Práctica sobre compartición de instancias remotas.

Para esta práctica se ha construido un pequeño sistema cliente-servidor que permite la resolución de Sudokus entre varios jugadores.

El servidor consta de una clase *SudokuServer* que recoge, a través de la interfaz *Remote ISudokuServer*, los "movimientos" que realizan los jugadores. Éstos, cuando colocan o quitan un número en alguna casilla de algún *Sudoku*, envían al *SudokuServer* el mensaje *poner* o *quitar*. Igualmente, pueden recuperar la lista de Sukous de que dispone el servidor enviando el mensaje *getSudokus*, que devuelve un Vector con una lista de cadenas, que se corresponden con los identificadores de cada sudoku.

A partir de esta lista, el cliente puede solicitar un sudoku determinado ejecutando *getSudoku(sudoku:String)*, operación a la que se pasa como parámetro el identificador del sudoku solicitado. Al recibir el mensaje, el *SudokuServer* busca el sudoku solicitado en una tabla hash en la que almacena los sudokus ya materializados: si lo encuentra, lo recupera de la tabla y devuelve una representación del objeto de clase *Sudoku* en forma de cadena; si no lo tiene, lo materializa a través del *Agente* y devuelve la representación del objeto en forma de cadena.

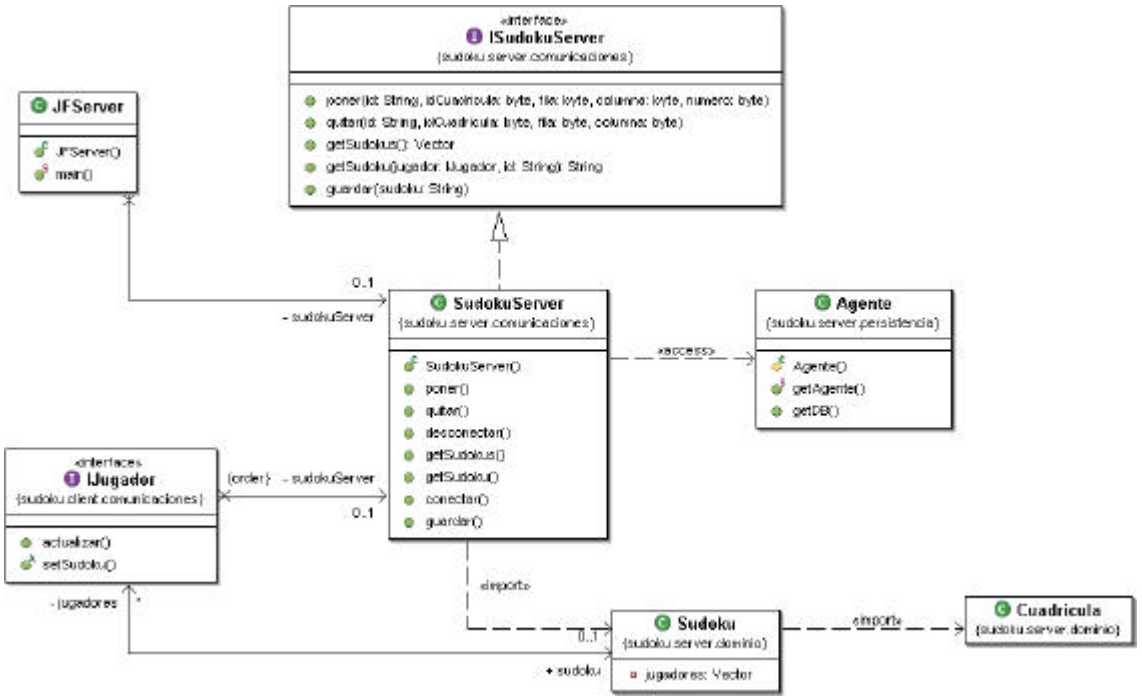

Figura 1. Diseño del servidor

La ventana del cliente tiene el aspecto mostrado en la Figura 2.

