Práctica ORB 3. Invocación remota de objetos de una misma Interfaz mediante el uso de esqueletos

Objetivos específicos que persigue la realización del trabajo.

Transformación de un programa que realiza una llamada a una serie de objetos locales, en otro que realiza la llamada remota a estos objetos situados en un nodo remoto, usando un esqueleto de Interfaz.

Fecha	Grupo	Tiempo resolución

Apellidos y nombre	Calificación

Un servidor multiobjeto monoservicio, como el implementado en las prácticas anteriores, ofrece solo la posibilidad de invocar métodos sobre múltiples objetos pertenecientes a la misma interfaz de servicio. En esta práctica se construirán las bases de un servidor multiservicio, usando un esqueleto para el servicio de Agenda. Esta construcción requiere el rediseño de la referencia remota de un objeto y la inclusión del concepto de esqueleto.

Rediseño de la referencia remota de un objeto (ObjRef), para que ésta incluya el identificador de la interfaz a la que pertenece. Como se ha visto en la prácticas anteriores, el servidor monoservicio asocia a cada objeto de una interfaz un número natural cuando lo crea. Por ejemplo, suponga que un cliente crea los siguientes objetos de tipo Agenda:

```
Agenda agendaTel = new Agenda();
Agenda guiaClaves = new Agenda();
```

El servidor, al tratar el constructor de cada objeto, asignará a agendaTel el identificador de objeto oid = 0, mientras que la guiaClaves, le asignará el oid = 1.

Suponga ahora que el servidor soporta simultáneamente objetos de la interfaz Agenda y objetos de otra interfaz, por ejemplo de ServicioBancario. Entonces el primer objeto del ServicioBancario también tendrá oid = 0, el segundo oid = 1, etc. Pero, cuando el servidor reciba una petición con un oid =1, ¿cómo sabe si se refiere a un objeto de Agenda o del ServicioBancario? Podría distinguir por el número de parámetros y por el tipo del método a invocar pero, desafortunadamente, también es posible que dos interfaces muestren métodos con firmas iguales. Por tanto, es necesario redefinir la referencia remota de un objeto con los siguientes campos:

ObjRef = < host, port, iid, oid >. Donde: host indica el nombre de la máquina donde reside el servidor; port indica el puerto por donde escucha dicho servidor; oid el identificador del objeto dentro de dicho servidor, e iid el identificador de interfaz. La referencia de un objeto (objRef) permite que el cliente envíe en el mensaje toda la información para que el servidor pueda determinar a qué interfaz pertenece el objeto, y al objeto en sí.

Protocolo de petición de servicio. El protocolo de petición de servicio se compone de dos tipos de mensajes: petición y respuesta. Estos mensajes incluyen siempre la referencia remota del objeto. El formato será el siguiente:

Mensaje genérico de petición:

Dir IP del servidor	**Puerto del servidor**	**identificador de interfaz**	**identificador de objeto**	**número de método a invocar**	**[parámetro entrada]***
Host	Port	iid	oid	0..n	Param
Dir ip	int	int	int	Int	Tipo Param

En el caso de que la conexión sea TCP, los dos primeros campos de la referencia remota, host y port, solo se usan para que el cliente establezca la conexión con el servidor, al inicio de cada petición. En nuestro caso en cada invocación de método. Una vez establecida, dichos campos no se envían más en los mensajes. La notación [param]* indica que el parámetro puede aparecer 0,1,2...

Mensaje de respuesta:

Resultado	**[parámetro salida]***
ok	Param
Boolean	Tipo Param

Observe que se comienza indicando si ha habido éxito en la invocación, en cuyo caso le devuelve el resultado.

Uso de esqueletos de servicio. Un esqueleto de servicio es una clase que se encarga de:

a) Identificar el objeto sobre el que se realizará la operación
b) descodificar la operación y sus parámetros, analizando el mensaje de petición
c) invocar el método local sobre el objeto con los parámetros obtenidos y,
d) envíar la respuesta del método al cliente.

En los apartados siguientes se mostrará cómo invocar objetos remotos considerando el identificador de interfaz en el ejemplo de la Agenda. También se va a usar el esqueleto correspondiente a dicha interfaz en el lado del servidor. No se van a usar más servicios que el ofrecido por el Repositorio, para no complicar el código. Tras la finalización de este ejemplo, la inclusión de un nuevo servicio es trivial.

