Práctica PCD 07/08 - Subastas

Daniel García Moreno <dani@danigm.net>

26 de junio de 2008

Índice

1. Cómo compilar y ejecutar

1.1. Compilar

Para compilar hay que ejecutar: `javac src/subastas/*.java -d bin/` desde el directorio PCD. Los .class se guardarán en el directorio bin.

1.2. Ejecutar

El cliente y servidor se comunican a través de un fichero .ior, así que el servidor tiene que tener permiso de escritura en el directorio desde el cual se ejecute y el cliente tiene que ejecutarse desde el mismo directorio despues de haber ejecutado el servidor.

- **Servidor:** `java -classpath PCD/bin subastas.Servidor`
- **Cliente:** `java -classpath PCD/bin subastas.Cliente`

Para conocer los comandos disponibles en el cliente basta con escribir el comando 'help' que mostrará una lista con todos los posibles comandos.

Hay que tener en cuenta que para poder hacer alguna operación con alguna de las subastas creadas hay que hacer una busqueda para que se inicialice el vector de subastas, y el id será el indice en esta busqueda.

2. Explicación del diseño

2.1. Interfaces IDL

Las tres interfaces que hay que implementar se han implementado dentro del modulo subastas.

2.1.1. clienteSubastas

La interfaz clienteSubastas está implementada según la especificación del trabajo y además contiene una operación nuevoValor que sirve para resolver el primer problema especificado en el documento.

2.1.2. coordinadorSubasta

Para la interfaz gestorSubasta se ha declarado un tipo de dato sequence para la lista de clientes.

En esta interfaz se definen los atributos y las operaciones especificadas en el documento. Además se define la operación cerrarSubasta que será llamado para cerrar una subasta y así evitar el segundo problema.

2.1.3. gestorSubastas

Para la interfaz gestorSubasta se ha definido un tipo sequence para la lista de subastas que hay que devolver en la operación localizarSubasta.

Por lo demás se ha implementado según lo especificado en el documento.

2.2. Sirvientes

Se han implementado los sirvientes:

2.2.1. clienteSubastasImpl

Este sirviente implementa la interfaz clienteSubastas por herencia. Se han implementado las operaciones según la especificación. Cuando se actualiza el valor de una suabasta el cliente será informado por el coordinador subasta e imprimirá el nuevo valor.

2.2.2. coordinadorSubastaImpl

Al igual que el sirviente anterior este está implementado por herencia. Se han definido las operaciones según lo especificado en el documento.

Se han definido los métodos `abrirSubasta`, `ganador`, `pujar`, `ultimaPuja` y `cerrarSubasta` como synchronized puesto que serán llamados desde diferentes clientes. Para la sincronización entre clientes se utilizan los métodos proporcionados por java y vistos en clase, wait y notifyAll. Cuando se cambia el estado se ejecuta notifyAll.

Las esperas de los clientes se realizan mediante wait y el atributo estado.

2.2.3. gestorSubastasImpl

Como los dos sirvientes anteriores este también se ha implementado por herencia.

Este sirviente es un objeto fábrica. Crea coordinadores de subastas.

Para evitar el segundo problema especificado en el documento se ha utilizado la clase SubastaKiller que implementa la interfaz Runnable para ser lanzado en un hilo. Este hilo duerme durante el tiempo especificado y después cierra la subasta especificada llamando al método cerrarSubasta del coordinadorSubasta.

El resto de operaciones está implementado según la especificación del documento.

2.3. SubastaKiller

Para solventar el problema de que una subasta no acabe nunca por culpa de un cliente se ha implementado esta clase que implementa la interfaz Runnable. Este hilo duerme durante un tiempo especificado y después cierra la subasta.

2.4. Servidor

El servidor crea un gestorSubastasImpl y crea un fichero `server.ior` donde escribe el ior correspondiente a este objeto. Y se lanza el servidor.