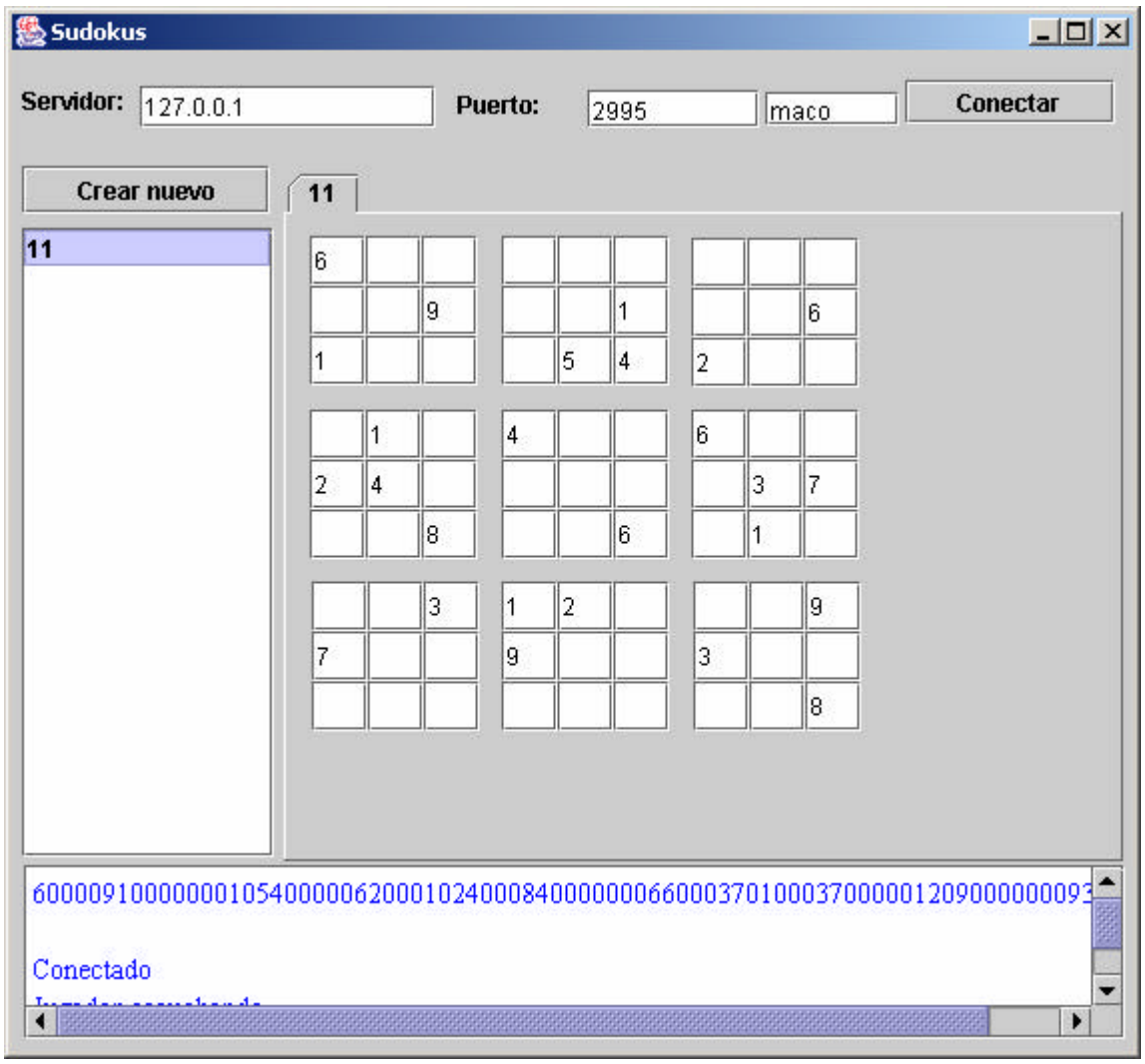

Figura 2. Ventana del cliente

La ventana mostrada se corresponde con una instancia de la clase *JFSudoku* de la Figura 3. *JFSudoku* conoce al *Proxy*, que es la clase que actúa de intermediario de comunicación entre el cliente y el servidor (obsérvese que está conectado a la interfaz *ISudokuServer*, la interfaz remota ofrecida por el servidor y que se mostraba en la Figura 1).

