Capítulo VI

Anexos

1. Anexo I: Archivo de simulación: teleasistencia.xml

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <simulation xmlns="http://www.lucent.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance" xsi:schemaLocation="http://www.lucent.com simulation.xsd" name="Simulacion de
    Teleasistencia">
3
4           <subscriberSet>
5
6                   <subscriber name="Fernando">
7                           <MSISDN>150111111111</MSISDN>
8                   </subscriber>
9
10                  <subscriber name="Carmen">
11                          <MSISDN>150222222222</MSISDN>
12                  </subscriber>
13
14                  <subscriber name="Antonia">
15                          <MSISDN>150333333333</MSISDN>
16                  </subscriber>
17
18                  <subscriber name="Manuel">
19                          <MSISDN>150444444444</MSISDN>
20                  </subscriber>
21
22                  <subscriber name="Central">
23                          <MSISDN>160111111111</MSISDN>
24                  </subscriber>
25                  <subscriber name="Operadora">
26                          <MSISDN>160222222222</MSISDN>
27                  </subscriber>
28                  <subscriber name="numeroAlarmasCaida">
29                          <MSISDN>160333333333</MSISDN>
30                  </subscriber>
31                  <subscriber name="numeroCambiosEstadoPAciente">
32                          <MSISDN>160444444444</MSISDN>
33                  </subscriber>
34
35                  <subscriber name="Cuidador1">
36                          <MSISDN>170111111111</MSISDN>
37                  </subscriber>
38                  <subscriber name="Cuidador2">
39                          <MSISDN>170222222222</MSISDN>
40                  </subscriber>
41                  <subscriber name="Cuidador3">
42                          <MSISDN>170333333333</MSISDN>
43                  </subscriber>
44
45                  <subscriber name="Ambulancia1">
46                          <MSISDN>180111111111</MSISDN>
47                  </subscriber>
48                  <subscriber name="Ambulancia2">
49                          <MSISDN>180222222222</MSISDN>
```

```
50                      </subscriber>
51
52              </subscriberSet>
53
54              <behaviorSet>
55
56                      <!-- Define comportamientos que se pueden referenciar en cualquier lugar del
    script -->
57                      <ccBehaviorEndCall name="end_by_caller" byCaller="true"/>
58                      <ccBehaviorEndCall name="end_by_callee" byCaller="false"/>
59
60                      <ccBehaviorStatic name="available" act="willAnswer" answerDelay="4000"/>
61                      <ccBehaviorStatic name="refuser" act="willRefuse"/>
62
63                      <!-- Datos de localizacion de usuarios -->
64                      <ulBehaviorStatic name="ul_Fernando" longitude="-5.98466"
    latitude="37.38911"/>
65                      <ulBehaviorStatic name="ul_Carmen" longitude="-5.97153" latitude="37.39874"/>
66                      <ulBehaviorStatic name="ul_Antonia" longitude="-5.92810" latitude="37.38381"/>
67                      <ulBehaviorStatic name="ul_Manuel" longitude="-5.95931" latitude="37.37933"/>
68
69                      <ulBehaviorStatic name="ul_Cuidador1" longitude="-5.97125"
    latitude="37.39050"/>
70                      <ulBehaviorStatic name="ul_Cuidador2" longitude="-5.95282"
    latitude="37.39764"/>
71                      <ulBehaviorStatic name="ul_Cuidador3" longitude="-5.95135"
    latitude="37.38389"/>
72
73                      <!-- Datos de localizacion de la central -->
74                      <ulBehaviorStatic name="ulCentral" longitude="-5.95589" latitude="37.39038"/>
75
76                      <!-- Datos de localizacion y de movimiento de las ambulancias -->
77                      <ulBehaviorRandom name="randomAmbulancia1" speed="100">
78                              <start longitude="-5.9722" latitude="37.4103"/>
79                              <end longitude="-6.0056" latitude="37.3969"/>
80                      </ulBehaviorRandom>
81
82                      <ulBehaviorRandom name="randomAmbulancia2" speed="200">
83                              <start longitude="-5.9811" latitude="37.3864"/>
84                              <end longitude="-5.9786" latitude="37.375"/>
85                      </ulBehaviorRandom>
86
87                      <!-- Todos los terminales, ya sean fijos o moviles, encendidos al iniciarse -->
88                      <usBehaviorStatic name="usMobile" status="ON" terminal="MOBILE"/>
89                      <usBehaviorStatic name="usFixed" status="ON" terminal="FIXED"/>
90
91              </behaviorSet>
92
93              <scsSet>
94                      <ccState>
95                              <!--  Establece el retraso, para el simulador CC, entre una solicitud y
    una respuesta a traves de la interfaz OSA Parlay -->
96                              <latency>50</latency>
97
```

```
98                              <!-- El simulador, despues de 30s, considerara la llamada como no
    respondida -->
99                              <parameters noAnswerTime="30000"/>
100                     </ccState>
101
102                     <ulState>
103                             <!--  Establece el retraso, para el simulador UL, entre una solicitud y
    una respuesta a traves de la interfaz OSA Parlay -->
104                             <latency>50</latency>
105                             <!-- This would generate simulated errors, so set to 0 -->
106                             <fixedErrorFilter percentage="0"/>
107                             <networkModel>
108                                     <!-- reports real position, modified by uncertainty radius -->
109                                     <ideal uncertainty="0"/>
110                             </networkModel>
111                     </ulState>
112
113                     <usState>
114                             <!--  sets the delay for the US SCS simulator between a request and
    response through the OSA/Parlay interface -->
115                             <latency>50</latency>
116                     </usState>
117
118                     <uiState>
119                             <!-- sets the initial state of the UI simulator -->
120                             <!-- inherited tags: latency, defaultSubscriberState and requestTimeout
    -->
121                             <!-- list of available announcements referenced by application -->
122                             <uiInfoSet>
123                                     <uiInfo id="1" sound="announce.wav" text="Pulse: 1-Paciente
    asistido. 2-No sabe nada." duration="10000"/>
124                                     <uiInfo id="2" sound="announce.wav" text="Muchas Gracias!
    Se tomaran las medidas oportunas." duration="4000"/>
125                                     <uiInfo id="3" sound="announce.wav" text="La opcion
    escogida no es valida." duration="4000"/>
126                                     <uiInfo id="4" sound="announce.wav" text="Demasiado
    tarde." duration="4000"/>
127                             </uiInfoSet>
128                     </uiState>
129
130             </scsSet>
131
132             <stepSet>
133                     <!-- Registro del Servicio de Teleasistencia con el Framework -->
134                     <step name="Registrar_Servicio_Teleasistencia">
135                             <fwStep>
136                                     <fwState>
137                                             <applicationId>
138                                                     <create id="Servicio de Teleasistencia"
    secret="00010203040506070809">
139                                                             <service
    name="P_USER_LOCATION"/>
140                                                             <service
    name="P_USER_STATUS"/>
```

```
141                                                     <service
    name="P_USER_INTERACTION"/>
142                                                     <service
    name="P_GENERIC_CALL_CONTROL"/>
143                                                     <service
    name="P_MULTI_MEDIA_MESSAGING"/>
144                                                 </create>
145                                             </applicationId>
146                                         </fwState>
147                                     </fwStep>
148                             </step>
149
150                             <!-- Aniade los usuarios al Simulador CC -->
151                             <step name="Anadir_CC_Suscribers">
152                                     <ccStep>
153                                         <addSubscriber>
154                                             <subscriberRef>Central</subscriberRef>
155                                         </addSubscriber>
156                                         <addSubscriber>
157                                             <subscriberRef>Cuidador1</subscriberRef>
158                                         </addSubscriber>
159                                         <addSubscriber>
160                                             <subscriberRef>Cuidador2</subscriberRef>
161                                         </addSubscriber>
162                                         <addSubscriber>
163                                             <subscriberRef>Cuidador3</subscriberRef>
164                                         </addSubscriber>
165                                         <addSubscriber>
166                                             <subscriberRef>Operadora</subscriberRef>
167                                         </addSubscriber>
168                                         <addSubscriber>
169
            <subscriberRef>numeroAlarmasCaida</subscriberRef>
170                                         </addSubscriber>
171                                         <addSubscriber>
172
            <subscriberRef>numeroCambiosEstadoPAciente</subscriberRef>
173                                         </addSubscriber>
174                                         <addSubscriber>
175                                             <subscriberRef>Fernando</subscriberRef>
176                                         </addSubscriber>
177                                         <addSubscriber>
178                                             <subscriberRef>Carmen</subscriberRef>
179                                         </addSubscriber>
180                                         <addSubscriber>
181                                             <subscriberRef>Antonia</subscriberRef>
182                                         </addSubscriber>
183                                         <addSubscriber>
184                                             <subscriberRef>Manuel</subscriberRef>
185                                         </addSubscriber>
186                                         <addSubscriber>
187                                             <subscriberRef>Ambulancia1</subscriberRef>
188                                         </addSubscriber>
189                                         <addSubscriber>
```

```
190                                         <subscriberRef>Ambulancia2</subscriberRef>
191                                 </addSubscriber>
192                         </ccStep>
193                 </step>
194
195                 <!-- Aniade losusuarios al simulador UL -->
196                 <step name="Establecer_User_Location">
197                         <ulStep>
198                                 <!-- Se establecen laas posiciones iniciales para los enfermos
    -->
199                                 <ulSubscriberState>
200                                         <subscriberRef>Fernando</subscriberRef>
201                                         <behaviorRef>ul_Fernando</behaviorRef>
202                                 </ulSubscriberState>
203                                 <ulSubscriberState>
204                                         <subscriberRef>Carmen</subscriberRef>
205                                         <behaviorRef>ul_Carmen</behaviorRef>
206                                 </ulSubscriberState>
207                                 <ulSubscriberState>
208                                         <subscriberRef>Antonia</subscriberRef>
209                                         <behaviorRef>ul_Antonia</behaviorRef>
210                                 </ulSubscriberState>
211                                 <ulSubscriberState>
212                                         <subscriberRef>Manuel</subscriberRef>
213                                         <behaviorRef>ul_Manuel</behaviorRef>
214                                 </ulSubscriberState>
215
216                                 <!-- Posicion inicial para la Central -->
217                                 <ulSubscriberState>
218                                         <subscriberRef>Central</subscriberRef>
219                                         <behaviorRef>ulCentral</behaviorRef>
220                                 </ulSubscriberState>
221
222                                 <!-- Posicion inicial para las ambulancias -->
223                                 <ulSubscriberState>
224                                         <subscriberRef>Ambulancia1</subscriberRef>
225                                         <behaviorRef>randomAmbulancia1</behaviorRef>
226                                 </ulSubscriberState>
227                                 <ulSubscriberState>
228                                         <subscriberRef>Ambulancia2</subscriberRef>
229                                         <behaviorRef>randomAmbulancia2</behaviorRef>
230                                 </ulSubscriberState>
231
232                                 <!-- Posicion inicial para los cuidadores -->
233                                 <ulSubscriberState>
234                                         <subscriberRef>Cuidador1</subscriberRef>
235                                         <behaviorRef>ul_Cuidador1</behaviorRef>
236                                 </ulSubscriberState>
237                                 <ulSubscriberState>
238                                         <subscriberRef>Cuidador2</subscriberRef>
239                                         <behaviorRef>ul_Cuidador2</behaviorRef>
240                                 </ulSubscriberState>
241                                 <ulSubscriberState>
242                                         <subscriberRef>Cuidador3</subscriberRef>
```