2.5. Cliente unificado

Esta es la parte con más código del trabajo aunque no por ello la más compleja.
Se han definido los suiguientes comandos:

```
/**
 * COMANDOS DISPONIBLES
 *
 * GESTOR SUBASTAS
 * ---------------
 * salir -> sale del bucle
 * crear valor tiempo desc -> crea una subasta
 * buscar desc -> busca una subasta y muestra todas las coincidencias
 * borrar id -> borra una subasta si esta cerrada
 *
 * COORDINADOR SUBASTA
 * -------------------
 * abrir id -> abre la subasta por id
 * estado id -> devuelve el estado de una subasta
 * valor id -> devuelve el valor actual de una puja
 * ganador_prov id -> dice el identificador del cliente que va ganando la puja
 * ganador id -> muestra el ganador, si la puja no esta cerrada aun, bloquea a
 * clientes id -> muestra todos los identificadores de los clientes de la suba
 * inscribir id cliente_id -> inscribe un cliente en una subasta
 * pujar id cliente_id valor
 * ultima id cliente_id
 *
 * CLIENTE SUBASTAS
 * ----------------
 * crearcliente identificacion
 * cldisp -> muestra los clientes disponibles
 */
```

Cuando se ejecuta un cliente muestra una línea de comandos que permite llamar a cualquiera de estos comandos. No se han capturado los errores por lo que si se llama a alguno con una sintaxis inadecuada el programa dará errores.

Este cliente unificado permite tanto crear y gestionar subastas como crear clientes y trabajar con más de un cliente desde un mismo cliente unificado.

3. Código fuente

3.1. Defínción de interfaces en IDL

```
1
2  module subastas{
3
4
5    interface clienteSubastas{
6      readonly attribute string identificacion;
7      void finSubasta();
8      void nuevoValor(in float valor);
9    };
10
11   typedef sequence<clienteSubastas> lista_clientes;
12   interface coordinadorSubasta{
13     /**
14      * coordinadorSubasta: interfaz de objetos que gestionan el
15      * desarrollo de la subasta de cada bien subastado por la
           empresa.
16
17      * La subasta pasara por tres estados:
18      *  Inscripcion
19      *  Abierta
20      *  Cerrada
21      *
22      */
23
24     readonly attribute clienteSubastas ganador_provisional;
25     readonly attribute lista_clientes clientes;
26
27     readonly attribute float valor;
28     readonly attribute string descripcion;
29     // Inscripcion -> Abierta -> Cerrada
30     readonly attribute string estado;
31
32     void inscribirCliente(in clienteSubastas cl);
33     void abrirSubasta();
34     boolean pujar(in float cantidad, in clienteSubastas cl);
35     void ultimaPuja(in clienteSubastas cl);
36     clienteSubastas ganador();
37     void cerrarSubasta();
38   };
39
40   typedef sequence<coordinadorSubasta> lista_subastas;
41   interface gestorSubastas{
42     coordinadorSubasta crearSubasta(in float valor, in string
         desc, in long tiempo);
```

```
43      void destruirSubasta(in coordinadorSubasta cs);
44      lista_subastas localizarSubasta(in string palabra);
45    };
46
47  };
```

3.2. clienteSubastasImpl.java

```
1
2   package subastas;
3
4   /**
5    * @author danigm
6    *
7    */
8   public class clienteSubastasImpl extends clienteSubastasPOA {
9     private String identificacion;
10    private boolean fin=false;
11    private float valor_subasta=0;
12    public clienteSubastasImpl(String id) {
13    super();
14    identificacion = id;
15    }
16
17    public void finSubasta() {
18    fin = true;
19    }
20
21    public String identificacion() {
22    return identificacion;
23    }
24
25    public void nuevoValor(float valor) {
26    valor_subasta = valor;
27    System.out.println("NUEVO VALOR " + valor);
28    }
29
30  }
```