JFSudoku tiene un panel de tipo *JPTablero* por cada Sudoku cargado; cada *JPTablero* tiene a su vez nueve instancias de tipo *JPCuadricula*, que se corresponden con las nueve cuadrículas del juego.

Cuando se arranca la aplicación (método *main* de *JFSudoku)*, se crea el *proxy* asociado que, a su vez, crea una instancia de *Jugador*, que es también un objeto remoto al que notifica el servidor cada vez que se produce un cambio en un *Sudoku*. Así, un escenario típico de ejecución en el que se combina el cliente con el servidor puede ser el siguiente:

1) El jugador pone un número en una posición de cierto sudoku. Para ello, utiliza una de las cajitas de texto situadas en alguna de las *JPCuadriculas.*
2) La *JPCuadricula* le dice a la *IVentanaJugador* a la cual conoce (que, como se observa en la figura, es la clase *JFSudoku)* que se ha colocado tal número en tal posición.
3) La ventana (*JFSudoku)* le dice al proxy que en tal sudoku se ha puesto tal número en tal posición.
4) El proxy se lo dice al *SudokuServer* vía rmi ejecutando el métod *poner*.
5) El SudokuServer toma de su caché el Sudoku en cuestión, le coloca el número y actualiza su estado en la base de datos.
6) Acto seguido, la instancia de *Sudoku* que corresponda ejecuta su operación *notificar*, que recorre la colección de *jugadores* (realmente, referencias remotas al *Jugador* que tiene todo cliente) avisando del cambio que se ha producido en ese *Sudoku.*
7) El *Jugador* se lo dice a la *IVentanaJugador* a la que conoce, que está implementada por su *JFSudoku.*