Diseño del Cliente

Llamada a objeto remoto. Ahora, en el cliente es necesario guardar en el proxy o stub del objeto remoto dos parámetros que indiquen el identificador de interfaz y el identificador de objeto, respectivamente. Suponga que el identificador de interfaz de Agenda es 0 (cosa que se debe saber de antemano por el cliente y el servidor), y que el identificador de objeto oid será el valor devuelto por el servidor tras la invocación al constructor. Fíjese en los comentarios del constructor de Agenda en el fichero de proxy mostrado a continuación:

```
/*************************************************/
// FICHERO: Proxy del Repositorio para el cliente
/*************************************************/
package cliente;
public class Agenda implements IRepositorio {

	static String host = "localhost";
	static int port = 9999;
	static int iid = 0; //Identificador de la interfaz. Interfaz Agenda = 0
	int oid = -1; //identificador de objeto dentro de la clase Agenda


	public  Agenda (){ //Constructor
		//A rellenar por el alumno
		//crea socket de cliente
		//manda petición de creación de objeto de agenda
		//lee del socket el oid  asignado por el servidor para este objeto
		this.oid = oid;
		//cierra el socket
	}
	public void asociar(String key, int v) {
		 //A rellenar por el alumno
	}

	public int obtener(String key) {
		 //A rellenar por el alumno
	}
}

/*************************************************************/
// FICHERO: CLIENTE
/*************************************************************/

public class Cliente {
public static void main(String[] args) {
	Agenda agendaTel = new Agenda();
	Agenda guiaClaves = new Agenda();
	agendaTel.asociar("Juan", 292929292);
```

```
        guiaClaves.asociar ("Moodle", 23323);
        guiaClaves.asociar ("Juan", 3333);
        System.out.println("Telefono Juan = " + agendaTel.obtener("Juan"));
        System.out.println("Clave Moodle = " + guiaClaves.obtener("Moodle"));
        System.out.println("Clave Juan = " + guiaClaves.obtener("Juan"));

        }
}
```

En cada método del stub del servicio se transfiere en primer lugar el identificador de interfaz iid, y luego el resto de campos del mensaje de acuerdo al método a invocar, como se ha hecho en la práctica anterior.

Diseño del servidor

Se muestran unas pinceladas de cómo construir el servidor usando el esqueleto del servicio de Agenda. Para ello, describiremos el ciclo de vida de un servicio, paso a paso:

Paso 1. Creación del esqueleto de servicio. Para que el servidor pueda mantener dinámicamente múltiples servicios, se propone **recoger en una clase de esqueleto de servicio el código que identifica e invoca los métodos y parámetros de una interfaz determinada**. Para que el servidor no tenga problemas para invocar cualquier esqueleto, se define un interfaz de esqueleto **ISkeleton**, con un método process().

```
        public interface ISkeleton {
        public void process( DataInputStream canalEntrada,
                                DataOutputStream canalSalida);
        }
```

Todo servicio tiene un esqueleto que implementa dicha interfaz, de tal forma que el servidor invocará el método process() del esqueleto correspondiente, sea cual sea el servicio.
Atención: ahora el esqueleto mantendrá la tabla objectHash de objetos creados bajo dicha interfaz. Dicha tabla guarda en tuplas < identificador oid, objeto>:

```
Hashtable<Integer, Object> objectHash = new Hashtable<Integer, Object>();
```

Paso 2. Alta del esqueleto del servicio. Ahora el servidor solo se encarga de guardar las tuplas del esqueleto del servicio con su identificador de interfaz iid. Observe que la creación de la tupla <iid, skeletonAgenda> debe hacerse antes de que el servidor atienda ninguna petición de Agenda. Para gestionar los esqueletos, el servidor usará la tabla y las funciones siguientes:

```
        //Tabla hash para guardar las tuplas <iid, skeleton>.
        Hashtable<Integer, ISkeleton> skeletonHash
```

```
//Asigna el iid a sk, y guarda <iid, sk> en la tabla skeletonHash
void addSkeleton(ISkeleton sk)

//Devuelve el esqueleto asociado a dicho identificador en la tabla  skeletonHash
ISkeleton getSkeleton(int iid)
```

Paso 3. Creación de un socket de servidor.
Paso 4. Espera solicitud de conexión del cliente.
Paso 5. Procesamiento de la petición de cliente. Cuando llega una solicitud de conexión, el servidor obtendrá el socket de cliente, leerá del canal el identificador de interfaz iid. Después obtendrá de sus tablas internas el esqueleto asociado a la interfaz iid.. El servidor invocará el método process de dicho esqueleto, pasando los descriptores del canal del socket del cliente.
Paso 6. Procesamiento de la petición por parte del esqueleto del servicio y retorno al cuerpo principal del servidor. El esqueleto: 1) obtiene del canal de entrada el identificador de objeto, el método a invocar, y los parámetros, 2) invoca el método local del objeto de servicio adecuado y 3) recoge el resultado y genera el mensaje de respuesta al cliente.
No olvide que cuando la operación es la creación de un objeto, el esqueleto tendrá que añadir una nueva tupla a su tabla de objetos, como ocurría en las prácticas anteriores.
Paso 7. Cierre del socket del cliente.
Paso 8. Vuelta a esperar una nueva conexión.