```
243                                         <behaviorRef>ul_Cuidador3</behaviorRef>
244                                 </ulSubscriberState>
245
246                         </ulStep>
247                 </step>
248                 <step name="Establecer_User_Status">
249                         <usStep>
250                                 <!-- Todos los usuarios con el dispositivo movil o fijo
    encendido -->
251                                 <usSubscriberState>
252                                         <subscriberRef>Fernando</subscriberRef>
253                                         <behaviorRef>usMobile</behaviorRef>
254                                 </usSubscriberState>
255                                 <usSubscriberState>
256                                         <subscriberRef>Carmen</subscriberRef>
257                                         <behaviorRef>usMobile</behaviorRef>
258                                 </usSubscriberState>
259                                 <usSubscriberState>
260                                         <subscriberRef>Antonia</subscriberRef>
261                                         <behaviorRef>usMobile</behaviorRef>
262                                 </usSubscriberState>
263                                 <usSubscriberState>
264                                         <subscriberRef>Manuel</subscriberRef>
265                                         <behaviorRef>usMobile</behaviorRef>
266                                 </usSubscriberState>
267                                 <usSubscriberState>
268                                         <subscriberRef>Ambulancia1</subscriberRef>
269                                         <behaviorRef>usMobile</behaviorRef>
270                                 </usSubscriberState>
271                                 <usSubscriberState>
272                                         <subscriberRef>Ambulancia2</subscriberRef>
273                                         <behaviorRef>usMobile</behaviorRef>
274                                 </usSubscriberState>
275                                 <usSubscriberState>
276                                         <subscriberRef>Cuidador1</subscriberRef>
277                                         <behaviorRef>usMobile</behaviorRef>
278                                 </usSubscriberState>
279                                 <usSubscriberState>
280                                         <subscriberRef>Cuidador2</subscriberRef>
281                                         <behaviorRef>usMobile</behaviorRef>
282                                 </usSubscriberState>
283                                 <usSubscriberState>
284                                         <subscriberRef>Cuidador3</subscriberRef>
285                                         <behaviorRef>usMobile</behaviorRef>
286                                 </usSubscriberState>
287
288
289                                 <usSubscriberState>
290                                         <subscriberRef>Central</subscriberRef>
291                                         <behaviorRef>usFixed</behaviorRef>
292                                 </usSubscriberState>
293                                 <usSubscriberState>
294                                         <subscriberRef>Operadora</subscriberRef>
295                                         <behaviorRef>usFixed</behaviorRef>
```

```
296                                     </usSubscriberState>
297                                     <usSubscriberState>
298
                <subscriberRef>numeroAlarmasCaida</subscriberRef>
299                                             <behaviorRef>usFixed</behaviorRef>
300                                     </usSubscriberState>
301                                     <usSubscriberState>
302
                <subscriberRef>numeroCambiosEstadoPAciente</subscriberRef>
303                                             <behaviorRef>usFixed</behaviorRef>
304                                     </usSubscriberState>
305                             </usStep>
306                     </step>
307
308                     <step name="Central_llama_170111111111">
309                       <ccStep>
310                                     <ccSubscriberState>
311                                             <subscriberRef>Cuidador1</subscriberRef>
312                                             <ccBehaviorSingleCall number="160111111111"/>
313                                     </ccSubscriberState>
314
315                                     <ccSubscriberState>
316                                             <subscriberRef>Central</subscriberRef>
317                                             <behaviorRef>available</behaviorRef>
318                                     </ccSubscriberState>
319                             </ccStep>
320                     </step>
321
322                     <step name="Central_llama_170222222222">
323                       <ccStep>
324                                     <ccSubscriberState>
325                                             <subscriberRef>Cuidador2</subscriberRef>
326                                             <ccBehaviorSingleCall number="160111111111"/>
327                                     </ccSubscriberState>
328
329                                     <ccSubscriberState>
330                                             <subscriberRef>Central</subscriberRef>
331                                             <behaviorRef>available</behaviorRef>
332                                     </ccSubscriberState>
333                             </ccStep>
334                     </step>
335                     <step name="Central_llama_170333333333">
336                       <ccStep>
337                                     <ccSubscriberState>
338                                             <subscriberRef>Cuidador3</subscriberRef>
339                                             <ccBehaviorSingleCall number="160111111111"/>
340                                     </ccSubscriberState>
341
342                                     <ccSubscriberState>
343                                             <subscriberRef>Central</subscriberRef>
344                                             <behaviorRef>available</behaviorRef>
345                                     </ccSubscriberState>
346                             </ccStep>
347                     </step>
```

```
348                 <step name="Central_llama_180111111111">
349                   <ccStep>
350                               <ccSubscriberState>
351                                         <subscriberRef>Central</subscriberRef>
352                                         <ccBehaviorSingleCall number="180111111111"/>
353                               </ccSubscriberState>
354
355                               <ccSubscriberState>
356                                         <subscriberRef>Ambulancia1</subscriberRef>
357                                         <behaviorRef>available</behaviorRef>
358                               </ccSubscriberState>
359                         </ccStep>
360                 </step>
361                 <step name="Central_llama_180222222222">
362                   <ccStep>
363                               <ccSubscriberState>
364                                         <subscriberRef>Central</subscriberRef>
365                                         <ccBehaviorSingleCall number="180222222222"/>
366                               </ccSubscriberState>
367
368                               <ccSubscriberState>
369                                         <subscriberRef>Ambulancia2</subscriberRef>
370                                         <behaviorRef>available</behaviorRef>
371                               </ccSubscriberState>
372                         </ccStep>
373                 </step>
374
375       </stepSet>
376
377       <programSet>
378                 <program name="setup">
379                         <!-- clears current simulation -->
380                         <programStep seq="0">
381                                   <clear/>
382                                   <delay>0</delay>
383                         </programStep>
384                         <programStep seq="10">
385                                   <stepRef>Registrar_Servicio_Teleasistencia</stepRef>
386                                   <delay>0</delay>
387                         </programStep>
388                         <programStep seq="20">
389                                   <stepRef>Anadir_CC_Suscribers</stepRef>
```

2. Anexo II: Código de la Aplicación

2.1 Teleasistencia.java

```
1   /*
2    * Proyecto Fin de Carrera: Servicio de Teleasistencia basado OSA/Parlay
3    *
4    * TeleAsistencia.java
5    */
6
7   import BaseDeDatos.BBDD;
8   import EstadoTerminal.EstadoUsuarioLis;
9   import Gestion.*;
10  import InteraccionUsuario.InteraccionUsuarioLoc;
11  import Localizacion.LocalizacionUsuario;
12  import MensajeriaMultimedia.MensajeMultimediaLis;
13  import TipoDatos.*;
14  import com.lucent.isg.appsdk.framework.*;
15  import com.lucent.isg.appsdk.userlocation.*;
16  import java.util.*;
17
18  /**
19   * Clase principal del servicio de tele asistencia. Contiene el método Main().
20   * @author Raúl Parras Eliche.
21   */
22  public class TeleAsistencia {
23
24          /**
25           * @uml.property name="fwAdapter"
26           */
27          private FrameworkAdapter fwAdapter;
28          private Script script = null;
29
30
31          /**
32           *          Metodo que inicializa el FrameworkAdapter.
33           *
34           * @throws FrameworkException
35           */
36          private void iniciaAplicacion()throws FrameworkException{
37
38                  String Application_ID = "Servicio de Teleasistencia";
39                  String password = "00010203040506070809";
40
41                  fwAdapter =
    FrameworkAdapterFactory.createFrameworkAdapter(Application_ID, password);
42
43          }
44
45
46          /**
```

```
47          * Metodo que termina todos los contratos de servicio con el framework y libera
48          * los recursos.
49          *
50          * @throws FrameworkException
51          *
52          */
53         private void cleanup() throws FrameworkException {
54
55                 if (script != null){
56
57                         script.destroy();
58                 }
59
60                 if (fwAdapter != null){
61
62                         fwAdapter.endAccess();
63                         fwAdapter.destroy();
64                 }
65
66         }
67
68
69         /**
70          *         Método que lista los servicios disponibles en el ISG.
71          *
72          * @throws FrameworkException
73          */
74         public void listarServicios()throws FrameworkException{
75
76
77                 String lista_servicios[] = fwAdapter.listServices();
78                 if (lista_servicios.length > 0){
79
80                         System.out.println("Se ofrecen los siguientes servicios: ");
81
82                         for (int i=0; i < lista_servicios.length; i++){
83
84                                 System.out.println("    --> " +lista_servicios[i]);
85                         }
86                 }else{
87                         System.out.println("No hay servicios disponibles.");
88                 }
89
90                 System.out.println("");
91         }
92
93
94         /**
95          * Función Main() del servicio.
96          *
97          * @param args
98          */
99         public static void main(String[] args) {
```

```
100		// TODO Auto-generated method stub
101
102		// Inicializo el vector de ambulancias
103		GestionAmbulancias.listaAmbulancias();
104
105		GestionPacientes gp = new GestionPacientes();
106
107		// muestro por pantalla todos los pacientes con algún servicio contratado
108		Vector Pacientes = gp.listaPacientes();
109		gp.imprimeVectorPacientes(Pacientes);
110
111		// preparo la tabla estadopacloc de la BBDD para la localización de pacientes
112		BBDD.reiniciaBBDDloc(Pacientes);
113
114		// no necesito más el vector Pacientes
115		Pacientes.clear();
116		//*************************************************************************
117
118		TeleAsistencia tA = new TeleAsistencia();
119
120		// cargo el fichero XML "teleasistencia" en el simulador, necesario para
121		// realizar la simulacion y ejecuto el programa setup
122		tA.script = new Script();
123		tA.script.cargaScript("teleasistencia.xml");
124		tA.script.ejecutaPrograma("setup", true);
125		//*************************************************************************
126
127		LocalizacionUsuario lu = null;
128		InteraccionUsuarioLoc iu = null;
129		MensajeMultimediaLis MMLis = null;
130
131		try {
132			// creo el FrameworkAdapter
133			tA.iniciaAplicacion();
134			tA.listarServicios();
135
136
137			// se inicia la localización periódica de los pacientes
138			lu = new LocalizacionUsuario(tA.fwAdapter);
139			lu.invocaUL();
140
141
142			// lanzo la monitorización de llamadas
143			iu = new InteraccionUsuarioLoc (tA.fwAdapter);
144			iu.MonitorizaLLamadaLoc();
145
146			// lanzo la monitorización en la recepción de MMS
147			MMLis= new MensajeMultimediaLis(tA.fwAdapter);
148			MMLis.monitorizaMMM();
149
150
151			// lanzo la monitorizacion del estado de los terminales
152			EstadoUsuarioLis euLis = new EstadoUsuarioLis(tA.fwAdapter);
153			euLis.establecerUsuarios();
```

```
154                             euLis.iniciaMonitorizacionEstado();
155
156
157                             //  bucle que comprueba continuamente si el Vector Alarmas está
    vacío;
158                             //  en caso contrario, tendrá que lanzar el tratamiento de alarmas
159
160                             Alarma alarma;
161
162                             while (true){
163
164                                     // primero comprobamos si hay alguna alarma generada
165
166                                     while (!GestionAlarmas.Alarmas.isEmpty()){
167
168                                             alarma = GestionAlarmas.Alarmas.firstElement();
169
170                                             switch (alarma.id_alarma){
171
172                                             case Alarma.CAIDA:
173                                                     System.out.println("Se va a procesar
    alarma por caida.");
174                                                     GestionCaida gc = new
    GestionCaida(tA.fwAdapter,tA.script);
175                                                     gc.procesaAlarmaCaida(alarma);
176                                                     break;
177
178                                             case Alarma.PERSONA_PERDIDA:
179                                                     System.out.println("Se va a procesar
    alarma por persona perdida.");
180                                                     lu.procesaAlarmaLoc(alarma, tA.script,
    tA.fwAdapter);
181                                                     break;
182
183                                             default:
184                                                     System.out.println("No se reconoce el tipo
    de alarma.");
185                                                     GestionAlarmas.eliminaAlarma(alarma);

186
187                                             }
188                                     }
189
190
191                                     synchronized (GestionAlarmas.Alarmas) {
192                                             System.out.println("");
193                                             System.out.println("Bloqueo el hilo principal a la
    espera de alarmas");
194                                             System.out.println("");
195                                             try {
196                                                     GestionAlarmas.Alarmas.wait();
197                                             } catch (InterruptedException e) {
198                                                     // TODO Auto-generated catch block
199                                                     e.printStackTrace();
```

