Proxy con capacidad de ejecutar SIP Servelts para dar servicios de llamada

El objetivo de esta segunda práctica es dotar al proxy SIP desarrollado en la práctica 1 de un entorno de ejecución de SIPServlets (contenedor de SIPServlets). De esta manera tendremos un entorno extensible programáticamente para la provisión de servicios de gestión de llamadas.

De cara a simplificar el entorno de ejecución, en el API que implementaremos para los SIPServlets solo atenderemos los mensajes de INVITE (es decir, nuestros SIPServlets solo responderán al flujo de control de llamada SIP con el método doInvite).

Cada usuario tendrá asociado un SIPServlet para procesar sus llamadas (entrantes y salientes). La carga en tiempo de ejecución de la clase apropiada que implemente el SIPServlet del usuario que nos interese para una determinada llamada se hará usando las facilidades que ofrece la clase Class en Java (en concreto sus métodos forName y newInstance).

El API

El API ofrecido para la programación de SIPServlets por nuestro Proxy SIP se basará en 4 interfaces:

```
public interface SIPServletInterface {

      public void doInvite(SipServletRequest request);

}

public interface SipServletRequest {

        public String getCallerURI();
        public String getCalleeURI();
        public SipServletResponse createResponse(int statuscode);
        public Proxing getProxy();
}

public interface SipServletResponse{

        public void send();
}


public interface Proxing {
        public void proxyTo(String uri);
}
```

La implementación de un SIPServlet será una clase pública Java que implemente al interfaz SIPServletInterface arriba definida.

Para la ejecución de un SIPServlet, nuestro contenedor (el proxy SIP extendido de la práctica 1) deberá utilizar clases de soporte que implementen las anteriores interfaces. De esta forma se propone implementar las clases SipServletRequestImpl, SipServletResponseImpl y ProxingImpl, cada una de ellas implementando una de las interfaces del API. Lógicamente, además de los métodos definidos en la interfaces se podrá dotar a dichas clases de los atributos, constructores y métodos adicionales que se estimen oportunos. Por ejemplo, cuando invocamos los métodos SipServletResponse.send() o Proxing.proxyTo() se debe notificar de la decisión tomada a la parte implementada en la práctica 1 que está gestionando los mensajes SIP en nuestro servidor proxy (desde donde habíamos instanciado el SIPSevlet y habíamos invocado el método doInvite). Con ello, nuestro servidor proxy SIP podrá actualizar el estado de la llamada. Además, los mensajes SIP a enviar se podrán enviar o bien desde la

implementación de las clases del API de Servlets SIP o bien desde la parte del servidor proxy SIP de la práctica 1. En uno u otro caso hará falta también que haya intercambio de objetos entre ambas partes.

Asociando un SIPServlet a un usuario concreto

Para asociar una implementación de un SIPServlet a una URI SIP se definirá un fichero de usuarios en formato XML, al que llamaremos Users.xml (que será nuestro descriptor de despliegue simplificado), que será leído por el Proxy al arrancar y que tendrá el siguiente formato:

```
<users>
      <user id="sip:mario@it.uc3m.es">
            <Servlet-class name="example.sip.MarioSIPServlet" />
      </user>
      …
</users>
```

Cada usuario tendrá pues asociada una clase que implemente la interfaz SIPServletInterface. Al recibir un INVITE para una nueva llamada, el proxy SIP extendido que implementamos en esta segunda práctica deberá buscar y comparar el contenido de las cabeceras TO y FROM del mensaje INVITE con los IDs de los usuarios en el Users.xml. Cuando encuentre al llamado y/o al llamante deberá crear una Class mediante el método Class.forName pasándole como nombre el atributo name del elemento Servlet-class para el usuario. Una vez obtenido el objeto Class invocaremos el método newInstance haciendo un casting a SIPServleInterface para su manejo. Una vez instanciado el ServletSIP, el servidor proxy que habíamos implementado en la práctica 1 ahora creará de igual manera un objeto de tipo SipServletRequestImpl. Cuando tanto el llamado como el llamante tienen asociado un SIP Servlet en el fichero de configuración Users.xml, y de cara a simplificar la implementación del servidor, solo se ejecutará el Servlet del llamado.

Detalles de Implementación

Se comentan en esta sección algunos detalles o pistas que se pueden seguir para la implementación de esta práctica 2. No son de obligatorio cumplimiento pues no forman parte de la especificación del API simplificado de Servlets SIP que hemos recogido anteriormente pero pueden dar ideas útiles.

Además de los métodos exportados por el SipServletRequestImpl (al igual que las otras clases implementadas, como SipServletResponseImpl o ProxingImpl) a través de la implementación de la interfaz SipServletRequest, al SipServletRequestImpl es bueno añadir algunos adicionales para el intercambio de datos entre la parte implementada en la práctica 1 y los Servlets SIP instanciados en esta práctica 2 así como para el intercambio de datos entre las clases de este nuevo API de Servlets SIP. Por ejemplo se podrá tener un método de inicialización para inicializar los atributos en los que guardar las URIs del llamante y llamado.

La invocación de los métodos en la interfaz SipServletRequest:

```
public SipServletResponse createResponse(int statuscode);
public Proxing getProxy();
```

podrán a su vez instanciar objetos de las clases SipServletResponseImpl y ProxingImpl. El objeto de la clase SipServletResponseImpl podrá tener también un método de inicialización donde recoger el parámetro con el estado de la respuesta a generar. Ambas clases, en su método de inicialización podrán recibir una referencia o mecanismo de "callback" que permite volver a comunicarse con el Proxy SIP. La implementación de esta referencia de "callback" no es relevante al programador de Servlets SIP simplificados (por eso es trasparente al API) y se deja a criterio del alumno su definición y codificación.

Para el procesamiento del fichero Users.xml no es necesario aún usar el API de SAX (lo dejaremos para la práctica 3). Se puede usar un procesamiento básico de cadenas de texto buscando las apariciones de las sub-cadenas "id" y "name" con métodos como indexOf y subString (por ejemplo). En cualquier caso, si así lo deseáis, podéis ya a empezar el api SAX e ir cogiendo familiaridad con el mismo.

Probando la práctica

Para poder comprobar el correcto funcionamiento de la práctica, se requiere que el alumno implemente un SIP Servlet según el API anterior para un usuario y que deje solo recibir llamadas de 9 de la mañana a 18 horas y solo para llamadas que provienen del usuario "boss". Se creará un fichero Users.xml que asocie dicho SIP Servlet a su usuario y se verificará su correcto funcionamiento.