3.3. coordinadorSubastaImpl.java

```
1
2   package subastas;
3   import java.util.Vector;
```

```
4
5   /**
6    * @author danigm
7    *
8    */
9   public class coordinadorSubastaImpl extends
        coordinadorSubastaPOA {
10    private clienteSubastas ganador_provisional;
11    private Vector<clienteSubastas> clientes;
12    private float valor;
13    private String descripcion;
14    // Inscripcion -> Abierta -> Cerrada
15    private String estado;
16
17    public coordinadorSubastaImpl(float v_inicial, String desc) {
18    super();
19      estado = "Inscripcion";
20      ganador_provisional=null;
21      clientes = new Vector<clienteSubastas>();
22      valor = v_inicial;
23      descripcion = desc;
24    }
25
26    synchronized public void abrirSubasta() {
27      if(estado.equals("Inscripcion")){
28        estado = "Abierta";
29        notifyAll();
30      }
31    }
32
33    public clienteSubastas[] clientes() {
34    clienteSubastas[] cs = new clienteSubastas[clientes.size()];
35    for (int i=0; i<clientes.size(); i++)
36      cs[i] = clientes.get(i);
37    return cs;
38    }
39
40    public String descripcion() {
41    return descripcion;
42    }
43
44    public String estado() {
45    return estado;
46    }
47
48    public synchronized clienteSubastas ganador() {
49      //Si la puja esta cerrada devuelve ganador
50      //sino se bloquea hasta que se cierre
51    try {
52      while(!estado.equals("Cerrada"))
```

```
53        wait();
54      } catch (InterruptedException e) {
55        // TODO Auto-generated catch block
56        e.printStackTrace();
57      }
58        return ganador_provisional;
59      }
60
61      public clienteSubastas ganador_provisional() {
62      return ganador_provisional;
63      }
64
65      public void inscribirCliente(clienteSubastas cl) {
66        //estaria bien lanzar una excepcion
67        if(estado.equals("Inscripcion"))
68          clientes.add(cl);
69
70      }
71
72      public synchronized boolean pujar(float cantidad,
            clienteSubastas cl) {
73      String id = cl.identificacion();
74      try {
75        while(estado.equals("Inscripcion"))
76        wait();
77      } catch (InterruptedException e) {
78        // TODO Auto-generated catch block
79        e.printStackTrace();
80      }
81        if(estado.equals("Cerrada"))
82          return false;
83        else{
84          if (clientes.contains(cl)){
85            if (cantidad > valor){
86              valor = cantidad;
87              ganador_provisional = cl;
88              for (int i=0; i<clientes.size(); i++)
89                clientes.get(i).nuevoValor(valor);
90              return true;
91            }
92            else
93              return false;
94          }
95          else{
96            //quizas mejor una excepcion
97            return false;
98          }
99        }
100
101     }
```

```
102
103   synchronized public void ultimaPuja(clienteSubastas cl) {
104     if (clientes.contains(cl)){
105       clientes.remove(cl);
106     }
107     if (clientes.size() == 0) {
108       estado = "Cerrada";
109       notifyAll();
110     }
111   }
112
113   public float valor() {
114   return valor;
115   }
116
117   synchronized public void cerrarSubasta() {
118   estado = "Cerrada";
119   notifyAll();
120   }
121
122 }
```

3.4. gestorSubastasImpl.java

```
1
2  package subastas;
3
4  import java.util.Vector;
5
6  import org.omg.CORBA.*;
7  import org.omg.CORBA.ORBPackage.InvalidName;
8  import org.omg.PortableServer.POA;
9  import org.omg.PortableServer.POAHelper;
10 import org.omg.PortableServer.POAManagerPackage.AdapterInactive
       ;
11 import org.omg.PortableServer.POAPackage.ServantNotActive;
12 import org.omg.PortableServer.POAPackage.WrongPolicy;
13
14
15
16 /**
17  * @author danigm
18  *
19  */
20 public class gestorSubastasImpl extends gestorSubastasPOA {
21
22   private Vector<coordinadorSubasta> subastas;
```

```
23
24      private POA poa;
25      public gestorSubastasImpl(POA arg_poa) {
26        super();
27        subastas = new Vector<coordinadorSubasta>();
28        poa = arg_poa;
29      }
30
31      public coordinadorSubasta crearSubasta(float valor, String
            desc, int tiempo) {
32      coordinadorSubasta c = null;
33        coordinadorSubastaImpl coord = new coordinadorSubastaImpl(
              valor, desc);
34        org.omg.CORBA.Object obj;
35      try {
36        obj = poa.servant_to_reference(coord);
37        c = coordinadorSubastaHelper.narrow(obj);
38        subastas.add(c);
39      } catch (ServantNotActive e) {
40        // TODO Auto-generated catch block
41        e.printStackTrace();
42      } catch (WrongPolicy e) {
43        // TODO Auto-generated catch block
44        e.printStackTrace();
45      }
46      SubastaKiller sk = new SubastaKiller(c, tiempo);
47
48        return c;
49      }
50
51      public void destruirSubasta(coordinadorSubasta cs) {
52        if (cs.estado().equals("Cerrada")){
53          subastas.remove(cs);
54        }
55      }
56
57      public coordinadorSubasta[] localizarSubasta(String palabra)
            {
58        coordinadorSubasta[] encontradas = new coordinadorSubasta[
              subastas.size()];
59        int j=0;
60        for(int i=0; i<subastas.size(); i++){
61          if (subastas.get(i).descripcion().contains(palabra)){
62            encontradas[j] = subastas.get(i);
63            j++;
64          }
65        }
66        coordinadorSubasta[] encontradas2 = new coordinadorSubasta[
              j];
67        for(int i=0; i<j; i++){
```

```
68          encontradas2[i] = encontradas[i];
69        }
70
71      return encontradas2;
72      }
73
74  }
```