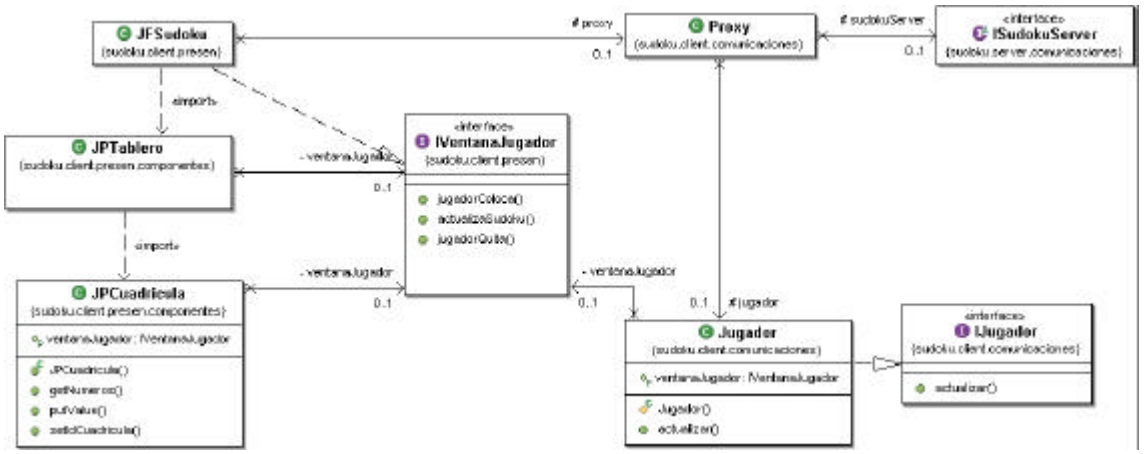

Figura 3. Diseño del cliente

Los Sudokus se devuelven a los clientes cuando hacen doble clic en la listilla que aparece en el lado izquierdo de sus ventanas (Figura 2). El efecto es que se envía, a través del proxy, una solicitud al *SudokuServer* que se traduce en una llamada a la operación *getSudoku(id : String) : String*, que devuelve el *Sudoku* en forma de cadena. La implementación de esta operación y de otras dos auxiliares aparece en la Figura 4: como se observa, primero se busca el *Sudoku* solicitado en la *cacheSudokus* (implementada como una tabla hash): si se encuentra, se devuelve la cadena que corresponda, se añade al jugador que la ha solicitado a la lista de jugadores que están

pendientes de este sudoku; si no, se materializa (llamada al constructor *new Sudoku(id)*), se coloca en la caché, se añade el jugador a la lista de jugadores y finalmente se devuelve la cadena.

```
public String getSudoku(IJugador jugador, String id)
                    throws RemoteException, SQLException {
       Sudoku result=this.findSudoku(id);
       if (result==null) {
              result=new Sudoku(id);
              this.putSudoku(result);
       }
       result.add(jugador);
       return result.toString();
}

private Sudoku findSudoku(String id) {
       return (Sudoku) this.cacheSudokus.get(""+id);
}

private void putSudoku(Sudoku result) {
       this.cacheSudokus.put(""+result.getId(), result);
}
```

Figura 4. Implementación de *getSudoku* en *SudokuServer*

En la Figura 5, el jugador de la ventana que está en primer plano coloca un 8 en la primera cuadrícula: el mensaje se envía al servidor, que notifica a las partes interesadas (las dos ventanas que se están mostrando).

