Práctica ORB 4. Invocación remota a n objetos de dos Interfaces distintas. Uso de esqueletos

Objetivos específicos que persigue la realización del trabajo:

- Transformación de un programa que realiza una llamada a objetos locales de diferentes clases, a otro programa que realiza la llamada de forma transparente a los mismos objetos que residen ahora en un servidor remoto.
- Construcción de un servidor que soporte un número no limitado de instancias de diferentes clases.

Fecha	Grupo		Tiempo de resolución

Apellidos y nombre

Calificación

1. Introducción

El servidor de Agenda implementado en las prácticas anteriores es un servidor multiobjeto monoservicio, puesto que solo permite la invocación de métodos sobre múltiples objetos, pero pertenecientes a la misma interfaz de servicio.

En esta práctica se pretende construir un servidor multiservicio, de tal forma que ofrezca el uso remoto de múltiples objetos pertenecientes a diferentes servicios: Agenda y Tiempo.

Para ello, analizaremos qué requisitos debemos tener en cuenta y cómo debemos modificar el servidor monoservicio para que sea multiservicio.

2. Referencia Remota de un Objeto

La referencia remota de un objeto permite identificar de manera única un objeto frente al resto del objetos que mantiene el ORB o gestor de objetos, independientemente de a qué clase pertenezca. Esta información tiene que formar parte de los mensajes de petición por parte del cliente. Veamos razonadamente qué campos debe contener.

Si el servidor mantiene objetos que implementan la interfaz IRepositorio, entonces asignará al objeto agendaTelefonica un identificador de objeto objId = 0, al segundo objeto, guiaClaves, objId =1, etc. Si el servidor soporta objetos de otra interfaz, por ejemplo de un servicio de tiempo ITime, entonces el primer objeto creado para esa interfaz tendrá objId = 1, el segundo con objId = 2, etc.
Entonces, cuando el servidor recibe una petición con un objId =1, ¿cómo sabe si se refiere a un objeto de la interfaz IRepositorio o del ITime?
El servidor 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 (firma = cabecera del método, donde se definen los parámetros), pero que tengan funcionalidades distintas.
Por tanto, la referencia a un objeto debe incluir el identificador de la interfaz a la que pertenece. El identificador de interfaz (iid) puede ser un número natural en orden creciente.

Resumiendo, la información necesaria para localizar dicho objeto en Internet es:

ObjRef = < host, port, iid, objId >

Donde: **`host`** indica el nombre de la máquina donde reside el objeto; **`port`**, el puerto por donde escucha el servidor que atiende peticiones sobre ese objeto; **`objId`** el identificador del objeto dentro de dicho servidor, e **`iid`** el identificador de interfaz, para que el servidor pueda determinar a qué interfaz pertenece el objeto.

3. Servidor multiservicio, la interfaz ISkeleton y los esqueletos de servicio

Es necesario modificar el servidor para que permita la atención a diferentes objetos de distintas clases. En nuestro caso, objetos de la clase Agenda y objetos de la clase Time.

La clase Time usa la siguiente interfaz de tiempo ITime, tanto para el cliente como para el servidor.

```
public interface ITime {
      public int getHour();
      public int getMinute();
      public int getSenconds();
}
```

Para mantener múltiples servicios, sería poco flexible que el servidor incluya en su código el tratamiento de cada uno de los métodos de todos los interfaces de servicio que mantiene. Para que el servidor pueda mantener dinámicamente múltiples servicios, se propone recoger el código de la identificación e invocación de los métodos y parámetros de una interfaz determinada **en una clase de esqueleto de servicio**.

Un esqueleto de servicio o *dispatcher* es una clase en el lado del servidor que se encarga de:
a) la decodificación de los parámetros de una petición sobre un objeto: analizando el mensaje de petición, para obtener qué método debe invocarse y con qué parámetros,
b) la invocación del método con los parámetros obtenidos y,
c) el envío de la respuesta del método al cliente.

Para que el servidor se adapte para invocar cualquier esqueleto, se define un interfaz de esqueleto `ISkeleton`, con un método `process()`. 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.

```
package server;

import java.net.Socket;

public interface ISkeleton {
  //returns the interface id
   public  int getIid();
   //process the request
  public void process(Socket sc);}
```

6.2. Comportamiento del servidor principal

El servidor principal redirige la petición del cliente sobre un objeto, al dispatcher de la interfaz a la que pertenece el objeto.
Obviamente, para el servidor pueda distribuir las peticiones al dispatcher adecuado, es necesario que el servidor mantenga un objeto dispatcher por cada interfaz de servicio que es capaz de servir. Cada dispatcher va asociado a su identificador de interfaz. Esta asociación se puede hacer cuando se arranca el servidor.
Una forma de hacerlo, es que el servidor guarde tuplas <iid, skeleton> en una tabla hash de esqueletos, skTable.

Cuando llega una solicitud de conexión al servidor, éste obtendrá el socket de cliente, leerá del canal el identificador de interfaz iid. Después obtendrá de la tabla skTable el esqueleto asociado a la interfaz iid. El servidor invocará el método process de dicho esqueleto pasando el descriptor del canal del socket del cliente.

7. Se pide:

Cliente

Sea el siguiente cliente, que crea dos objetos, de las clases Time y Agenda. Se pretende que el cliente muestre la hora del servidor junto con un número de teléfono.

```
public class ClienteAgendaYTiempo {

  public static void main(String[] args) {
        Time horaActual = new Time();
    System.out.println("Hora Actual Servidor " + horaActual.getHour() + ":"+
    horaActual.getMinute()+":"+ horaActual.getSenconds());

        Agenda agendaTelefonica = new Agenda();
        agendaTelefonica.asociar("Juan", 66756677);
        System.out.println("Telefono Juan = " + agendaTelefonica.obtener("Juan"));
        }

}
```

Construya un proxy `Time` en el cliente para que sea posible pedir la hora al servidor. Suponga que el identificador de interfaz para el servicio de hora es iid = 2. Suponga que para el servicio de agenda es 1. (iid = 1).
Entréguelo.

Servidor

7.3) Modifique el servidor principal para que registre las interfaces ITime e IRepositorio, y sea capaz de asociar un esqueleto o dispatcher para cada petición que reciba del cliente. Entregue dicho servidor.

7.4) Escriba un esqueleto de tiempo TimeSkeleton, que trate las operaciones del servicio de tiempo. Entreguelo.

A continuación se muestra el servicio de la hora de clase Time:

```
import java.util.Calendar;
import java.util.GregorianCalendar;


public class Time implements ITime {

  public int getHour() {
        Calendar date = new GregorianCalendar();
        int hora = date.get(Calendar.HOUR_OF_DAY);
        return hora;
  }

  public int getMinute() {
        Calendar date = new GregorianCalendar();
        int minute = date.get(Calendar.MINUTE);
        return minute;
  }
  public int getSenconds() {
        Calendar date = new GregorianCalendar();
         int second = date.get(Calendar.SECOND);
        return second;
  }

}
```