3.5. SubastaKiller.java

```
1
2   package subastas;
3
4   public class SubastaKiller implements Runnable{
5
6           private Thread hilo;
7           private long time;
8           private coordinadorSubasta cs;
9
10          public SubastaKiller(coordinadorSubasta cs, int time){
11                  hilo = new Thread(this);
12                  this.time = time*1000;
13                  this.cs = cs;
14                  hilo.start();
15          }
16
17          public void run() {
18                  try {
19                          hilo.sleep(time);
20                  } catch (InterruptedException e) {
21                          e.printStackTrace();
22                  }
23                  cs.cerrarSubasta();
24
25          }
26
27
28  }
```

3.6. Servidor.java

```
1
2   package subastas;
3
```

```
4  import java.io.File;
5  import java.io.FileNotFoundException;
6  import java.io.FileOutputStream;
7  import java.io.PrintWriter;
8
9  import org.omg.CORBA.*;
10 import org.omg.CORBA.ORBPackage.InvalidName;
11 import org.omg.PortableServer.*;
12 import org.omg.PortableServer.POAManagerPackage.AdapterInactive
       ;
13 import org.omg.PortableServer.POAPackage.ServantNotActive;
14 import org.omg.PortableServer.POAPackage.WrongPolicy;
15
16 public class Servidor {
17
18   /**
19    * @param args
20    * @throws InvalidName
21    * @throws AdapterInactive
22    * @throws WrongPolicy
23    * @throws ServantNotActive
24    * @throws FileNotFoundException
25    */
26   public static void main(String[] args) throws InvalidName,
         AdapterInactive, ServantNotActive, WrongPolicy,
         FileNotFoundException {
27     //inicializacion del orb
28     ORB orb = ORB.init(args, null);
29
30     //creacion del poa raiz
31     POA poa = POAHelper.narrow(orb.resolve_initial_references("
           RootPOA"));
32
33     //crear y activar sirvientes
34     poa.the_POAManager().activate();
35     gestorSubastasImpl sirviente = new gestorSubastasImpl(poa);
36     org.omg.CORBA.Object obj = poa.servant_to_reference(
           sirviente);
37
38     String ior = orb.object_to_string(obj);
39     PrintWriter ps = new PrintWriter(new FileOutputStream(new
           File("server.ior")));
40     ps.println(ior);
41     ps.close();
42
43     orb.run();
44   }
45
46 }
```