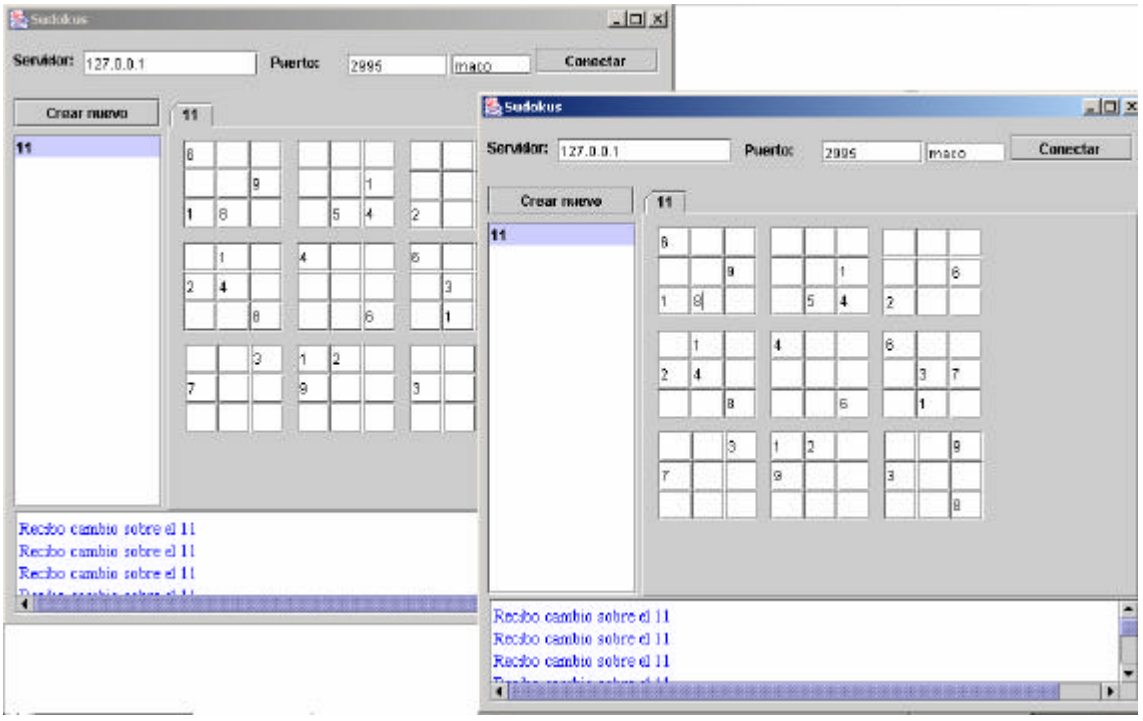

Figura 5. Ejemplo

El servidor almacena los sudokus en una base de datos implementada en un servidor SQL Server que consta de una sola tabla, cuya estructura se muestra en la siguiente figura:

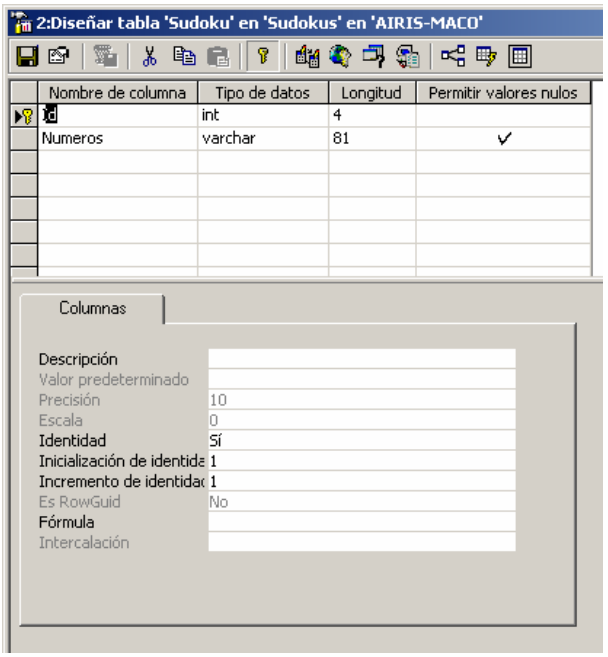

Figura 6. Diseño de la tabla Sudokus

Como se observa, de cada sudoku se almacena su identificador y los 81 números que la componen, que se guardan en forma de cadena. Las casillas en blanco se almacenan como el valor cero.