```
200                                                     }
201
202                                             }
203
204                                     }
205
206                             } catch (FrameworkException e1) {
207                                     // TODO Auto-generated catch block
208                                     System.out.println("\n--- FrameworkException atrapada ---\n");
209                                     e1.printStackTrace();
210                                     System.out.println("");
211
212                             }finally{
213
214                                     try {
215
216                                             tA.cleanup();
217                                             lu.cleanup();
218
219                                     } catch (FrameworkException e) {
220                                             // TODO Auto-generated catch block
221                                             System.out.println("\n--- FrameworkException atrapada ---
    \n");
222                                             e.printStackTrace();
223                                             System.out.println("");
224                                     } catch (UserLocationException e) {
225                                             // TODO Auto-generated catch block
226                                             System.out.println("\n--- UserLocationException atrapada ---
    \n");
227                                             e.printStackTrace();
228                                             System.out.println("");
229                                     }
230                             }
231                     }
232
233     }
```

2.2 Paquete BaseDeDatos

2.2.1 BBDD.java

```
1       /*
2        * Proyecto Fin de Carrera: Servicio de Teleasistencia basado en OSA/Parlay
3        *
4        * BBDD.java
5        */
6
7       package BaseDeDatos;
8
9       import java.sql.*;
10      import java.util.Iterator;
11      import java.util.Vector;
12
```

```
13  import TipoDatos.Alarma;
14  import TipoDatos.Paciente;
15
16  /**
17   * Clase que implementa los métodos necesarios para el tratamiento de la base de datos del
18   * servicio.
19   *
20   * @author Raúl Parras Eliche
21   */
22  public class BBDD {
23
24          private static Connection con = null;
25          private static Statement stmt = null;
26          private static ResultSet rs = null;
27
28
29          /**
30           * Método que realiza una consulta en la BBDD de la aplicación.
31           *
32           * @param sentencia_SQL - consulta SQL que queremos realizar.
33           * @return rs - objeto ResultSet con la información consultada en la BBDD.
34           */
35
36          public static ResultSet consulta (String sentencia_SQL){
37
38                  try {
39
40                          Class.forName("com.mysql.jdbc.Driver");
41
42                          } catch (ClassNotFoundException ex) {
43                                  // TODO Auto-generated catch block
44                                  System.out.print("\n--- ClassNotFoundException caught ---\n");
45                                  System.out.println(ex.getMessage());
46                          }
47
48                  try {
49                          //host por defecto: 127.0.0.1
50                          //puerto por defecto: 3306
51                          String url = "jdbc:mysql://localhost/teleasistencia";
52                          con = DriverManager.getConnection(url,"user","pass");
53
54                          stmt = con.createStatement();
55                          rs = stmt.executeQuery(sentencia_SQL);
56
57                          } catch (SQLException ex) {
58                                  // TODO Auto-generated catch block
59                                  System.out.println("\n--- SQLException caught ---\n");
60                                  while (ex != null) {
61                                          System.out.println("Message:   " + ex.getMessage ());
62                                          System.out.println("SQLState:  " + ex.getSQLState ());
63                                          System.out.println("ErrorCode: " + ex.getErrorCode ());
64                                          ex = ex.getNextException();
65                                          System.out.println("");
66                                  }
67                          }
68
69                          return rs;
70          }
71
72          /**
```

```
73          * Método para actualizar datos (actualizar, insertar o borrar) de la BBDD de
74          * la aplicación
75          *
76          * @param sentencia_SQL - orden en SQL a ejecutar.
77          */
78
79          public static void actualiza(String sentencia_SQL){
80
81                  try {
82
83                          Class.forName("com.mysql.jdbc.Driver");
84
85                          } catch (ClassNotFoundException ex) {
86                                  // TODO Auto-generated catch block
87                                  System.out.print("\n--- ClassNotFoundException caught ---\n");
88                                  System.out.println(ex.getMessage());
89                          }
90
91                  try {
92                          //host por defecto: 127.0.0.1
93                          //puerto por defecto: 3306
94                          String url = "jdbc:mysql://localhost/teleasistencia";
95                          con = DriverManager.getConnection(url,"user","pass");
96
97                          stmt = con.createStatement();
98                          stmt.execute(sentencia_SQL);
99
100
101                         } catch (SQLException ex) {
102                                 // TODO Auto-generated catch block
103                                 System.out.println("\n--- SQLException caught ---\n");
104                                 while (ex != null) {
105                                         System.out.println("Message:   " + ex.getMessage ());
106                                         System.out.println("SQLState:  " + ex.getSQLState ());
107                                         System.out.println("ErrorCode: " + ex.getErrorCode ());
108                                         ex = ex.getNextException();
109                                         System.out.println("");
110                                 }
111                         }
112
113         }
114
115
116         /**
117          * Método que prepara la tabla estdopacloc de la Base de Datos antes de
118          * comenzar a comprobar y procesar la posición de los pacientes. Esta tabla
119          * almacena el estado de dichos pacientes (estado relacionado con la
120          * necesidad o no de comprobar sus posiciones en cada instante).
121          *
122          * @param Pacientes - Vector con todos los pacientes que existen en la BBDD
123          *  y que tienen al menos algún servicio contratado.
124          */
125         public static void reiniciaBBDDloc(Vector Pacientes){
126
127                 // borro los datos de la tabla estadopacloc
128                 String sentencia_SQL = "TRUNCATE TABLE estadopacloc";
129                 BBDD.actualiza(sentencia_SQL);
130                 //*************************************************************************
131
132
```

```
133                     // Actualizo la tabla estadopacloc con los estados de los pacientes de
134                     // localización
135                     Iterator it1 = Pacientes.iterator();
136
137                     Paciente pac = null;
138                     Vector serv = null;
139                     Iterator it2 = null;
140                     Integer alarm;
141
142                     while (it1.hasNext()){
143
144                             pac = (Paciente) it1.next();
145                             serv = pac.servicios;
146
147                             it2 = serv.iterator();
148
149                             while (it2.hasNext()){
150                                     alarm = (Integer)it2.next();
151
152                                     if (alarm == Alarma.PERSONA_PERDIDA){
153
154                                             sentencia_SQL = "INSERT INTO estadopacloc " +
155                                                             "(MSISDN,estadoLoc) " +
156                                                             "VALUES ('" + pac.MSISDN + "','"
157                                                             + Paciente.PacNORMAL + "')";
158
159                                             BBDD.actualiza(sentencia_SQL);

160                                     }
161
162                             }
163                     }
164             }
165
166     }
```

2.3 Paquete EstadoTerminal

2.3.1 EstadoUsuario.java

```
1       /*
2        * Proyecto Fin de Carrera: Servicio de Teleasistencia basado en OSA/Parlay
3        *
4        * EstadoUsuario.java
5        */
6
7       package EstadoTerminal;
8
9       import com.lucent.isg.appsdk.userstatus.*;
10      import com.lucent.isg.appsdk.framework.*;
11
12      /**
13       * Clase que implementa los métodos necesarios para averiguar el estado de un terminal.
14       *
```

```
15  * @author Raúl Parras Eliche
16  */
17  public class EstadoUsuario {
18
19          /**
20           * Adaptador de servicio de estado de usuario.
21           */
22          UserStatusAdapter usAdapter;
23
24
25          /**
26           * Construye un objeto EstadoUsuario e inicializa el Adaptador UserStatus mediante
27           * el FrameworkAdapter.
28           *
29           * @param fwAdapter - FrameworkAdapter del servicio de tele asistencia, necesario
30           * para obtener el resto de adaptadores.
31           */
32          public EstadoUsuario(FrameworkAdapter fwAdapter){
33
34                  // Creo el adaptador de servicio
35                  try {
36
37                          usAdapter = fwAdapter.selectUserStatusService();
38
39                          System.out.println("Token de servicio de Estado de Usuario: "
40                                          + usAdapter.getServiceToken() +"\n");
41
42                  } catch (FrameworkException e) {
43                          // TODO Auto-generated catch block
44                          System.out.println("\n--- FrameworkException atrapada ---\n");
45                          e.printStackTrace();
46                          System.out.println("");
47                  }
48          }
49
50          /**
51           * Método que devuelve el estado de un terminal (REACHABLE, NOT REACHABLE O
    BUSY).
52           *
53           * @param MSISDN - Número del terminal del que se quiere averiguar su estado.
54           * @return estadoUsuario - Estado del terminal: REACHABLE, NOT REACHABLE O
    BUSY.
55           */
56          public int estadoTerminal(String MSISDN){
57
58
59                  // por defecto el terminal no está disponible, por si falla la petición que no
60                  // envíe otra cosa cuando no lo sabemos
61                  int estadoUsuario = UserStatus.US_NOT_REACHABLE;
62
63                  try {
64                          UserStatus usStatus = usAdapter.requestStatus(MSISDN);
65                          estadoUsuario = usStatus.getStatus();
66
```

```
67                      } catch (UserStatusException e) {
68                              // TODO Auto-generated catch block
69                              e.printStackTrace();
70                      }
71
72                      return estadoUsuario;
73              }
74
75              /**
76       * Método que elimina el adaptador de servicio UserStatus.
77       */
78      public void cleanup(){
79
80
81        try {
82                      usAdapter.destroy();
83
84              } catch (UserStatusException e) {
85                      // TODO Auto-generated catch block
86                      System.out.println("\n--- UserInteractionException atrapada ---\n");
87                      e.printStackTrace();
88                      System.out.println("");
89              }
90
91      }
92  }
```

2.3.2 EstadoUsuarioLis.java

```
1   /*
2    * Proyecto Fin de Carrera: Servicio de Teleasistencia basado en OSA/Parlay
3    *
4    * EstadoUsuarioLis.java
5    */
6
7   package EstadoTerminal;
8
9   import BaseDeDatos.BBDD;
10  import Gestion.GestionPacientes;
11  import MensajeriaMultimedia.MensajeMultimedia;
12  import TipoDatos.NumerosServicio;
13  import TipoDatos.Paciente;
14
15  import com.lucent.isg.appsdk.framework.FrameworkAdapter;
16  import com.lucent.isg.appsdk.userstatus.*;
17
18  import java.sql.ResultSet;
19  import java.sql.SQLException;
20  import java.util.*;
21
22  /**
23   * Clase que implementa la interfaz UserStatusListener. Esta clase se encarga de
24   * establecer criterios para activar la monitorización en el cambio de estado de
```

```
25   * los terminales de los pacientes y define los métodos que capturan los eventos
26   * producidos como consecuencia de esa monitorización.
27   *
28   * @author Raúl Parras Eliche
29   *
30   */
31  public class EstadoUsuarioLis implements UserStatusListener {
32
33
34          private FrameworkAdapter fwAdapter = null;
35          private EstadoUsuario eu = null;
36          private int assignmentId;
37
38          /**
39           * Array de UserStatus con los MSISDN de los pacientes que se va a comprobar sus
    estados.
40           */
41          private String[] usuarios = null;
42
43
44          /**
45           * Array de UserStatus que contendrá el estado de los pacientes cuando se produzca
46           * un evento.
47           */
48          private UserStatus[] estadoActual = null;
49
50          /**
51           * Construye un objeto UserStatusLis usando el FrameworkAdapter para inicializar
52           * el adaptador UserStatus necesario.
53           *
54           * @param fwAdapter - FrameworkAdapter del servicio de tele asistencia, necesario
55           * para obtener el resto de adaptadores.
56           */
57          public EstadoUsuarioLis(FrameworkAdapter fwAdapter){
58
59                  // obtengo un nuevoAdaptador de servicio usAdapter
60                  this.fwAdapter = fwAdapter;
61                  eu = new EstadoUsuario(fwAdapter);
62
63          }
64
65
66          /**
67           * Método que establece la información acerca de los pacientes para poder iniciar
68           * la monitorización de sus estados.
69           *
70           */
71          public void establecerUsuarios(){
72
73                  // obtengo un vector con todos los pacientes que tienen contratado algún
74                  // servicio, que son a los que voy a chequear el estado de sus terminales
75                  GestionPacientes gp = new GestionPacientes();
76                  Vector pacientes = gp.listaPacientes();
77
```