3.7. Cliente.java

```
1
2  package subastas;
3
4  import java.io.BufferedReader;
5  import java.io.FileNotFoundException;
6  import java.io.FileReader;
7  import java.io.IOException;
8  import java.io.InputStreamReader;
9  import java.io.LineNumberReader;
10
11 import org.omg.CORBA.*;
12 import org.omg.PortableServer.POA;
13 import org.omg.PortableServer.POAHelper;
14 import org.omg.PortableServer.POAPackage.ServantNotActive;
15 import org.omg.PortableServer.POAPackage.WrongPolicy;
16
17 public class Cliente {
18
19   /**
20    * @param args
21    * @throws IOException
22    */
23   private static clienteSubastas buscar(String id,
       clienteSubastas[] cs){
24   clienteSubastas cl = null;
25   for (int i=0; i<cs.length; i++){
26     if (cs[i].identificacion().contains(id)){
27     cl = cs[i];
28     break;
29     }
30   }
31   return cl;
32   }
33
34   private static clienteSubastas crearCliente(String ident, POA
       poa){
35   org.omg.CORBA.Object obj = null;
36   clienteSubastasImpl cl1x = new clienteSubastasImpl(ident);
37   try {
38     obj = poa.servant_to_reference(cl1x);
39   } catch (Exception e1) {e1.printStackTrace();}
40
41   clienteSubastas cl1 = clienteSubastasHelper.narrow(obj);
42   return cl1;
43   }
44
45   public static void main(String[] args) throws IOException {
```

```
46      ORB orb = ORB.init(args, null);
47
48      LineNumberReader input = new LineNumberReader(new FileReader(
            "server.ior"));
49      String ior = input.readLine();
50
51      org.omg.CORBA.Object obj = orb.string_to_object(ior);
52
53      gestorSubastas gs = gestorSubastasHelper.narrow(obj);
54
55      POA poa=null;
56      try{
57      //creacion del poa raiz
58        poa = POAHelper.narrow(orb.resolve_initial_references("
              RootPOA"));
59        //crear y activar sirvientes
60        poa.the_POAManager().activate();
61      }
62      catch (Exception e){e.printStackTrace();}
63
64      /**
65       * COMANDOS DISPONIBLES
66       *
67       * GESTOR SUBASTAS
68       * ---------------
69       * salir -> sale del bucle
70       * crear valor tiempo desc -> crea una subasta
71       * buscar desc -> busca una subasta y muestra todas las
             coincidencias
72       * borrar id -> borra una subasta si esta cerrada
73       *
74       * COORDINADOR SUBASTA
75       * -------------------
76       * abrir id -> abre la subasta por id
77       * estado id -> devuelve el estado de una subasta
78       * valor id -> devuelve el valor actual de una puja
79       * ganador_prov id -> dice el identificador del cliente que
             va ganando la puja
80       * ganador id -> muestra el ganador, si la puja no esta
             cerrada aun, bloquea al cliente
81       * clientes id -> muestra todos los identificadores de los
             clientes de la subasta
82       * inscribir id cliente_id -> inscribe un cliente en una
             subasta
83       * pujar id cliente_id valor
84       * ultima id cliente_id
85       *
86       * CLIENTE SUBASTAS
87       * ----------------
88       * crearcliente identificacion
```

```
89       * cldisp -> muestra los clientes disponibles
90       */
91
92      coordinadorSubasta[] subastas = null;
93      clienteSubastas[] clientes = new clienteSubastas[100];
94      int n_clientes = 0;
95
96      boolean salir = false;
97      String comando = "";
98      System.out.println("== INICIO ==");
99      while(!salir){
100       System.out.print(">>");
101       try{
102       BufferedReader br = new BufferedReader(new
              InputStreamReader(System.in));
103       comando = br.readLine();
104       }catch(Exception e){ e.printStackTrace();}
105       String[] partesDelComando = comando.split(" ");
106       if(comando.equals("salir")){
107       salir = true;
108       System.out.println("Adios ;)");
109       break;
110       }
111       else if(comando.equals("help")){
112       System.out.println(
113      "* COMANDOS DISPONIBLES\r\n" +
114      "*\r\n " +
115      "* GESTOR SUBASTAS\r\n"+
116      "* --------------\r\n"+
117      "* salir -> sale del bucle\r\n"+
118      "* crear valor tiempo desc -> crea una subasta\r\n"+
119      "* buscar desc -> busca una subasta y muestra todas las
             coincidencias\r\n"+
120      "* borrar id -> borra una subasta si esta cerrada\r\n"+
121      "* \r\n"+
122      "* COORDINADOR SUBASTA\r\n"+
123      "* -------------------\r\n"+
124      "* abrir id -> abre la subasta por id\r\n"+
125      "* estado id -> devuelve el estado de una subasta\r\n"+
126      "* valor id -> devuelve el valor actual de una puja\r\n"+
127      "* ganador_prov id -> dice el identificador del cliente que
             va ganando la puja\r\n"+
128      "* ganador id -> muestra el ganador, si la puja no esta
             cerrada aun, bloquea al cliente\r\n"+
129      "* clientes id -> muestra todos los identificadores de los
             clientes de la subasta\r\n"+
130      "* inscribir id cliente_id -> inscribe un cliente en una
             subasta\r\n"+