A continuación, se describe posible diseño del código del servidor. A partir de ahora, el servidor lo llamaremos ORB, puesto que ya puede comportarse como un gestor de objetos u Object Request Broker.

```
/*************************************************************/
// FICHEROS EN EL SERVIDOR
/*************************************************************/
/*************************************************************/
// FICHERO: Repositorio para el servidor
/*************************************************************/

package servidor;

public class Agenda implements IRepositorio {
	// Se implementa la interfaz IRepositorio con una tabla hash
	private Hashtable<String, Integer> ht = new Hashtable <String, Integer>();

	public void asociar(String s, int v) {
		ht.put(s,new Integer(v));
	}

	public int obtener(String s) {
		return ((Integer) ht.get(s)).intValue();
	}
}
```

```
/*************************************************************/
// PROGRAMA Servidor de objetos (Object Request Broker)
// FICHERO: ORB Server
/*************************************************************/
package servidor;

import ..;

public class ORB {// servidor ORB
      ServerSocket ss;
      Socket sc;
      DataInputStream canalEntrada;
      DataOutputStream canalSalida;
      Hashtable<Integer, ISkeleton> skeletonHash = new Hashtable<Integer,
      ISkeleton>();

      int iid = 0;
      int serverPort;

      ORB (int serverPort) {
            this.serverPort = serverPort;
      }

      public void addSkeleton (ISkeleton sk) {
            skeletonHash.put(sk.getIid(), sk);
      }

      public ISkeleton getSkeleton (int iid) {
            return skeletonHash.get(iid);
      }

      public void start() {
      // A rellenar por el alumno
      // Pone en funcionamiento el servidor.
      // Atiende peticiones y las redirige al esqueleto adecuado
      }

      public static void main(String[] args) {
            ORB myORB = new ORB(9999);
            //Se crea el esqueto para tratar peticiones de Agenda
            ISkeleton skAgenda = new SkeletonAgenda();
            //Se añade al servidor ORB
            myORB.addSkeleton(skAgenda);
            //Se arranca el servidor ORB
            myORB.start();
      }
}
```

```
/*************************************************************/
// PROGRAMA Interfaz de Esqueleto de servicio
// FICHERO: ISkeleton
/*************************************************************/
package servidor;
import ...

public interface ISkeleton {
      public void process( DataInputStream canalEntrada,
                           DataOutputStream canalSalida);
}



/*************************************************************/
// PROGRAMA Esqueleto del servicio de Repositorio
// FICHERO: SkeletonAgenda
/*************************************************************/
package servidor;
import …;

public class SkeletonAgenda implements ISkeleton {

      private final int iid = 0;
      //No puede haber dos esqueletos con iids iguales
      Hashtable<Integer, Object> objectHash = new Hashtable<Integer, Object>();

      public void addObject(Object obj) {
            objectHash.put(objid, obj);
            objid++;
      }

      public Object getObject(int objid) {
            return objectHash.get(objid);
      }

      public int getIid() {
            return iid;
      }
      public void process( DataInputStream canalEntrada,
                                 DataOutputStream canalSalida) {
            try {
                  // leer el informacion de la operacion a invocar
                  int numMethod = canalEntrada.readInt();
                  Agenda objetoRep = (Agenda) objeto;
                  switch (numMethod) {
                  ....
            } catch (IOException e) {
                  e.printStackTrace();
            }
      }
}
```

Se pide:

1. Complete el sevidor ORB para que cuando acepte una conexión la redirija al esqueleto de Agenda a través de la invocación del método process del esqueleto. Obtenga dicho esqueleto usando el método getSkeleton.
2. Complete el fichero `SkeletonAgenda` para que atienda una petición de un cliente de para crear un objeto de agenda, o asociar u obtener valores sobre un objeto agenda ya existente.
3. Modifique el proxy del cliente para incluya en los mensajes el identificador de interfaz.