```
78                      // construyo a partir de ese vector otro vector con los MSISDN de los pacientes
79                      Iterator it = pacientes.iterator();
80                      Paciente p;
81                      Vector<Long> pac= new Vector<Long>();
82
83                      while (it.hasNext()){
84
85                              p = (Paciente)it.next();
86                              pac.addElement(new Long(p.MSISDN));
87
88                      }
89
90                      // elimino el vector de objetos paciente
91                      pacientes.clear();
92
93
94                      // transformo el vector de MSISDN a array de String
95                      usuarios = new String[pac.size()];
96                      it = pac.iterator();
97                      int i =0;
98                      Long l;
99                      while (it.hasNext()){
100                             l = (Long)it.next();
101                             usuarios[i] = l.toString();
102                             i++;
103                     }
104
105                     // elimino el vector pac
106                     pac.clear();
107
108             }
109
110
111             /**
112              * Método que inicia la monitorización del estado de los terminales de
113              * los pacientes.
114              *
115              */
116             public void iniciaMonitorizacionEstado(){
117
118                     try {
119                             assignmentId = eu.usAdapter.requestTriggeredStatus(usuarios, this);
120                     } catch (UserStatusException e) {
121                             // TODO Auto-generated catch block
122                             System.out.println("\n--- UserInteractionException atrapada ---\n");
123                             e.printStackTrace();
124                             System.out.println("");
125                     }
126
127             }
128
129             /**
130              * Método utilizado para parar la monitorización del estado de los terminales
131              * de los pacientes.
```

```
132             */
133             public void pararMonitorizacionEstado(){
134
135                     try {
136                             eu.usAdapter.stopTriggeredStatus(assignmentId);
137                     } catch (UserStatusException e) {
138                             // TODO Auto-generated catch block
139                             System.out.println("\n--- UserInteractionException atrapada ---\n");
140                             e.printStackTrace();
141                             System.out.println("");
142                     }
143
144             }
145
146
147             /**
148              * Método llamado desde onEvent cuando el terminal de un paciente cambia de
149              * estado, realizando las acciones pertinentes.
150              *
151              */
152             private void procesaEstado(){
153
154                     UserStatus estAct = null;
155                     String sentencia_SQL;
156                     Long MSISDNcuidador = null;
157                     MensajeMultimedia mm = null;
158
159                     for (int p = 0; p < estadoActual.length; p++){
160
161                             estAct = estadoActual[p];
162
163                             switch (estAct.getStatus()){
164
165                             case UserStatus.US_NOT_REACHABLE:
166                                     // si el evento es el terminal apagado, es porque antes tenía
    que
167                                     // estar REACHABLE
168
169                                     try {
170                                             // busco el MSISDN de su cuidador
171                                             sentencia_SQL = "SELECT id_cuidador FROM
    pacientes " +
172                                                             "WHERE MSISDN='" +
    estAct.getUser() +"'";
173
174                                             ResultSet rs = BBDD.consulta(sentencia_SQL);
175                                             rs.next();
176
177                                             sentencia_SQL = "SELECT MSISDN FROM
    cuidadores "
178                                                     + "WHERE id_cuidador='" + rs.getInt(1)
    +"'";
179
180                                             rs = BBDD.consulta(sentencia_SQL);
```

```
181                                         rs.next();
182
183                                         MSISDNcuidador = rs.getLong(1);
184
185                                 } catch (SQLException e) {
186                                         // TODO Auto-generated catch block
187                                         System.out.println("\n--- SQLException atrapada ---
    \n");
188                                         e.printStackTrace();
189                                         System.out.println("");
190                                 }
191
192                                 // se prodede a enviar un MMS al cuidador del paciente
193                                 mm = new MensajeMultimedia(fwAdapter);
194                                 mm.establecerDirecciones(new
    String[]{MSISDNcuidador.toString()},
195                                                 NumerosServicio.MSISDNcentral);
196                                 // se le pasa como parámetro el archivo de texto que contiene
    la
197                                 // información a enviar y el MSISDN del paciente que ha
    generado
198                                 // el evento
199                                 mm.enviarMensaje("avisoTerminalOff.txt",estAct.getUser());
200                                 mm.cleanup();
201
202                                 // cambio el estado del paciente a atendido
203                                 sentencia_SQL = "UPDATE estadopacloc SET estadoLoc='"
204                                         +Paciente.PacATENDIDO +"' WHERE MSISDN='"
    +estAct.getUser() +"'";
205                                 BBDD.actualiza(sentencia_SQL);

206
207                                 break;
208
209                         case UserStatus.US_REACHABLE:
210
211                                 try {
212                                         // busco el MSISDN de su cuidador
213                                         sentencia_SQL = "SELECT id_cuidador FROM
    pacientes " +
214                                                         "WHERE MSISDN='" +
    estAct.getUser() +"'";
215
216                                         ResultSet rs = BBDD.consulta(sentencia_SQL);
217                                         rs.next();
218
219                                         sentencia_SQL = "SELECT MSISDN FROM
    cuidadores "
220                                                 + "WHERE id_cuidador='"     + rs.getInt(1)
    +"'";
221
222                                         rs = BBDD.consulta(sentencia_SQL);
223                                         rs.next();
224
```

```
225                                             MSISDNcuidador = rs.getLong(1);
226
227                                     } catch (SQLException e) {
228                                             // TODO Auto-generated catch block
229                                             System.out.println("\n--- SQLException atrapada ---
    \n");
230                                             e.printStackTrace();
231                                             System.out.println("");
232                                     }
233
234                                     // se prodede a enviar un SMS al cuidador del paciente
235                                     mm = new MensajeMultimedia(fwAdapter);
236                                     mm.establecerDirecciones(new
    String[]{MSISDNcuidador.toString()},
237                                                     NumerosServicio.MSISDNcentral);
238                                     // se le pasa como parámetro el archivo de texto que contiene
    la
239                                     // información a enviar y el MSISDN del paciente que ha
    generado
240                                     // el evento
241                                     mm.enviarMensaje("avisoTerminalOn.txt",estAct.getUser());
242                                     mm.cleanup();
243
244                                     // cambio el estado del paciente a normal
245                                     sentencia_SQL = "UPDATE estadopacloc SET estadoLoc='"
246                                             + Paciente.PacNORMAL + "' WHERE MSISDN='"
247                                             + estAct.getUser() + "'";
248                                     BBDD.actualiza(sentencia_SQL);
249
250                                     break;
251
252                             default:
253                                     System.out.println("Estado no reconocido: " +
    estAct.getUser() + ".");
254
255                             }
256
257                     }
258
259             }
260
261     //================================================
262     // Implementacion de la interfaz UserStatusListener
263     //================================================
264
265             /**
266              * Método llamado por las Convenience Classes cuando se produce alguno de los
267              * eventos programados por la aplicación.
268              *
269              * @param usEvent - El evento UserStatus que se acaba de producir.
270              */
271             public void onEvent(UserStatusEvent usEvent) {
272                     // TODO Auto-generated method stub
273
```

```
274                     System.out.println("En método onEvent() del Listener de Estado de Usuario.");
275
276                     estadoActual = usEvent.getUserStatusList();
277
278                     procesaEstado();
279             }
280
281             /**
282              * No se usa.
283              */
284             public void onError(UserStatusError usError) {
285                     // TODO Auto-generated method stub
286
287             }
288     }
```

2.4 Paquete Localizacion

2.4.1 LocalizacionUsuario.java

```
1       /*
2        * Proyecto Fin de Carrera: Servicio de Teleasistencia basado en OSA/Parlay
3        *
4        * LocalizacionUsuario.java
5        */
6
7       package Localizacion;
8
9       import java.sql.ResultSet;
10      import java.sql.SQLException;
11
12      import BaseDeDatos.BBDD;
13      import Gestion.GestionAlarmas;
14      import Gestion.GestionAmbulancias;
15      import Gestion.Script;
16      import MensajeriaMultimedia.MensajeMultimedia;
17      import TipoDatos.Alarma;
18      import TipoDatos.NumerosServicio;
19      import TipoDatos.Paciente;
20
21      import com.lucent.isg.appsdk.framework.FrameworkAdapter;
22      import com.lucent.isg.appsdk.framework.FrameworkException;
23      import com.lucent.isg.appsdk.userlocation.UserLocation;
24      import com.lucent.isg.appsdk.userlocation.UserLocationAdapter;
25      import com.lucent.isg.appsdk.userlocation.UserLocationError;
26      import com.lucent.isg.appsdk.userlocation.UserLocationException;
27
28      /**
29       * Clase que engloba todo lo relacionado con el servicio de localización de los
30       * pacientes. Crea un hilo que es el encargado de comprobar sus posiciones y define
31       * métodos para tratar las alarmas correspondientes así como la respuestas de los
```

```
32      * cuidadores.
33      *
34      * @author Raúl Parras Eliche
35      *
36      */
37     public class LocalizacionUsuario {
38
39             /**
40              * Tiempo que dormimos el hilo entre comprobación y comprobación de la posición
41              * de los pacientes
42              */
43             public static final int TIEMPO_DORMIDO_LOC = 30;
44
45             /**
46              * Adaptador del servicio de localización.
47              */
48             private UserLocationAdapter ulAdapter = null;
49
50             /**
51              * Variable que sirve para detener la ejución del hilo hasta que la respuesta del
52              * cuidador asociado al paciente que ha generado la alarma está lista.
53              */
54             public static boolean stopLoc = true;
55
56             /**
57              * Variable donde se almacena la respuesta del cuidador asociado a un paciente
58              * que ha generado una alarma de localización.
59              */
60             public static int RespCuidLoc;
61
62             /**
63              * Variable que almacena el MSISDN del cuidador asociado al paciente cuya
64              * alarma se está tratando.
65              */
66             public static long MSISDNcuidAlarma;
67
68
69             /**
70              * Constructor de la clase LocalizacionUsuario. Inicia el adaptador de servicio para
71              * la localización de los usuarios.
72              *
73              * @param fwAdapter - FrameworkAdapter del servicio de tele asistencia, necesario
74              * para obtener el resto de adaptadores.
75              */
76             public LocalizacionUsuario(FrameworkAdapter fwAdapter){
77
78                     try {
79                             ulAdapter = fwAdapter.selectUserLocationService();
80
81                     } catch (FrameworkException e) {
82                             // TODO Auto-generated catch block
83                             e.printStackTrace();
84                     }
85
```

```
86                  System.out.println("Token del Servicio de Localización: "
87                                  + ulAdapter.getServiceToken());
88                  System.out.println("");
89
90          }
91
92
93          /**
94           * Método que termina el adaptador de servicio para localización.
95           *
96           * @throws UserLocationException
97           * de servicio.
98           */
99          public void cleanup() throws UserLocationException {
100
101                 if (ulAdapter != null){
102
103                         ulAdapter.destroy();
104                 }
105
106         }
107
108         /**
109          * Método que imprime por pantalla un array de localizaciones.
110          *
111          * @param locations - Array de posiciones que se va a imprimir.
112          */
113         public static void imprimeLocalizacion(UserLocation[] locations){
114
115                 for (int i= 0; i<locations.length; i++){
116
117                         UserLocation loc = locations[i];
118                         System.out.print("userid: " +loc.getUser());
119
120                         UserLocationError error =loc.getError();
121                         if (error!=null){
122                                 System.out.println(", error: " +error.toString());
123                         }else{
124                                 System.out.print(", longitude: " +loc.getLongitude());
125                                 System.out.println(", latitude: " +loc.getLatitude());
126                         }
127                 }
128         }
129
130
131         /**
132          * Método que inicia el servicio de localización. Para ello crea un  hilo específico,
133          * que será el encargado de comprobar periódicamente las posiciones de los
134          * pacientes.
135          *
136          */
137         public void invocaUL() {
138
139                 HiloLoc hiloLoc = new HiloLoc(ulAdapter);
```

```
140                     hiloLoc.start();
141             }
142
143
144             /**
145              * Método que procesa las posiciones de los pacientes buscando algún tipo de anomalía.
146              * Si la encuentra, crea una nueva alarma.
147              *
148              * @param locations - Array de posiciones que se comprobarán comparándolos con los
149              * de referencia que se possen de los pacientes.
150              *
151              * @throws SQLException
152              */
153             public static void procesaPosiciones(UserLocation[] locations) throws SQLException{
154
155                     System.out.println("");
156
157                     String sentencia_SQL;
158
159                     // calculamos la distancia ahora
160
161                     for (int i= 0; i < locations.length; i++){
162
163                             int alarma;
164                             long MSISDN;
165                             double distancia;
166                             int estadoPac;
167
168                             ResultSet rs;
169
170                             UserLocation loc_ref;
171                             locations[i].getError();
172
173                             System.out.println("");
174                             System.out.println("userid: " +locations[i].getUser());
175
176                             if (locations[i].getError()!=null){
177
178                                     System.out.println(", error: " +
    locations[i].getError().toString());
179                             }else{
180                                     //  obtengo la latitud y la longitud de referencia de cada
    paciente
181                                     //  utilizo MSISDN para buscarlas porque antes ya he
    asociado
182                                     //  los MSISDN a los dni
183                                     sentencia_SQL = "SELECT longitud,latitud FROM pacientes "
    +
184                                                     "WHERE MSISDN='" +
    locations[i].getUser() +"'";
185                                     rs = BBDD.consulta(sentencia_SQL);
186
187                                     rs.next();
188                                     loc_ref = new UserLocation
```

