Sistemas distribuidos (EI1021)
Práctica 4. Objetos distribuidos con Java RMI

En esta práctica utilizaremos el paradigma de objetos distribuidos para implementar una versión distribuida (cliente/servidor) de el servicio de compartición de coches. En concreto utilizaremos Java RMI para implementar un servicio que atenderá las peticiones de clientes.

Al igual que ocurre con la implementación distribuida basada en sockets, el aspecto y funcionalidad visible por los clientes del servicio será exactamente el mismo que en la versión local de la misma implementada en la Práctica 1. El cliente no tiene porque ser consciente de que está accediendo a un Sistema Distribuido y que sus peticiones en el servicio de compartición son atendidas por métodos de objetos remotos, que pueden estar ejecutándose en una máquina servidor distinta de su ordenador desde el que accede al servicio.

Primer ejemplo

En este ejercicio vamos a utilizar Eclipse para probar un ejemplo simple de aplicación distribuida creada con RMI. Para ello descargaremos una copia adaptada de uno de los ejemplos del libro `Distributed Computing` y lo integraremos paso a paso en un proyecto de IntelliJ. Para diferenciar las distintas componentes de la aplicación distribuida usaremos 3 paquetes dentro del proyecto: uno con el código del cliente, otro con el código del servidor y un tercero conteniendo el código común a ambos, en este caso la interfaz del servicio.

1. Descarga del Aula Virtual el código ejemplo `HelloWorldRMI.zip` y descomprímelo.
2. Abre el proyecto descomprimido con la opción `Open` del menú `File`.
3. Fíjate cómo se usa el `import` en el cliente para hacer referencia a la interfaz compartida con el servidor. ¿Por qué es necesario este import en el cliente? ¿Tiene que estar también accesible en el servidor?
4. Ejecuta el código del servidor, `HelloServer`. Hemos programado que el puerto para el registro sea el `1099`.
5. Ejecuta el código del cliente, `HelloClient`. Hemos programado que el host donde está el registro es `localhost` y el puerto que escucha ese registro es `1099`.
6. Si quieres puedes arrancar varios clientes sin parar el servidor y comprobar el efecto resultante.
7. ¿Qué ocurre si paras el servidor e intentas arrancar un cliente?

Servicio de compartición con RMI

Utilizando como modelo el ejemplo anterior, vamos a implementar el servicio de compartición con la misma funcionalidad que la desarrollada en la Práctica 1. Implementaremos las distintas clases a utilizar siguiendo la metodología presentada en los apartados 7.7 y 7.8 de [Liu'04].

Para entender el funcionamiento del sistema distribuido y el papel jugado por cada una de las clases a implementar nos basaremos en la siguiente figura.

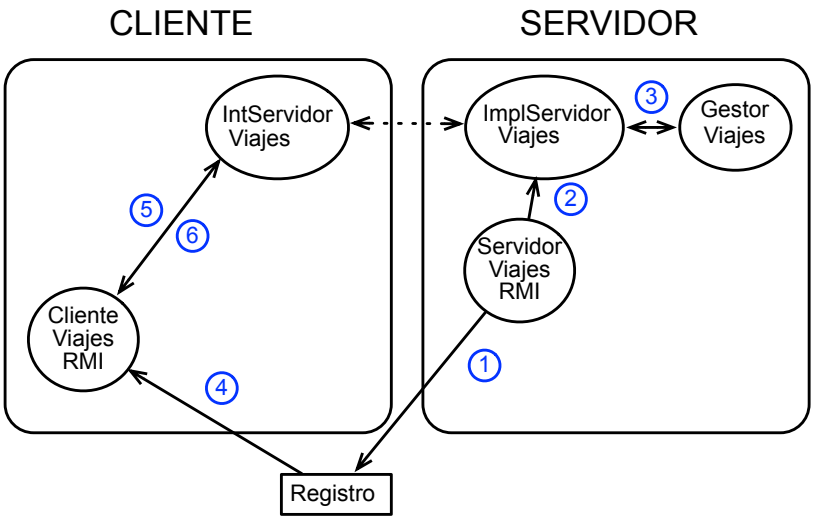

Figura 1: Funcionamiento de la versión RMI del servicio de compartición y objetos implicados en el cliente y el servidor.