131      "* pujar id cliente_id valor\r\n"+
132      "* ultima id cliente_id\r\n"+
```

```
133       "* \r\n"+
134       "* CLIENTE SUBASTAS\r\n"+
135       "* ----------------\r\n"+
136       "* crearcliente identificacion\r\n"+
137       "* cldisp -> muestra los clientes disponibles\r\n");
138        }
139
140        // Opciones de gestorSubastas
141        else if(partesDelComando[0].equals("crear")){
142        String desc = "";
143        float valor = 0;
144        int time = 0;
145        valor = Float.valueOf(partesDelComando[1]);
146        time = Integer.valueOf(partesDelComando[2]);
147        for(int i=3; i<partesDelComando.length; i++){
148          desc += " "+partesDelComando[i];
149        }
150        coordinadorSubasta cs = gs.crearSubasta(valor, desc, time);
151        System.out.println("Subasta creada");
152        }
153        else if(partesDelComando[0].equals("buscar")){
154        String desc = "";
155        for(int i=1; i<partesDelComando.length; i++){
156          desc += " "+partesDelComando[i];
157        }
158
159        subastas = gs.localizarSubasta(desc);
160        for (int i=0; i<subastas.length; i++)
161          System.out.println(i+": "+subastas[i].descripcion()+" ("+
               subastas[i].valor()+") > " + subastas[i].estado());
162        }
163        else if(partesDelComando[0].equals("borrar")){
164        int id = 0;
165        id = Integer.valueOf(partesDelComando[1]);
166        gs.destruirSubasta(subastas[id]);
167        System.out.println("Subasta borrada");
168        }
169
170        // Opciones de coordinadorSubasta
171        else if(partesDelComando[0].equals("abrir")){
172        int id = 0;
173        id = Integer.valueOf(partesDelComando[1]);
174        subastas[id].abrirSubasta();
175        System.out.println(subastas[id].estado());
176        }
177        else if(partesDelComando[0].equals("estado")){
178        int id = 0;
179        id = Integer.valueOf(partesDelComando[1]);
180        System.out.println(subastas[id].estado());
181        }
```

```
182       else if(partesDelComando[0].equals("valor")){
183       int id = 0;
184       id = Integer.valueOf(partesDelComando[1]);
185       System.out.println(subastas[id].valor());
186       }
187       else if(partesDelComando[0].equals("ganador_prov")){
188       int id = 0;
189       id = Integer.valueOf(partesDelComando[1]);
190       System.out.println(subastas[id].ganador_provisional().
              identificacion());
191       }
192       else if(partesDelComando[0].equals("ganador")){
193       int id = 0;
194       id = Integer.valueOf(partesDelComando[1]);
195       System.out.println(subastas[id].ganador().identificacion())
              ;
196       }
197       else if(partesDelComando[0].equals("clientes")){
198       int id = 0;
199       id = Integer.valueOf(partesDelComando[1]);
200       clienteSubastas[] clientes2 = subastas[id].clientes();
201       for (int i=0; i< clientes2.length; i++){
202         System.out.println(i+": "+clientes2[i].identificacion());
203       }
204       }
205       else if(partesDelComando[0].equals("inscribir")){
206       int id = 0;
207       String cliente_id="";
208       clienteSubastas cl=null;
209       id = Integer.valueOf(partesDelComando[1]);
210       cliente_id = partesDelComando[2];
211       cl = buscar(cliente_id, clientes);
212       subastas[id].inscribirCliente(cl);
213       System.out.println("Cliente inscrito");
214       }
215
216       else if(partesDelComando[0].equals("pujar")){
217       int id = 0;
218       float valor = 0;
219       String cliente_id="";
220       clienteSubastas cl=null;
221       id = Integer.valueOf(partesDelComando[1]);
222       cliente_id = partesDelComando[2];
223       valor = Float.valueOf(partesDelComando[3]);
224       cl = buscar(cliente_id, clientes);
225       if(subastas[id].pujar(valor, cl))
226         System.out.println("Vas ganando");
227       else
228         System.out.println("No es suficiente");
229       }
```

```
230       else if(partesDelComando[0].equals("ultima")){
231       int id = 0;
232       String cliente_id="";
233       clienteSubastas cl=null;
234       id = Integer.valueOf(partesDelComando[1]);
235       cliente_id = partesDelComando[2];
236       cl = buscar(cliente_id, clientes);
237       subastas[id].ultimaPuja(cl);
238       System.out.println("Has dejado de pujar");
239       System.out.println(subastas[id].estado());
240       }
241
242       // creacion de clientes
243       else if(partesDelComando[0].equals("crearcliente")){
244       String id="";
245       clienteSubastas cl=null;
246       id = partesDelComando[1];
247       cl = crearCliente(id, poa);
248       clientes[n_clientes++] = cl;
249       }
250       else if(partesDelComando[0].equals("cldisp")){
251       for(int i=0; i<n_clientes; i++){
252         System.out.println(i+": "+clientes[i].identificacion());
253       }
254       }
255
256     }
257
258     System.out.println("FIN");
259     }
260
261   }
```