```
(rs.getDouble(1),rs.getDouble(2));
189
190                                     //  calculo la distancia entre ambas posiciones
191                                     distancia = locations[i].distanceTo(loc_ref);
192
193                                     System.out.println("Distancia del paciente = "
194                                                     + locations[i].getUser() +
195                                                     " con respecto a su posicion de
referencia: "
196                                                     + distancia +" km.");
197
198                                     //  supongo un radio de seguridad de 1 km
199                                     if (distancia < 1){
200
201                                             System.out.println("No hay necesidad de generar
una " +
202                                                             "alarma para este " +
"paciente.");
203                                             System.out.println("");
204
205                                     }else{
206
207                                             alarma = Alarma.PERSONA_PERDIDA;
208                                             MSISDN =
Long.valueOf(locations[i].getUser()).longValue();
209
210                                             sentencia_SQL = "SELECT estadoLoc FROM
estadopacloc " +
211                                                             "WHERE MSISDN='" + MSISDN
+ "'";
212                                             rs = BBDD.consulta(sentencia_SQL);
213                                             rs.next();
214                                             estadoPac = rs.getInt(1);
215
216                                             // comprobamos primero que la alarma no se haya
generado ya y que el
217                                             // estado del paciente sea normal (no atendido)
antes de crear la
218                                             // alarma
219                                             if
((!GestionAlarmas.compruebaAlarmaCreada(alarma, MSISDN)) &&
220                                                             (estadoPac ==
Paciente.PacNORMAL)){
221
222                                                     System.out.println("El paciente ha
excedido el radio de seguridad de 1 km."
223                                                             + "Se procederá a la
creación de la alarma correspondiente.");
224                                                     System.out.println("");
225
226                                                     Alarma a = new Alarma(alarma,
MSISDN);
227
228                                                     GestionAlarmas.anadeAlarma(a);
```

```
229
230                                                     }
231
232                                                 }
233                                             }
234                                         }
235                 }
236
237                 /**
238                  * Método encargado de procesar las alarmas generadas por una posición incorrecta
239                  * del paciente.
240                  *
241                  * @param alarma - Objeto alarma que se va a tratar.
242                  * @param script - Clase para cargar y ejecutar programas en el simulador.
243                  * @param fwAdapter - FrameworkAdapter del servicio de tele asistencia, necesario
244                  * para obtener el resto de adaptadores.
245                  */
246                 public void procesaAlarmaLoc(Alarma alarma, Script script,
247                                     FrameworkAdapter fwAdapter){
248
249                         System.out.print ("Procesando alarma.....ORIGEN: ");
250                         System.out.print("usuario " + alarma.MSISDN +", tipo " +alarma.id_alarma);
251                         System.out.println(".....");
252
253                         // primero vamos a comprobar cúal es la posición del cuidador que
254                         // esté a cargo del paciente que ha generado la alarma;
255                         // es decir, si están juntos (distancia entre ambos menor a 20
256                         // metros), no hay ningúm problema.
257                         // Para ello, necesito hallar primero al cuidador y después su MSISDN
258
259                         String sentencia_SQL = "SELECT id_cuidador FROM pacientes WHERE
    MSISDN='"
260                                     + alarma.MSISDN +"'";
261
262
263                         try {
264
265                                     ResultSet rs = BBDD.consulta(sentencia_SQL);
266                                     rs.next();
267
268                                     sentencia_SQL = "SELECT MSISDN,nombre,apellido1,apellido2
    FROM " +
269                                                 "cuidadores " +     "WHERE id_cuidador='" +
    rs.getInt(1) +"'";
270                                     rs = BBDD.consulta(sentencia_SQL);
271                                     rs.next();
272
273                                     MSISDNcuidAlarma = rs.getLong(1);
274                                     Long MSISDNpaciente = alarma.MSISDN;
275
276                                     System.out.println("Los datos del cuidador asociado al paciente son: ");
277                                     System.out.println("    MSISDN -->> " + MSISDNcuidAlarma);
278                                     System.out.println("    Nombre -->> " + rs.getString(2));
279                                     System.out.println("    Apellido 1 -->> " + rs.getString(3));
```

```
280                             System.out.println("    Apellido 2 -->> " + rs.getString(4));
281
282
283                             // calculo ahora la posicion del cuidador y la del paciente, para poder
284                             // calcular la distancia entre ambos
285                             Long MSISDNcuidador = MSISDNcuidAlarma;
286                             String[] ULcuidador =
    {MSISDNcuidador.toString(),MSISDNpaciente.toString()};
287                             UserLocation[] LOCcuid;
288
289
290                             LOCcuid = ulAdapter.requestLocation(ULcuidador);
291                             LocalizacionUsuario.imprimeLocalizacion(LOCcuid);
292
293                             // la distancia entre ambos es una cantidad en km
294                             double distanciaKM = LOCcuid[0].distanceTo(LOCcuid[1]);
295                             long distanciaM = (long)(distanciaKM * 1000);
296                             System.out.println("Distancia entre paciente y cuidador: "
297                                     + distanciaM +" m.");
298
299                             if (distanciaM <= 20){
300
301                                     System.out.println("Suponemos que el paciente se " +
302                                             "encuentra con su cuidador.");
303                                     System.out.println("Se le enviará un MMS avisándole.");
304
305                                     MensajeMultimedia mm = new
    MensajeMultimedia(fwAdapter);
306                                     mm.establecerDirecciones(new
    String[]{MSISDNcuidador.toString()},
307                                             NumerosServicio.MSISDNcentral);
308                                     // se le pasa como parámetro el archivo de texto que contiene
    la
309                                     // información a enviar
310                                     mm.enviarMensaje("avisoAlarma.txt",
    MSISDNpaciente.toString());
311                                     mm.cleanup();
312
313                                     // cambio el estado del paciente a atendido
314                                     sentencia_SQL = "UPDATE estadopacloc SET estadoLoc='"
315                                             + Paciente.PacATENDIDO + "' WHERE MSISDN='"
    + alarma.MSISDN + "'";
316                                     BBDD.actualiza(sentencia_SQL);
317
318
319                             }else{
320
321                                     String programa = "LLamada_Central_" + MSISDNcuidador;
322                                     int contLLamadas = 1;
323                                     script.ejecutaPrograma(programa, true);
324
325                                     procesaRespCuid(alarma, ulAdapter, script, LOCcuid[1],
    contLLamadas,
326                                             MSISDNcuidador, fwAdapter);
```

```
327
328                             }
329
330                     } catch (UserLocationException e1) {
331                             // TODO Auto-generated catch block
332                             System.out.println("\n--- UserLocationException caught ---\n");
333                             e1.printStackTrace();
334
335                     } catch (SQLException e) {
336                             // TODO Auto-generated catch block
337                             e.printStackTrace();
338                     }
339
340                     // eliminamos la alarma
341                     GestionAlarmas.eliminaAlarma(alarma);
342
343             }
344
345
346             /**
347              * Método que procesa la respuesta del cuidador encargado del paciente que ha
348              * generado la alarma por localización.
349              *
350              * @param alarma - Objeto alarma que se está tratando.
351              * @param ulAdapter - Adaptador del servicio de localización.
352              * @param script - Clase para cargar y ejecutar programas en el simulador.
353              * @param locPac - Posición del paciente que ha causado la alarma.
354              * @param contLLamadas - Intentos de llamadas al cuidador sin éxito.
355              * @param MSISDNcuidador - Número del terminal del cuidador asociado al paciente que
356              * ha generado la alarma.
357              * @param fwAdapter - FrameworkAdapter del servicio de tele asistencia, necesario
358              * para obtener el resto de adaptadores.
359              */
360             private static void procesaRespCuid(Alarma alarma, UserLocationAdapter ulAdapter,
361                             Script script, UserLocation locPac, int contLLamadas, Long
    MSISDNcuidador,
362                             FrameworkAdapter fwAdapter){
363
364                     String sentencia_SQL = null;
365                     String programa = null;
366                     String MSISDNamb = null;
367
368                     // bucle que nos sirve para dormir el hilo hasta que tengamos lista la respuesta
369                     // del cuidador
370                     while (stopLoc){
371
372                             try {
373                                     Thread.sleep(5000);
374                             } catch (InterruptedException e) {
375                                     // TODO Auto-generated catch block
376                                     e.printStackTrace();
377                             }
378                     }
379                     stopLoc = true;
```

```
380
381                     // compruebo cual ha sido la respuesta del cuidador y realizo las
382                     // acciones correspondientes
383
384                     switch(RespCuidLoc){
385
386                     case GestionAlarmas.ATENDIDO:  // cambio el estado del paciente
387
388                             sentencia_SQL = "UPDATE estadopacloc SET estadoLoc='"
389                                     + Paciente.PacATENDIDO + "' WHERE MSISDN='" +
    alarma.MSISDN + "'";
390                             BBDD.actualiza(sentencia_SQL);
391
392                             break;
393
394                     case GestionAlarmas.DESCONOCIDO:
395                             // llamo a la ambulancia más cercana al paciente
396
397                             MSISDNamb = GestionAmbulancias.ambulanciaMasCercana(locPac,
398                                             ulAdapter,fwAdapter);
399
400                             if (MSISDNamb != null){
401
402                                     programa = "LLamada_Central_" + MSISDNamb;
403                                     script.ejecutaPrograma(programa, false);
404
405                                     // y modifico el estado del paciente para que no siga
    generando la
406                                     // misma alarma
407                                     sentencia_SQL = "UPDATE estadopacloc SET estadoLoc='"
408                                             + Paciente.PacATENDIDO + "' WHERE MSISDN='"
    + alarma.MSISDN + "'";
409                                     BBDD.actualiza(sentencia_SQL);
410                             }
411
412                             break;
413
414                     default:
415                             // se engloban aquí los casos de error de marcado del cuidador
416                             // y de expiración del tiempo
417                             if (contLLamadas < 2){
418
419                                     programa = "LLamada_Central_" + MSISDNcuidador;
420                                     contLLamadas ++;
421                                     script.ejecutaPrograma(programa, true);
422                                     procesaRespCuid(alarma, ulAdapter, script, locPac,
423                                                     contLLamadas,
    MSISDNcuidador,fwAdapter);
424
425                             } else {
426                                     // si ya hemos llamado dos veces al cuidador sin éxito,
    llamamos a la
427                                     // ambulancia más cercana
428                                     System.out.println("Se ha llamado al cuidador 2 veces sin
```

```
éxito.");
429                                     System.out.println("Se procederá a llamar a la ambulancia
más cercana.");
430                                     MSISDNamb =
GestionAmbulancias.ambulanciaMasCercana(locPac,
431                                                     ulAdapter,fwAdapter);
432
433                                     if (MSISDNamb != null){
434
435                                             programa = "LLamada_Central_" + MSISDNamb;
436                                             script.ejecutaPrograma(programa, false);
437
438                                             // y modificamos el estado del paciente para que no
siga
439                                             // generando la misma alarma
440                                             sentencia_SQL = "UPDATE estadopacloc SET
estadoLoc='"
441                                                     + Paciente.PacATENDIDO + "' WHERE
MSISDN='"
442                                                     + alarma.MSISDN + "'";
443                                             BBDD.actualiza(sentencia_SQL);
444
445                                     }
446                             }
447                     }
448             }
449
450 }
```

2.4.2 HiloLoc.java

