SiFi: Sistema de Intercambio de Ficheros

Álvaro Navarro Clemente, Jaime Pérez Crespo
anavarro@javasucks.es
jperez@javasucks.es
Universidad Rey Juan Carlos, Móstoles, España

Febrero del 2006

1. Introducción

La práctica está estructurada en diferentes directorios:

- client
- objectclient
- monitor
- pasarela_entrada
- pasarela_salida
- agente_externo

En cada uno de ellos está almacenada la implementación así como algún recurso extra como iconos o imágenes. Además se ha incluido un pequeño *script* escrito en *bash* para la ejecución de cada elemento. Dicho ejecutable se encarga de recoger los argumentos introducidos desde línea de comandos y llamar a la clase que ejecuta cada elemento con los parámetros adecuados (recordemos que necesitamos algún parámetro extra para CORBA y RMI).

Para compilar la práctica es necesario establecer una serie de variables de entorno (como por ejemeplo el CLASSPAH). Para ello nos serviremos de un script llamado *environment.sh* que se encarga del trabajo. Una vez configurado el entorno adecuadamente podemos proceder a la compilación del código fuente mediante la herramienta *make*. Así pues, para compilar la práctica bastaría con hacer:

```
anavarro@xeon:~/sifi$ . environment.sh
anavarro@xeon:~/sifi$ make all
```

El fichero Makefile contiene diferentes reglas, una para cada módulo, de forma que podríamos compilar módulos de forma independiente ahorrando así tiempo.

Es importante recordar que antes de ejecutar cualquier binario de la infraestructura, es indispensable ejecutar los Servicios de Nombrado y Eventos de CORBA, así como el *rmiregistry* de la parte RMI.

2. Diseño e Implementación

A continuación pasaremos a describir brevemente las características más relevantes en cuanto a diseño e implementación.

2.1. Cliente

El Cliente SiFi posee dos partes fundamentales:

- GUI, se trata de un robusto interfaz de usuario que permite la interacción de un usuario dentro de la red SiFi. El interfaz posee varios menús: configuración de la carpeta compartidos, pantalla de descargas y subidas y recordatorio de login.
- Comunicación RMI. Un interfaz de comunicación vía RMI con su objetocliente asociado. Recordemos que éste será el vínculo de unión entre el cliente y el resto de la red.

La implementación del cliente se basa en el patrón de diseño vista/controlador. Para cada una de las ventanas tenemos asociados dos ficheros: su vista y su controlador, así por ejemplo, la pantalla de configuración de descargas tendrá dos clases asociados: VentanadDownloads.java y ControladorDownloads.java.

El cliente necesita dos argumentos:

- host: donde está lanzado el *rmiregistry*.
- id: identificador único dentro de la red SiFi. Este será el identificador por el que otros usuarios le reconocerán.

2.2. Objeto Cliente

El Objeto Cliente SiFi se encarga de recibir las peticiones vía RMI del Cliente y distribuirlas vía CORBA por el resto de la red. Así pues tendrá dos interfaces:

- Interfaz RMI con métodos tales como search (búsqueda) y getFile (descarga)
- Interfaz CORBA. Conecta con el servicio de nombres y se une al canal de eventos. Además posee una interfaz que contiene métodos usados para recoger resultados de una búsqueda así como la descarga de un fichero.

El ObjetoCliente necesita dos argumentos:

- host: donde está lanzado el *rmiregistry*.
- id: identificador único del cliente al que se asociará vía RMI.

2.3. Monitor

El monitor se encarga de monitorizar todas las acciones que tienen lugar en nuestra red corporativa. Así pues, las búsquedas, los resultados y las peticiones de descarga quedarán registradas.

Además el monitor posee un servidor web interno con el fin de mostrar las acciones registradas. De este modo gana un grado más de distribución en la arquitectura.

El Monitor necesita dos argumentos:

- host: donde está lanzado el Servicio de Nombres CORBA.
- port: puerto donde escuchará peticiones el servidor web.

2.4. Agente Externo

El agente externo tan sólo se conecta a la red sifi por las pasarelas de entrada y salida. No comparte ningún fichero con el resto de la red limitándose su uso a búsquedas simples y descargas de un fichero concreto.

El Agente Externo posee una sencilla interfaz gráfica basada en *AWT*, el cual se encargará de llamar a los métodos RMI exportados por la pasarela de entrada.

El Agente Externo necesita un sólo argumento:

- host: donde está lanzado el *rmiregistry*.

2.5. Pasarelas de Entrada y Salida

Ambas pasarelas permiten la conexión de agentes externos con el mundo SiFi. De tal forma que:

- Pasarela Entrada: tan sólo permite tráfico hacia el mundo Sifi (nunca al contrario). Posee una interfaz RMI para conectar con el agente externo. Además se conecta al canal de eventos para mandar las búsquedas provenientes desde el exterior.
- Pasarela Salida: se encarga de descargar el fichero pedido por el agente externo (proxy). Posee una implementación del API para envío de correos en Java (*javamail*). De esta forma al agente externo le llegarán por correo las respuestas de los diferentes agentes SiFi.

La pasarelas necesitan los siguientes argumentos:

- host: donde está lanzado el *rmiregistry* y el servicio de nombres y eventos (para las dos pasarelas).

- smtp: servidor con capacidad de relay mediante el cual enviaremos el correo (sólo para la pasarela de salida).

3. Futuras aplicaciones y conclusiones

Pese a que la infraestructura implementada es totalmente funcional, se podría realizar alguna mejora no recogida en el enunciado inicial de la práctica:

- Envío distribuido de ficheros. Si necesitamos un fichero y hay varios clientes con ese mismo *hash* de fichero, lo idóneo sería un envío por fragmentos (*chunks*) desde los diferentes clientes, ganando así velocidad (sobre todo en ficheros grandes). De esta forma conseguiríamos un verdadero sistema P2P distribuido.
- Si no poseemos un *display* local no podremos ejecutar el cliente gráfico (a menos que exportemos el display con la consiguiente pérdida de velocidad). Lo idóneo sería la posibilidad de elegir interfaz: *ncurses*, *html* o *java awt*. Tal y como tenemos desacoplada la vista del controlador no sería muy complicado de extender.
- Incrementar la seguridad del sistema: conexiones bajo *SSL*, autenticación web para el monitor, seguridad en la integridad de los ficheros (no posibilidad de intercambio de ficheros ejecutables), etc..

Por último reseñar que la práctica ha sido desarrollada únicamente con herramientas libres (a excepción de java, naturalmente): *vim*, *subversion* y *make*.

Referencias

[1] OpenORB Documentation
http://openorb.sf.net

[2] Curso Practico de CORBA en GNU/Linux
Alvaro del Castillo