará ***reintentos cada 2 segundos*** *hasta obtener respuesta.*

Establecimiento de llamada

Se iniciará con un mensaje de INVITE desde el terminal de usuario dirigido al proxy cuando el usuario introduzca por teclado la línea:

INVITE usuarioSIPDestino

Para facilitar la interfaz de usuario, usuarioSIPDestino se especificará con nombre de usuario, sin dominio. Para generar el mensaje INVITE se añadirá automáticamente el dominio sacado del usuario llamante (que se pasó por línea de comandos). El mensaje SIP generado para el inicio de la llamada tendrá un formato como el siguiente:

```
INVITE sip:bob@atlanta.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com
Max-Forwards: 70
To: Bob <sip:bob@atlanta.com>
From: Alice <sip:alice@atlanta.com>
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142

(Alice's SDP not shown)
```

Aunque la carga útil SDP no se muestra, es importante notar que existe una línea en blanco entre la última de las cabeceras SIP y el contenido en SDP.

El mensaje INVITE, en la línea de petición, contiende la URI sip del destinatario usando el formato de usuario@dominio. Aunque el proxy resolverá esta URI a una dirección concreta donde ubicar la recepción de llamadas por parte de este usuario, para la implementación de esta práctica no será necesario cambiar la URI de la línea de petición.

El Via se puede rellenar con direcciones IPs, al igual que el Record-Route y el Route.

El Call-ID se muestra sin la @ y puede usarse de esta manera.

El Max-Forwards puede inicializarse a cualquier otro valor que se estime conveniente.

El CSeq puede empezar en 1 o en un valor aleatorio pero cada mensaje irá incrementando en una unidad el contador para ese mismo mensaje u otro mensaje anterior (es decir, si el último mensaje tiene CSeq 23 INVITE, el siguiente ACK será 24 ACK).

El proxy contesta con un mensaje de intentándolo:

```
SIP/2.0 100 Trying
Via: SIP/2.0/UDP pc33.atlanta.com
To: Bob <sip:bob@atlanta.com>
From: Alice <sip:alice@atlanta.com>
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Content-Length: 0
```

A continuación el proxy reenvía el mensaje de INVITE al destino (solo implementamos llamadas en el mismo dominio). El Proxy usará la información de los usuarios registrados para saber cómo progresar la llamada.

```
INVITE sip:bob@atlanta.com SIP/2.0
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com
Via: SIP/2.0/UDP pc33.atlanta.com
[Record-route opcional]
Max-Forwards: 69
To: Bob <sip:bob@atlanta.com>
From: Alice <sip:alice@atlanta.com>
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142
(Alice's SDP not shown)
```

El destino hará sonar el terminal y mandará un mensaje 180 Ringing hacia atrás usando la Via:

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com
Via: SIP/2.0/UDP pc33.atlanta.com
[Record-route opcional]
To: Bob <sip:bob@atlanta.com>
From: Alice <sip:alice@atlanta.com>
Call-ID: a84b4c76e66710
Contact: <sip:bob@192.0.2.4>
CSeq: 314159 INVITE
Content-Length: 0
```

A medida que los servidores intermedios reciben el mensaje irán quitándose de la Via hasta que el mensaje llegue al origen.

Nótese que la cabecera Record-Route que añade el proxy se incluye en los mensajes que se generan por el llamado como respuesta al INVITE y el proxy no quita dicha cabecera que debe llegar íntegra al llamado. De esta forma, los siguientes mensajes dentro de la llamada (empezando por el ACK que confirma el establecimiento de llamada) llevarán la cabecera Route con la dirección que contenía el Record-Route. En el caso de la cabecera Route, el proxy si se quitará de la misma cuando reciba un mensaje en el que esté incluido en dicha cabecera. Aunque de cara a notar que el comportamiento esperado es el definido en la RFC 3261 (loose-routing) frente a la versión anterior según al RFC 2543 (strict-routing), se debería añadir la etiqueta ;lr a la dirección del proxy, para la presente práctica no se hará para simplificar la implementación.

Cuando el usuario descuelgue se propagará igualmente hacia atrás un mensaje 200 OK. Si el usuario no descuelga se manda hacia atrás un código de 408 Request Timeout y si el usuario rechaza la llamada o ya se encuentra en una llamada se mandará un 486 Busy.

Describiendo el contenido con SDP

El establecimiento de la llamada (mensajes INVITE) debe contener información describiendo la sesión. Para ello se usará SDP (http://www.ietf.org/rfc/rfc2327.txt). Para facilitar la implementación usaremos los siguientes campos de SDP para describir la sesión:

```
c=IN IP4 224.5.6.7
m=audio 49230 RTP/AVP 96 97 98
a=rtpmap:96 L8/8000
a=rtpmap:97 L16/8000
a=rtpmap:98 L16/11025/2
```

donde c indica la dirección IP donde se desea recibir el audio, m indica el Puerto y las diferentes opciones soportadas y para cada opción se usa un campo a para describirlo.

El puerto de RTCP se entiende que es el siguiente puerto (RTP en puerto par y RTCP en puerto impar).

Para la implementación de la práctica y de cara a simplificar la configuración de los agentes de usuario, solo se podrán mandar dentro del único médio m tipo audio, los valores 96, 97 o 98, cada uno con un atributo rtpmap según lo capturado en el mensaje anterior y según las características del audio de la práctica de la primera parte de la asignatura.

Cerrando sesión SIP con BYE

Cualquiera de los dos intervinientes en la llamada puede mandar un mensaje de BYE que será confirmado con un mensaje de OK. Un ejemplo recogido de la RFC sería:

```
BYE sip:alice@pc33.atlanta.com SIP/2.0
Via: SIP/2.0/UDP 192.0.2.4
[Route opcional]
Max-Forwards: 70
From: Bob <sip:bob@atlanta.com>
To: Alice sip:alice@atlanta.com
Call-ID: a84b4c76e66710
CSeq: 231 BYE
Content-Length: 0
```

Que será confirmado con:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.0.2.4
[Route opcional]
From: Bob <sip:bob@atlanta.com>
To: Alice <sip:alice@atlanta.com>
Call-ID: a84b4c76e66710
CSeq: 231 BYE
Content-Length: 0
```

Si a la hora de establecer la llamada no hemos forzado una ruta para los mensajes de señalización mediante loose-routing, el mensaje de BYE se enviará extremo a extremo. En caso contrario pasará por el proxy. Nótese que para el caso de tener el loose routing activado, el proxy se añadirá a la Vía del BYE y se quitará de la cabera Route.

API de mensajes SIP proporcionado

Para simplificar el esfuerzo en la realización de las prácticas de esta segunda parte de la asignatura, se proporciona la implementación de un API de mensajes que permite parsear/analizar los mensajes en formato texto recibidos de la red y transformarlos en clases Java cuyos atributos contienen la información de las cabeceras de los mensajes de cara a poder procesarlos. A su vez, las clases proporcionadas contendrán métodos para simplificar el procesamiento y manipulación de mensajes (añadir Vias por ejemplo) así como el volver a generar una nueva cadena con el mensaje procesado/generado.

Este API es de obligatorio uso

Si el formato de las cabeceras de los mensajes no coincide con la especificación del presente enunciado, el API lanzará excepciones que nos permiten saber lo que ha ocurrido.

Se proporciona también documentación generada con Javadoc para conocer los diferentes métodos implementados así como los diferentes atributos de cada una de las clases. Se ha añadido detalles del comportamiento de cada uno de los métodos en 4 clases principales

- *InviteMessage*
- *ACKMessage*
- *SDPMessage*
- *SIPMessage*

Para el resto de clases no se han añadido detalles a la documentación generada pues el comportamiento es similar y se puede consultar en estas clases.

También se proporciona un programa de Prueba en Java que usa el API para componer y parsear/analizar varios mensajes a modo de ejemplo de cómo utilizar el API y para que se pueda ver cómo el API reacciona ante cabeceras no especificadas o con diferentes formatos permitidos para las mismas.

En concreto el API consta de las siguientes clases:

- *ACKMessage*
- *BusyHereMessage*
- *ByeMessage*
- *InviteMessage*
- *NotFoundMessage*
- *OKMessage*
- *RegisterMessage*
- *RequestTimeoutMessage*
- *RingingMessage*
- *SDPMessage*
- *ServiceUnavailableMessage*
- *SIPException*
- *SIPMessage*
- *TryingMessage*

Todos los mensajes heredan de la clase SIPMessage y generan SIPExceptions en caso que el parseo de las cabeceras no coincida con el formato esperado.