```
1   /*
2    * Proyecto Fin de Carrera: Servicio de Teleasistencia basado en OSA/Parlay
3    *
4    * HiloLoc.java
5    */
6
7   package Localizacion;
8
9   import java.sql.SQLException;
10
11  import Gestion.GestionPacientes;
12  import TipoDatos.Alarma;
13
14  import com.lucent.isg.appsdk.userlocation.UserLocation;
15  import com.lucent.isg.appsdk.userlocation.UserLocationAdapter;
16  import com.lucent.isg.appsdk.userlocation.UserLocationException;
17
18  /**
19   * Clase que implementa un nuevo hilo para el servicio de localización.
20   * Se encargará de comprobar periódicamente las posiciones de los pacientes.
21   *
22   * @author Raúl Parras Eliche
```

```
23   */
24   public class HiloLoc extends Thread {
25
26           /**
27            * Variable para poder parar el hilo (con valor false).
28            */
29           private boolean continuar = true;
30
31           /**
32            * Adaptador del servicio de localización.
33            */
34           private UserLocationAdapter ulAdapter;
35
36           /**
37            * Método para detener este hilo.
38            */
39           public void detenerHilo(){
40
41                   continuar = false;
42           }
43
44
45           /**
46            * Construye un objeto de la clase HiloLoc. Realiza una copia del adaptador
47            * UserLocation que recibe como parámetro.
48            *
49            * @param ulAdapter - Adaptador del servicio de localización.
50            */
51           public HiloLoc (UserLocationAdapter ulAdapter){
52
53                   this.ulAdapter = ulAdapter;
54
55           }
56
57
58           /**
59            * Definimos el método run() similar al main() pero para Hilos.
60            *
61            */
62           public void run(){
63
64                   // creo un array de string con todos los MSISDN de los pacientes que tienen
65                   // este servicio contratado
66
67                   GestionPacientes g_p = new GestionPacientes();
68                   String[] UL_abonados = g_p.listado_MSISDN(Alarma.PERSONA_PERDIDA);
69
70                   // obtengo la posicion de los abonados
71                   UserLocation[] locations;
72                   try {
73
74                           locations = ulAdapter.requestLocation(UL_abonados);
75                           LocalizacionUsuario.imprimeLocalizacion(locations);
76
```

```
77                      } catch (UserLocationException e1) {
78                              // TODO Auto-generated catch block
79                              System.out.println("\n--- UserLocationException caught ---\n");
80                              e1.printStackTrace();
81                      }
82
83                      System.out.println("");
84
85                      while(continuar){
86
87                              try {
88
89                                      sleep(LocalizacionUsuario.TIEMPO_DORMIDO_LOC *
    1000);
90
91                              } catch (InterruptedException e) {
92                                      // TODO Auto-generated catch block
93                                      System.out.println("\n--- InterruptedException caught ---\n");
94                                      e.printStackTrace();
95                              }
96
97                              //  obtenemos las nuevas posiciones de los pacientes
98
99                              try {
100
101                                     locations = ulAdapter.requestLocation(UL_abonados);
102                                     LocalizacionUsuario.procesaPosiciones(locations);
103
104                             } catch (UserLocationException e) {
105                                     // TODO Auto-generated catch block
106                                     System.out.println("\n--- UserLocationException caught ---\n");
107                                     e.printStackTrace();
108
109                             } catch (SQLException e) {
110                                     // TODO Auto-generated catch block
111                                     System.out.println("\n--- SQLException caught ---\n");
112                                     while (e != null) {
113                                             System.out.println("Message:   " + e.getMessage
    ());
114                                             System.out.println("SQLState:  " + e.getSQLState
    ());
115                                             System.out.println("ErrorCode: " + e.getErrorCode
    ());
116                                             e = e.getNextException();
117                                             System.out.println("");
118                                     }
119
120                             }
121
122                     }
123
124             }
125     }
```

2.5 Paquete InteraccionUsuario

2.5.1 InteraccionUsuarioLoc.java

```
1   /*
2    * Proyecto Fin de Carrera: Servicio de Teleasistencia basado en OSA/Parlay
3    *
4    * InteraccionUsuarioLoc.java
5    */
6
7   package InteraccionUsuario;
8
9   import Gestion.GestionAlarmas;
10  import Localizacion.LocalizacionUsuario;
11  import TipoDatos.NumerosServicio;
12
13  import com.lucent.isg.appsdk.callcontrol.*;
14  import com.lucent.isg.appsdk.framework.*;
15  import com.lucent.isg.appsdk.userinteraction.*;
16
17  /**
18   * Clase que engloba todo lo relacionado con la captura de las llamadas, pues
19   * implementa la interfaz CallControlListener, y establece la interacción con
20   * los usuarios. Establece los criterios de captura y define métodos para tratar
21   * la información asociada ha dicha interacción.
22   *
23   * @author Raúl Parras Eliche
24   *
25   */
26
27  public class InteraccionUsuarioLoc implements CallControlListener {
28
29          private CallControlAdapter ccAdapter = null;
30          private UserInteractionAdapter uiAdapter = null;
31
32
33          // objeto de interaccion de usuario de llamada
34          private UICall uiCall = null;
35
36          // parámetros de la monitorización de las llamadas
37          // las direcciones origen y destino están cambiadas porque el simulador sólo permite
38          // que la interaccion con el usuario se produzca en el sentido del llamante, así que
39          // para que el cuidador oiga el menú cuando se produzca la alarma, tengo que forzar a
40          // que sea él el que llame a la central, en vez de la central a él, como se haría en
41          // la realidad
42
43          private String direccionOrigen = "*";
44          private String direccionDestino = NumerosServicio.MSISDNcentral;
45          private int mascara = Event.ADDRESS_ANALYSED;
46          private boolean originatorEvents = false;
47          private boolean modoInterrupcion = true;
48
49          // identificadores de los mensajes a enviar (se corresponden con los del script
```

```
50          // teleasistencia.xml)
51          private static final int ID_MENU = 1;
52          private static final int ID_GRACIAS = 2;
53          private static final int ID_INCORRECTO = 3;
54          //private static final int ID_TARDE = 4;
55
56          // valor devuelto al habilitar la notificación de llamadas
57          private int assignmentId;
58
59
60          /**
61           * Constructor de la clase InteraccionUsuario. Inicializa los adaptadores para los
62           * servicios de control de llamada e interacción con el usuario.
63           *
64           * @param fwAdapter - FrameworkAdapter del servicio de tele asistencia, necesario
65           * para obtener el resto de adaptadores.
66           */
67          public InteraccionUsuarioLoc(FrameworkAdapter fwAdapter){
68
69                  // Creo los adaptadores de servicio
70                  try {
71
72                          ccAdapter = fwAdapter.selectCallControlService();
73                          uiAdapter = fwAdapter.selectUserInteractionService();
74                          System.out.println("Token de servicio de Interaccion de Usuario: "
75                                          + uiAdapter.getServiceToken() +"\n");
76
77                  } catch (FrameworkException e) {
78                          // TODO Auto-generated catch block
79                          e.printStackTrace();
80                  }
81          }
82
83          /**
84           * Método que establece los criterios para la escucha de llamadas y lanza la
85           * monitorización de las mismas.
86           *
87           */
88          public void MonitorizaLLamadaLoc(){
89
90
91                  // creo el objeto call criteria necesario para la monitorizacion de las llamadas
92                  CallCriteria criterio = new CallCriteria(direccionOrigen,direccionDestino,
93                                  mascara,originatorEvents,modoInterrupcion);
94
95                  try {
96                          assignmentId = ccAdapter.enableCallNotification(criterio, this);
97                  } catch (CallControlException e) {
98                          // TODO Auto-generated catch block
99                          e.printStackTrace();
100                 }
101
102         }
103
```

```
104
105             /**
106              * Método que envía a través de la llamada que ha sido capturada un menú para
107              * que el cuidador responda.
108              *
109              * @param call - Objeto llamada capturado.
110              */
111             private void enviarMenuAlarmaLoc(Call call){
112
113
114                     // obtengo el objeto de interaccion de usuario de la llamada
115                     try {
116                             uiCall = uiAdapter.createUICall(call);
117                     } catch (UserInteractionException e1) {
118                             // TODO Auto-generated catch block
119                             e1.printStackTrace();
120                     }
121
122                     // especifico el mensaje a enviar
123                     SendInfoCriteria sic = new SendInfoCriteria(ID_MENU);
124
125                     // espero un único número de respuesta (valor de tiempo de 5 segundos)
126                     CollectCriteria cc = new CollectCriteria(1,1,null,5000,3000);
127
128                     // se reproduce el menú y se espera la respuesta del usuario
129                     UIReport uiReport = null;
130
131                     try {
132
133                             uiReport = uiCall.sendInfoAndCollect(sic, cc);
134
135
136                         } catch (UserInteractionException e1) {
137                             // TODO Auto-generated catch block
138                             System.out.println("UserInteractionException atrapada.");
139                             System.out.println("Error: " + e1.getReportError());
140
141                             uiReport = new UIReport (UIReport.REPORT_TIMEOUT);
142
143                         } finally{
144
145                             System.out.println("Menú enviado. Respuesta: " +uiReport);
146
147                             // tratamiento de la respuesta del cuidador
148                             tratarRespCuid(uiReport);
149
150                         }
151
152                     // termino la interaccion con el usuario en esta llamada y termino
153                     // la llamada
154
155                     try {
156
157                             uiCall.release();
```

```
158                                 call.release();
159
160                         } catch (UserInteractionException e1) {
161                                 // TODO Auto-generated catch block
162                                 e1.printStackTrace();
163                         } catch (CallControlException e) {
164                                 // TODO Auto-generated catch block
165                                 e.printStackTrace();
166                         }
167         }
168
169
170         /**
171          * Método que trata la respuesta del cuidador.
172          *
173          * @param respuesta - Respuesta obtenida de la interacción con el usuario.
174          */
175         private void tratarRespCuid(UIReport respuesta){
176
177                 SendInfoCriteria sic = null;
178
179                 // si se entra en el siguiente if es que el cuidador ha introducido algún dato
180                 if (respuesta.getReportType() == UIReport.REPORT_INFO_COLLECTED){
181
182                         // resupero el digito introducido por el cuidador
183                         String digito = respuesta.getCollectedInfo();
184
185                         if(digito != null && digito.equals("1")){
186
187                                 LocalizacionUsuario.RespCuidLoc =
    GestionAlarmas.ATENDIDO;
188
189                                 sic = new SendInfoCriteria(ID_GRACIAS);
190
191                         } else if( digito != null && digito.equals("2")){
192
193                                 LocalizacionUsuario.RespCuidLoc =
    GestionAlarmas.DESCONOCIDO;
194
195                                 sic = new SendInfoCriteria(ID_GRACIAS);
196
197                         } else {
198
199                                 System.out.println("La información introducida por el cuidador
    no " +
200                                                 "es correcta.");
201                                 LocalizacionUsuario.RespCuidLoc =
    GestionAlarmas.ERROR;
202
203                                 sic = new SendInfoCriteria (ID_INCORRECTO);
204                         }
205
206                 // si se entra aquí es que el tiempo del cuidador para introducir datos expiró
207                 } else if(respuesta.getReportType() == UIReport.REPORT_TIMEOUT){
```

```
208
209                                 System.out.println("El cuidador no ha pulsado nada o tardado
    demasiado.");
210                                 LocalizacionUsuario.RespCuidLoc = GestionAlarmas.TIMEOUT;
211
212
213
214                         } else {
215
216                                 System.out.println("UIReport no válido, tipo: " +
    respuesta.getReportType());
217                         }
218
219                         if (sic != null){
220
221                                 try {
222                                         uiCall.sendInfo(sic);
223
224                                 } catch (UserInteractionException e) {
225                                         // TODO Auto-generated catch block
226                                         e.printStackTrace();
227                                 }
228                         }
229
230                         LocalizacionUsuario.stopLoc = false;
231
232                 }
233
234                 /**
235                  * Método que elimina los adaptadores de servicio.
236                  */
237                 public void cleanup(){
238
239                     try {
240
241                         if (uiAdapter != null){
242
243                                 uiAdapter.destroy();
244                         }
245
246                         if (ccAdapter != null){
247
248                                 ccAdapter.destroy();
249                         }
250
251                         } catch (UserInteractionException e) {
252                                 // TODO Auto-generated catch block
253                                 System.out.println("\n--- UserInteractionException atrapada ---\n");
254                                 e.printStackTrace();
255                                 System.out.println("");
256                         } catch (CallControlException e) {
257                                 // TODO Auto-generated catch block
258                                 System.out.println("\n--- CallControlException atrapada ---\n");
259                                 e.printStackTrace();
```