A continuación se ofrece el código de la clase *Sudoku* (Figura 7). Como se veía en la Figura 1, el Sudoku tiene nueve cuadrículas (objetos de clase *Cuadricula*), cada una de las cuales tiene una matriz de 3x3 bytes. El constructor sin parámetros inicaliza las 9 cuadrículas; el constructor con un parámetro recupera de la base de datos el sudoku cuyo identificador se pasa y coloca los 81 caracteres en las nueve cuadrículas mediante el método *setNumeros(ristra:String).* El método *toString():String*, que redefine la especialización que se hereda implícitamente de *Object*, devuelve el sudoku en forma de cadena; *poner* y *quitar* ponen y quitan, respectivamente, un número en la cuadrícula, fila y columna que se pasen, y luego notifican el cambio a todos los jugadores que están pendientes de este sudoku; *insert* inserta un sudoku nuevo en la base de datos; *update* lo actualiza. Respecto de estas dos últimas operaciones, es interesante notar que no se sigue ninguno de los patrones de transformación habituales para mantener la correspondencia entre las clases y las tablas.

```
public class Sudoku {
   protected int id;
   protected Cuadricula[] cuadriculas;
   protected Vector jugadores;

   public Sudoku() {
      cuadriculas=new Cuadricula[9];
      for (int i=0; i<9; i++)
         cuadriculas[i]=new Cuadricula();
```

```
}

public Sudoku(String id) throws SQLException {
   this();
   String SQL="Select Numeros from Sudoku where Id=?";
   PreparedStatement p=
      Agente.getAgente().getDB().prepareStatement(SQL);
   p.setInt(1, Integer.parseInt(id));
   ResultSet r=p.executeQuery();
   if (r.next()) {
      this.id=Integer.parseInt(id);
      String ristra=r.getString(1);
      this.setNumeros(ristra);
   }
   jugadores=new Vector();
   p.close();
}

public void setNumeros(String ristra) {
   byte fila=0, columna=-1;
   for (byte i=0; i<ristra.length(); i++) {
      String c=""+ristra.charAt(i);
      byte idCuadricula=(byte) (i/9);
      if (idCuadricula>0 && i%9==0) {
         fila=0;
         columna=0;
      } else {
         columna++;
         if (columna==3) {
            fila++;
            columna=0;
         }
      }
      if (c.equals(" "))
         cuadriculas[idCuadricula].colocar(fila, columna, (byte) 0);
      else {
         byte b=Byte.parseByte(c);
         cuadriculas[idCuadricula].colocar(fila, columna, b);
      }
   }
}

public String toString() {
   String result="";
   for (int i=0; i<9; i++) {
      result+=cuadriculas[i].toString();
   }
   return result;
}

public void add(IJugador jugador) {
   this.jugadores.add(jugador);
}

public void poner(byte idCuadricula, byte fila, byte columna, byte numero)
        throws RemoteException {
   this.cuadriculas[idCuadricula].colocar(fila, columna, numero);
   update();
   notificar();
}


public void quitar(byte idCuadricula, byte fila, byte columna)
                                              throws RemoteException {
   this.cuadriculas[idCuadricula].colocar(fila, columna, (byte) 0);
   update();
   notificar();
}
```

```
   public void insert() throws SQLException {
      String SQL="Insert into Sudoku (Numeros) values (?)";
      PreparedStatement p=Agente.getAgente().getDB().prepareStatement(SQL);
      p.setString(1, this.toString());
      p.executeUpdate();
      p.close();
   }

   private void update() throws RemoteException {
      String SQL="Update Sudoku set Numeros=? where Id=?";
      try {
         PreparedStatement p=Agente.getAgente().getDB().prepareStatement(SQL);
         p.setString(1, this.toString());
         p.setInt(2, this.id);
         p.executeUpdate();
         p.close();
      } catch (SQLException e) {
         throw new RemoteException(e.toString());
      }
   }

   private void notificar() throws RemoteException {
      for (int i=0; i<jugadores.size(); i++) {
         try {
            IJugador jugador=(IJugador) jugadores.get(i);
            jugador.actualizar(this.id, this.toString());
         }
         catch (Exception ex) {
         }
      }
   }
}
```

Figura 7. Código de la clase *Sudoku*, situada en el servidor

Distribución y puesta en marcha.

Ambas aplicaciones (cliente y servidor) se han construido utilizando Eclipse, y se encuentran en los ficheros *SudokuCliente.zip* y *SudokuServer.zip*.

Puesto que el cliente y el servidor se conocen mutuamente (el cliente cuando modifica un sudoku; el servidor para notificar cambios a los clientes), es preciso publicar crear sendos ficheros *jar* con los *stubs.*

Para tal fin, se adjuntan también los ficheros *SudokuServerStubs.jar* (necesario para el cliente), y *SudokuClientStubs.jar* (necesario para el servidor). Para generar los stubs se utilizan estos comandos (se asume que los proyectos *SudokuClient* y *SudokuServer* cuelgan de *c:\sudokus*):

```
rmic -classpath c:\sudokus\SudokuClient\;. sudoku.server .comunicaciones.SudokuServer
rmic -classpath c:\sudokus\SudokuServer\;. sudoku.client.comunicaciones.Jugador
```

Los proyectos pueden ejecutarse desde fuera. Para ello, se incluyen los ficheros ejecutables *SudokuServer.exe.jar* y *SudokuClient..exe.jar*.

Es importante recordar que el servidor requiere disponer en local (en otro caso, debería modificarse la dirección IP del servidor en el código del *Agente*) de SQL Server que disponga de una base de datos llamada *Sudokus* con una tabla *Sudoku,* que tenga la estructura mostrada en la Figura 6.