4. Anexos (Cliente python)

Dado el interés que esta asignatura ha despertado en mí, me he decidido a implementar un cliente en otro lenguaje diferente a java para probar la interoperabilidad entre diferentes lenguajes del middleware CORBA.

Después de implementar los sirvientes en java, la implementación de un cliente en python que se conecte al servidor en java no ha sido nada complicado.

Este cliente en python no es un cliente unificado sino que sólo permite manejar un cliente que se puede inscribir a subastas y pujar, no se pueden manejar las subastas desde este.

4.1. Código del cliente python

4.1.1. cliente.py

```
1
2   #!/usr/bin/python
3
4   import sys
5   import CORBA, subastas
6   import PortableServer
7   from clienteSubastas_i import *
8
9   orb = CORBA.ORB_init()
10  ior = open('../server.ior').readline()
11
12  gestor_subasta = orb.string_to_object(ior)
13
14  poa = orb.resolve_initial_references("RootPOA")
15  poa._get_the_POAManager().activate()
16
17
18  def crear_cliente(id, poa):
19      cl = clienteSubastas_i(id)
20      cl1 = poa.servant_to_reference(cl)
21      return cl1
22
23  subastas = gestor_subasta.localizarSubasta('')
24  try:
25      nombre = sys.argv[1]
26  except:
27      nombre = raw_input("nombre del cliente: ")
28  cl = crear_cliente(nombre, poa)
29
30  while True:
31      comando = raw_input(">>")
32      if comando.split()[0] == 'salir':
33          break
34      elif comando.split()[0] == 'inscribir':
35          subasta = subastas[int(comando.split()[1])]
36          subasta.inscribirCliente(cl)
37      elif comando.split()[0] == 'pujar':
38          subasta = subastas[int(comando.split()[1])]
39          valor = float(comando.split()[2])
40          if subasta.pujar(valor, cl):
41              print "Vas ganando"
42          else:
43              print "No es suficiente"
44      elif comando.split()[0] == 'ultima':
45          subasta = subastas[int(comando.split()[1])]
```

```
46          subasta.ultimaPuja(cl)
47          print "Has dejado de pujar"
48
49      elif comando.split()[0] == 'buscar':
50          try:
51              desc = comando.split()[1]
52          except:
53              desc = ''
54          subastas = gestor_subasta.localizarSubasta(desc)
55          for i,s in enumerate(subastas):
56              print i, s._get_descripcion(), s._get_estado()
57
58      elif comando.split()[0] == 'ganador':
59          subasta = subastas[int(comando.split()[1])]
60          print subasta.ganador()._get_identificacion()
61
62      elif comando.split()[0] == 'ganador_prov':
63          subasta = subastas[int(comando.split()[1])]
64          print subasta._get_ganador_provisional().
                _get_identificacion()
65
66      elif comando.split()[0] == 'estado':
67          subasta = subastas[int(comando.split()[1])]
68          print subasta._get_estado()
69
70      elif comando.split()[0] == 'valor':
71          subasta = subastas[int(comando.split()[1])]
72          print subasta._get_valor()
73
74      elif comando.split()[0] == 'clientes':
75          subasta = subastas[int(comando.split()[1])]
76          clientes = subasta._get_clientes()
77          for i, c in enumerate(clientes):
78              print i, c._get_identificacion()
79
80      elif comando.split()[0] == 'help':
81          help_str = '''
82          COMANDOS DISPONIBLES
83          --------------------
84          buscar desc -> muestra la lista de subastas
85          ganador id -> muestra el ganador de una subasta
86          inscribir id -> te inscribe en una puja
87          pujar id cantidad -> puja cantidad en la puja con id
88          ultima id -> dejas de pujar en la puja id
89          ganador_prov id -> muestra el ganador provisional
90          estado id -> muestra el estado de una puja
91          valor id -> muestra el valor actual de una puja
92          clientes id -> muestra los clientes que hay en una puja
93          salir -> sale del cliente
94          help -> muestra esta ayuda
```

```
95
96          '''
97          print help_str
```

4.1.2. cliente.py

```
1
2   import subastas__POA
3
4   class clienteSubastas_i(subastas__POA.clienteSubastas):
5       def __init__(self, id):
6           self.identificacion = id
7
8       def finSubasta(self):
9           pass
10
11      def nuevoValor(self, valor):
12          print "NUEVO VALOR: ", valor
13
14      def _get_identificacion(self):
15          return self.identificacion
```