```
260                     System.out.println("");
261             }
262
263
264     }
265
266     /**
267      * Método que deshabilita la monitorización de llamadas.
268      *
269      * @return resultado
270      */
271     public boolean paraMonitorizacionLLamada(){
272
273             boolean resultado = false;
274
275             try {
276                             resultado = ccAdapter.disableCallNotification(assignmentId);
277                     } catch (CallControlException e) {
278                             // TODO Auto-generated catch block
279                             e.printStackTrace();
280                     }
281
282                     return resultado;
283     }
284
285
286     //================================================
287     // Implementacion de la interfaz CallControlListener
288     //================================================
289
290             /**
291              * Método llamado por las Convenience Classes cuando se produce alguno de los
292              * eventos programados por la aplicación.
293              *
294              * @param ccEvent - El evento CallCOntrol que se acaba de producir.
295              */
296             public void onEvent(CallControlEvent ccEvent) {
297                     // TODO Auto-generated method stub
298
299                     Call call = ccEvent.getCall();
300                     String MSISDNcuidador = call.getOriginatorAddress();
301                     Long MSISDNcuid = LocalizacionUsuario.MSISDNcuidAlarma;
302
303                     if (MSISDNcuidador.equalsIgnoreCase(MSISDNcuid.toString())) {
304
305                             System.out.println("En método onEvent() de InteraccionUsuario.");
306                             System.out.println("Evento capturado en CallControlListener:"
307                                             + ccEvent.getCallEvent());
308
309                             switch (ccEvent.getCallEvent()) {
310
311                             case Event.ADDRESS_ANALYSED:
312
313                                     enviarMenuAlarmaLoc(call);
```

```
314                                     break;
315
316                             default:
317                                     System.out.println("onEvent(): event call capturado no válido:"
318                                                     + ccEvent);
319
320                             }
321                     } else {
322
323                             try {
324                                     // si un cuidador o quien sea realiza una llamada a la central
    se
325                                     // desvía la llamada a una operadora para que lo atienda
326                                     call.routeRequest(NumerosServicio.MSISDNoperadora, 0, 0);
327
328                                     // la aplicación se desentiende de la llamada; ésta seguirá
    activa
329                                     // hasta que una de las dos partes cuelgue
330                                     call.deassign();
331
332                             } catch (CallControlException e) {
333                                     // TODO Auto-generated catch block
334                                     e.printStackTrace();
335                             }
336                     }
337             }
338
339             /**
340              * No se usa.
341              */
342             public void onError(CallControlError arg0) {
343                     // TODO Auto-generated method stub
344                     System.out.println("En método onError() de InteraccionUsuarioLoc.");
345             }
346     }
```

2.6 Paquete MensajeriaMultimedia

2.6.1 MensajeMultimedia.java

```
1   /*
2    * Proyecto Fin de Carrera: Servicio de Teleasistencia basado en OSA/Parlay
3    *
4    * MensajeMultimedia.java
5    */
6
7   package MensajeriaMultimedia;
8
9   import com.lucent.isg.appsdk.mmm.*;
10  import com.lucent.isg.appsdk.framework.*;
```

```
11
12  import java.io.*;
13
14  /**
15   * Clase que ofrece la funcionalidad de enviar un MMS (indicando el fichero de
16   * texto que contiene la información) especificando desde que terminal y a cual.
17   *
18   * @author Raúl Parras Eliche
19   *
20   */
21  public class MensajeMultimedia {
22
23          private MMMAdapter mmmAdapter = null ;
24          private MMMSession mmmSession = null;
25          private String receptores[];
26          private String emisor;
27
28
29          /**
30           * Constructor de la clase. Crea los adaptadores de servicio necesario para enviar
31           * un mensaje multimedia (MMMAdapter y MMMSession).
32           *
33           * @param fwAdapter - FrameworkAdapter del servicio de tele asistencia, necesario
34           * para obtener el resto de adaptadores.
35           */
36          public MensajeMultimedia(FrameworkAdapter fwAdapter){
37
38                  // Creo el adaptador de servicio
39                  try {
40
41                          mmmAdapter = fwAdapter.selectMMMService();
42
43                          System.out.println("Token de servicio de Mensajería Multimedia: "
44                                          + mmmAdapter.getServiceToken() +"\n");
45
46                  } catch (FrameworkException e) {
47                          // TODO Auto-generated catch block
48                          System.out.println("\n--- FrameworkException atrapada ---\n");
49                          e.printStackTrace();
50                          System.out.println("");
51                  }
52          }
53
54
55          /**
```

```
56              * Método que establece las direcciones origen y destino del MMS.
57              *
58              * @param receptores - Lista de direcciones destino para el MMS.
59              * @param emisor - Número que envía el MMS.
60              */
61             public void establecerDirecciones(String[] receptores, String emisor){
62
63                     this.receptores = receptores;
64                     this.emisor = emisor;
65
66             }
67
68
69             /**
70              * Método que envía en el MMS el contenido del archivo de texto que se le pasa
71              * como parámetro. Dicho archivo de texto debe estar en el directorio etc de la
72              * instalación de MiLiFE ISG SDK.
73              *
74              * @param archivo - nombre del fichero que contiene la información ha enviar en
75              * el MMS.
76              */
77             public void enviarMensaje(String archivo, String MSISDNpaciente){
78
79                     try {
80                             // creao una sesión multimedia
81                             mmmSession = mmmAdapter.openMMMSession(receptores, emisor);
82
83                             MMMMessageTreatment treatment[] = new
    MMMMessageTreatment[1];
84                             treatment[0] = new MMMMessageTreatment();
85
86                             File inputFile;
87                             DataInputStream miDStream;
88                             FileInputStream miFStream;
89
90                             String home = "C:/Program Files/LT/ISG SDK/etc/" + archivo;
91                             inputFile = new File(home);
92                             miFStream = new FileInputStream(inputFile);
93                             miDStream = new DataInputStream (miFStream);
94
95
96                             // copio en un primer array de bytes la información del fichero de texto
97                             // correspondiente
98                             byte[] mensaje1 = new byte[(int)inputFile.length()];
99
```

```
100                     for (int j=0; j<(int)inputFile.length();j++){
101                             mensaje1[j] = miDStream.readByte();
102                     }
103
104                     // copio en un segundo array el MSISDN del paciente, por si el
    cuidador
105                     // tiene a su cargo más de un paciente
106                     byte[] mensaje2 = MSISDNpaciente.getBytes();
107
108                     // uno los dos arrays de byte en uno sólo, que es el que se va a enviar
109                     // por SMS
110                     int longitud = (mensaje1.length + mensaje2.length);
111                     byte[] mensaje3 = new byte[longitud];
112
113                     for (int p = 0; p < mensaje1.length; p ++){
114
115                             mensaje3[p] = mensaje1[p];
116                     }
117
118                     for(int r = mensaje1.length; r < longitud; r++){
119
120                             mensaje3[r] = mensaje2[r-mensaje1.length];
121
122                     }
123
124                     mmmSession.sendMessage(emisor, receptores,
    MMMDeliverType.SMS,
125                                     new MMMMessageTreatment[0], mensaje3,
126                                     new org.csapi.mmm.TpMessageHeaderField[0]);
127
128                     System.out.print("Mensaje enviado a: " );
129                     for (int i=0; i < receptores.length; i++){
130
131                             System.out.println(receptores[i]);
132                             System.out.print("               ");
133                     }
134
135                     System.out.println("");
136
137                     // cierro la sesión multimedia
138                     mmmSession.close();
139             } catch (MMMException e) {
140                     // TODO Auto-generated catch block
141                     System.out.println("\n--- MMMException atrapada ---\n");
142                     e.printStackTrace();
143                     System.out.println("");
```

```
144                     } catch (FileNotFoundException e) {
145                             // TODO Auto-generated catch block
146                             e.printStackTrace();
147                     } catch (IOException e) {
148                             // TODO Auto-generated catch block
149                             e.printStackTrace();
150                     }
151             }
152
153             /**
154      * Método que elimina el adaptador de servicio MMMessaging.
155      */
156             public void cleanup(){
157
158                     try {
159
160                             if (mmmAdapter != null){
161
162                                     mmmAdapter.destroy();
163                             }
164                     } catch (MMMException e) {
165                             // TODO Auto-generated catch block
166                             e.printStackTrace();
167                     }
168             }
169     }
```

2.6.2 MensajeMultimediaLis.java

```
1       /*
2        * Proyecto Fin de Carrera: Servicio de Teleasistencia basado en OSA/Parlay
3        *
4        * MensajeMultimediaLis.java
5        */
6
7       package MensajeriaMultimedia;
8
9       import BaseDeDatos.BBDD;
10      import Gestion.GestionAlarmas;
11      import Gestion.GestionPacientes;
12      import TipoDatos.Alarma;
13      import TipoDatos.NumerosServicio;
14      import TipoDatos.Paciente;
15
16      import com.lucent.isg.appsdk.framework.*;
17      import com.lucent.isg.appsdk.mmm.*;
18
```

```
19  import java.sql.ResultSet;
20  import java.sql.SQLException;
21
22
23  /**
24   * Clase que implementa la interfaz MMMListener. Es la encargada de capturar eventos
25   * (de la interfaz OSA/Parlay) relacionados con la recepción de MMS y define los
26   * métodos que procesan dichos eventos.
27   *
28   * @author Raúl Parras Eliche
29   *
30   */
31  public class MensajeMultimediaLis implements MMMListener {
32
33          /**
34           * Adaptadores del servicio de Mensajería Multimedia.
35           */
36          private MMMAdapter mmmAdapter;
37          private MMMSession mmmSession = null;
38
39
40          // se definen los receptores delos MMS que queremos que provoquen los eventos
41          private String receptorMMSalarmaCaida = NumerosServicio.MSISDNalarmaCaida;
42          private String receptorMMSCambioEstadoPaciente =
    NumerosServicio.MSISDNcambioEstPac;
43
44          // origen de los MMS que queremos capturar
45          // para las alarmas por caída, los MMS sólo procederán de los pacientes
46          private String emisor1 = "150111111111";
47          private String emisor2 = "150222222222";
48          private String emisor3 = "150333333333";
49          private String emisor4 = "150444444444";
50          // para los cambios de estado de los pacientes, los MMS procederán de los cuidadores
51          private String emisor5 = "170111111111";
52          private String emisor6 = "170222222222";
53          private String emisor7 = "170333333333";
54
55          /**
56           * Constructor de la clase MensajeMultimediaLis. Utiliza el FrameworkAdapter
57           * para inicializar el adaptador de servicio necesario (MMMAdapter).
58           *
59           * @param fwAdapter - FrameworkAdapter del servicio de tele asistencia, necesario
60           * para obtener el resto de adaptadores.
61           */
62          public MensajeMultimediaLis(FrameworkAdapter fwAdapter){
63
64                  // creo el adaptador del servicio
65                  try {
66
67                          System.out.println("En el constructor de la clase
    MensajeMultimediaLis.");
68
69                          mmmAdapter = fwAdapter.selectMMMService();
70                          System.out.println("Token de sericio de Mensajería Multimedia: "
```

```
71                                              + mmmAdapter.getServiceToken());
72
73                  } catch (FrameworkException e) {
74                          // TODO Auto-generated catch block
75                          e.printStackTrace();
76                  }
77
78          }
79
80
81          /**
82           * Método que establece los criterios de monitorización de MMS y lanza la escucha.
83           *
84           */
85          public void monitorizaMMM(){
86
87                  try {
88                          System.out.println("En el método monitorizaMMM");
89
90                          MMMNotificationRequest criterio[] = new MMMNotificationRequest[7];
91                          criterio[0] = new MMMNotificationRequest(emisor1,
92                                          receptorMMSalarmaCaida,true);
93                          criterio[1] = new MMMNotificationRequest(emisor2,
94                                          receptorMMSalarmaCaida,true);
95                          criterio[2] = new MMMNotificationRequest(emisor3,
96                                          receptorMMSalarmaCaida,true);
97                          criterio[3] = new MMMNotificationRequest(emisor4,
98                                          receptorMMSalarmaCaida,true);
99                          criterio[4] = new MMMNotificationRequest(emisor5,
100                                         receptorMMSCambioEstadoPaciente,true);
101                         criterio[5] = new MMMNotificationRequest(emisor6,
102                                         receptorMMSCambioEstadoPaciente,true);
103                         criterio[6] = new MMMNotificationRequest(emisor7,
104                                         receptorMMSCambioEstadoPaciente,true);
105
106                         mmmAdapter.createNotification(criterio, this);
107
108                         System.out.println("");
109
110
111                 } catch (MMMException e) {
112                         // TODO Auto-generated catch block
113                         e.printStackTrace();
114                 }
115
116         }
117
118         /**
119          * Método llamado por onEvent y que, según el destino del MMS, realiza la acción
120          * correspondiente (bien genera una alarma por caída o bien cambia el estado de
121          * un paciente).
122          *
123          * @param MSISDNdestinoMMS - número de destino del MMS capturado.
124          * @param MSISDNorigenMMS - número del origen del MMS cappturado.
```