1. En primer lugar, el servidor, `ServidorViajesRMI`, arranca el registro, si no está ya arrancado.
2. A continuación instancia un objeto remoto `ImplServidorViajes` y lo registra. Este objeto remoto hará de intermediario entre los clientes y el gestor de Viajes en el servidor.
3. El constructor del objeto anterior instancia un objeto `GestorViajes` que se encargará de gestionar las peticiones de todos los clientes sobre un único fichero compartido.
4. Al arrancar cada cliente, de la clase `ClienteViajesRMI`, este obtendrá del registro una referencia al objeto remoto `ImplServidorViajes`.
5. El cliente guardará esa referencia al objeto remoto en una referencia a su interfaz `IntServidorViajes`, ya que el cliente no dispone de su implementación.
6. El cliente ejecutará un código muy similar al de las prácticas anteriores. En esta ocasión en lugar de acceder a un objeto local `GestorViajes` utilizará la interfaz del objeto remoto `IntServidorViajes` para acceder a las operaciones del gestor, que se ejecutarán en el servidor.

Ejercicios

1. Crea la interfaz del servidor remoto `IntServidorViajes` con los seis métodos públicos ofrecidos a los clientes por el gestor de Viajes desde la práctica 1.
2. Crea la clase que implementa la interfaz remota anterior: `ImplServidorViajes`. Utiliza para ello los métodos de la clase `GestorViajes` de la práctica 1 sin modificarla.
3. Implementa la clase `ServidorViajesRMI` conteniendo el programa principal que instancia un objeto de la clase anterior y registra el servicio.
4. Copia el código de la clase `ClienteViajes` de la práctica anterior y llámalo `ClienteViajesRMI`. Modifica la clase para que tras localizar el servicio en el registro y obtener el resguardo (*stub*) del objeto remoto `ImplServidorViajes`, presente repetidamente un menú que permita acceder a las funciones ofrecidas por el servidor usando el objeto remoto.

Uso de *callbacks*

En este apartado vamos a imitar el ejemplo del apartado 8.1 de [Liu'04] para permitir que el servidor pueda notificar a los clientes que deseen que se les avise cuando se oferte un nuevo viaje con un origen determinado. Para realizar dichas notificaciones se utilizará el mecanismo de *callback* proporcionado por RMI.

Vamos a modificar una copia del proyecto creado en los ejercicios anteriores para añadir la nueva funcionalidad al servicio.

1. Haz una copia del proyecto creado en los ejercicios anteriores.

2. Incluye dos nuevos métodos en la interfaz `IntServidorViajes` e impleméntalos en la clase correspondiente, de modo similar a las figuras 8.6 y 8.7. Dichos métodos permitirán a los clientes registrarse para recibir notificaciones y borrarse del registro asociado a un origen dado.

   El objeto remoto `ImplServidorViajes` guardará en un diccionario los objetos remotos de los clientes que quieran recibir notificaciones. Cada elemento de dicho diccionario estará asociado a un origen que actuará como clave. El valor asociado a esa clave es un vector dinámico con los objetos remotos de los clientes que se hayan registrado para recibir dichas notificaciones. Para evitar asociar vectores distintos a un mismo origen escrito con distintas combinaciones de mayúsculas o minúsculas, los métodos asociados a las notificaciones deben usar versiones normalizadas de los orígenes escritos completamente en minúsculas (o mayúsculas).

3. Crea una nueva interfaz remota del cliente, `IntCallbackCliente`, e impleméntala de modo similar a como se muestra en las figuras 8.3 y 8.4 para que los clientes registrados puedan recibir notificaciones cuando se oferte un nuevo viaje con un origen dado.

4. Implementa un nuevo método en la clase `ImplServidorViajes` que dado un nuevo viaje ofertado, notifique la oferta a los clientes registrados para recibir notificaciones asociadas al origen del mismo.

   Puede ocurrir que alguno de los clientes que se había registrado para recibir notificaciones ya no esté disponible. En esa situación se producirá una excepción al intentar notificarle, en cuyo caso el servidor debe borrar el objeto callback del vector para evitar ese mismo problema en el futuro y continuar notificando al resto de clientes registrados.

5. Modifica el método `ofertaViaje` para que, cuando se oferte un nuevo viaje, use el método del apartado anterior para notificar a los clientes registrados para su origen.

6. Modifica el programa principal del cliente para incluir dos nuevas opciones en el menú que le permitan registrarse y borrarse para recibir notificaciones.