```
125             */
126             private void procesaMMS(String MSISDNdestinoMMS, String MSISDNorigenMMS){
127
128                     String sentencia_SQL;
129                     ResultSet rs = null;
130
131                     try {
132                             // comprobamos cual es el destino del MMS; si es para cambiar
133                             // el estado de un paciente, el mensaje procederá de un cuidador,
134                             // por lo que buscamos su dni
135
136                             if
    (MSISDNdestinoMMS.equalsIgnoreCase(receptorMMSCambioEstadoPaciente)){
137
138                                     sentencia_SQL = "SELECT nombre,apellido1,id_cuidador
    FROM cuidadores " +
139                                             "WHERE MSISDN='" +
    MSISDNorigenMMS +"'";
140
141                                     rs = BBDD.consulta(sentencia_SQL);
142                                     rs.next();
143
144                                     System.out.println("Mensaje procedente del cuidador: " +
    rs.getString(1)
145                                             + " " + rs.getString(2) + ", con DNI  " +
    rs.getInt(3) + ".");
146
147                                     // ahora busco los pacientes asociados a este cuidador
148                                     sentencia_SQL = "SELECT MSISDN FROM pacientes
    WHERE id_cuidador='"
149                                             + rs.getInt(3) +"'";
150
151                                     rs = BBDD.consulta(sentencia_SQL);
152                                     long MSISDNpaciente;
153                                     ResultSet rs2 = null;
154
155                                     if (rs.first()){
156
157                                             // al menos tiene un paciente asociado
158                                             do{
159                                                     MSISDNpaciente = rs.getLong(1);
160                                                     sentencia_SQL = "SELECT estadoLoc
    FROM estadopacloc " +
161                                                             "WHERE MSISDN='" +
    MSISDNpaciente + "'";
162
163                                                     rs2 = BBDD.consulta(sentencia_SQL);
164
165                                                     if(rs2.first()){
166
167                                                             switch(rs2.getInt(1)){
168
169                                                             case Paciente.PacATENDIDO:
170                                                                     sentencia_SQL =
```

```
"UPDATE estadopacloc SET estadoLoc='"
171                                                                                 +
Paciente.PacNORMAL + "' WHERE MSISDN='"
172                                                                                 +
MSISDNpaciente + "'";
173
174
                BBDD.actualiza(sentencia_SQL);
175                                                                 System.out.println("Se
ha actualizado el estado " +
176                                                                                 "del
paciente.");
177                                                                 System.out.println("Se
vigilará su posición de nuevo.");
178
179                                                                 // si no estoy en la
última fila, m voy a ella.
180                                                                 // se supone que un
cuidador no va a estar
181                                                                 // atendiendo a más de
un paciente
182                                                                 if (!rs.last()){
183                                                                         rs.last();
184                                                                 }
185
186                                                         default:
187                                                                 // si el estado es
normal (se está vigilando
188                                                                 // su posición) no hago
nada
189                                                         }
190                                                 }

191
192                                         } while(rs.next());
193
194
195                                 } else{
196
197                                         System.out.println("Este cuidador no tiene asociado
" +
198                                                         "ningún paciente.");
199                                 }
200
201                         } else{
202
203                                 // el mensaje proviene de un paciente
204                                 // por lo que se trata de una alarma por caída
205
206                                 // creo un array de string con todos los MSISDN de los
pacientes
207                                 // que tienen este servicio contratado
208
209                                 GestionPacientes g_p = new GestionPacientes();
```

```
210                                     String clientesServCaida[] =
    g_p.listado_MSISDN(Alarma.CAIDA);
211
212                                     boolean servCaidaContratado = false;
213                                     int i = 0;
214
215                                     // compruebo si el cliente del que procede la alarma tiene
216                                     // este servicio contratado
217                                     while ((!servCaidaContratado) && (i <
    clientesServCaida.length)){
218
219                                             if
    (MSISDNorigenMMS.equals(clientesServCaida[i])){
220
221                                                     servCaidaContratado = true;

222                                             }
223
224                                             i++;
225                                     }
226
227                                     if (servCaidaContratado){
228
229                                             sentencia_SQL = "SELECT
    nombre,apellido1,id_paciente FROM " +
230                                                     "pacientes " +"WHERE
    MSISDN='" + MSISDNorigenMMS +"'";
231
232                                             rs = BBDD.consulta(sentencia_SQL);
233                                             rs.next();
234
235                                             System.out.println("Caída del paciente: " +
    rs.getString(1) + " "
236                                                     + rs.getString(2) + ", con DNI  " +
    rs.getInt(3) + ".");
237
238                                             // se genera la alarma
239                                             System.out.println("Se procederá a generar la
    alarma " +
240                                                     "correspondiente.");
241                                             sentencia_SQL = "SELECT MSISDN FROM
    pacientes WHERE id_paciente='"
242                                                     + rs.getInt(3) +"'";
243                                             rs = BBDD.consulta(sentencia_SQL);
244                                             rs.next();
245                                             Alarma a = new Alarma(Alarma.CAIDA,
    rs.getLong(1));
246
247                                             GestionAlarmas.anadeAlarma(a);
248
249                                     } else{
250
251                                             System.out.println("Se ha producido una alarma por
    caída, pero el "+
```

```
252                                                                 "usuario no tiene contratado el
    servicio.");
253                                         }
254                                 }
255                         } catch (SQLException e) {
256                                 // TODO Auto-generated catch block
257                                 e.printStackTrace();
258                         }
259
260                 }
261
262         //================================================
263         // Implementacion de la interfaz MMMListener
264         //================================================
265
266                 /**
267                  * Método llamado por las Convenience Classes cuando se produce alguno de los
268                  * eventos programados por la aplicación.
269                  *
270                  * @param mmmEvent - El evento MultiMediaMessaging que se acaba de producir.
271                  */
272                 public void onEvent(MMMEvent mmmEvent) {
273                         // TODO Auto-generated method stub
274                         System.out.println("Evento capturado en onEvent de MMMListener.");
275                         System.out.println("Recepción de un MMS.");
276
277                         mmmSession = mmmEvent.getSession();
278
279                         try {
280
281                                 System.out.println("Dirección origen del mensaje: "
282                                                 + mmmSession.getDefaultSourceAddress());
283
284                                 String DireccionDestino[] =
    mmmSession.getDefaultDestinationAddressList();
285                                 System.out.println("Dirección destino: " + DireccionDestino[0]);
286
287                                 // llamada al método que procesaría el contenido del mensaje que ha
288                                 // provocado el evento
289                                 procesaMMS(DireccionDestino[0],
    mmmSession.getDefaultSourceAddress());
290
291                         } catch (MMMException e) {
292                                 // TODO Auto-generated catch block
293                                 e.printStackTrace();
294                         }
295                 }
296
297                 /**
298                  * No se usa.
299                  */
300
301                 public void onError(MMMError mmmError) {
302                         // TODO Auto-generated method stub
```

```
303
304             }
305 }
```

2.7 Paquete Gestion

2.7.1 GestionAlarmas.java

```
1   /*
2    * Proyecto Fin de Carrera: Servicio de Teleasistencia basado en OSA/Parlay
3    *
4    * GestionAlarmas.java
5    */
6
7   package Gestion;
8
9   import java.util.Iterator;
10  import java.util.Vector;
11
12  import TipoDatos.Alarma;
13
14  /**
15   * Clase encargada de la gestión de los objetos alarmas en el sistema.
16   * Define un vector donde se almacenarán según se creen. Define métodos para
17   * insertar las alarmas y eliminarlas de manera sincronizada.
18   *
19   * @author Raúl Parras Eliche
20   *
21   */
22  public class GestionAlarmas {
23
24
25              /**
26               * Posible valor de las respuesta de un cuidador ante una alarma de localización.
27               */
28              public static final int ATENDIDO = 1;
29              /**
30               * Posible valor de las respuesta de un cuidador ante una alarma de localización.
31               */
32              public static final int DESCONOCIDO = 2;
33              /**
34               * Posible valor de las respuesta de un cuidador ante una alarma de localización.
35               */
36              public static final int ERROR = 3;
```

```
37          /**
38           * Posible valor de las respuesta de un cuidador ante una alarma de localización.
39           */
40          public static final int TIMEOUT = 4;
41
42
43          /**
44           *  Vector donde se irán almacenando las alarmas que se creen.
45           */
46          public static Vector<Alarma> Alarmas = new Vector<Alarma>();
47
48
49
50          /**
51           * Método que añade una nueva alarma en el vector de alarmas que procesa el
52           * hilo principal. Es un método sincronizado para que no haya problemas de
53           * acceso múltiple al vector.
54           *
55           *  @param alarma - Objeto alarma a insertar.
56           */
57
58          public static void anadeAlarma (Alarma alarma) {
59
60                  synchronized (Alarmas) {
61
62                          Alarmas.addElement(alarma);
63                          System.out.println("La alarma se ha insertado con éxito.");
64                          Alarmas.notify();
65
66                  }
67
68          }
69
70
71          /**
72           *          Método que elimina una alarma en el vector de alarmas que procesa el
73           *  hilo principal. Es un método sincronizado para evitar accesos múltiples.
74           *
75           *  @param alarma - Objeto alarma a eliminar del vector.
76           */
77
78          public static void eliminaAlarma (Alarma alarma) {
79
80                  synchronized (Alarmas) {
81
```

```
82                              if (Alarmas.removeElement(alarma)){
83
84                                      System.out.println("La alarma se ha eliminado con éxito.");
85                                      Alarmas.notify();
86
87                              }else{
88                                      System.out.println("La alarma no se ha podido eliminar.");
89                              }
90
91                      }
92              }
93
94
95              /**
96               * Método que comprueba si ya se ha generado una alarma con los parámetros que se le
97               * pasan. Esto impedirá generar la misma alarma dos veces (por si se da la posibilidad de
98               * que el hilo hijo vuelva a comprobar la misma situacion anómala dos o más veces sin que
    el
99               * hilo principal haya procesado la alarma).
100              *
101              * @param alarma - tipo de  alarma que se quiere comprobar.
102              * @param msisdn - número de abonado creador de la alarma que se quiere comprobar.
103              * @return alarmaCreada - True si la alarma ya se había creado previamente; False en
104              * cao contrario.
105              */
106
107             public static synchronized boolean compruebaAlarmaCreada(int alarma, long msisdn){
108
109                     boolean alarmaCreada = false;
110
111                     Iterator it = Alarmas.iterator();
112
113                     while (it.hasNext() && !alarmaCreada){
114
115                             Alarma a = (Alarma) it.next();
116
117                             if (a.MSISDN == msisdn){
118
119                                     if (a.id_alarma == alarma){
120
121                                             alarmaCreada = true;
122                                     }
123                             }
124                     }
125                     return alarmaCreada;
```

```
126             }
127     }
```

2.7.2 GestionAmbulancias.java

```
1   /*
2    * Proyecto Fin de Carrera: Servicio de Teleasistencia basado en OSA/Parlay
3    *
4    * GestionAlarmas.java
5    */
6
7   package Gestion;
8
9   import java.sql.ResultSet;
10  import java.sql.SQLException;
11  import java.util.Iterator;
12  import java.util.Vector;
13
14  import BaseDeDatos.BBDD;
15  import EstadoTerminal.EstadoUsuario;
16  import TipoDatos.Ambulancia;
17
18  import com.lucent.isg.appsdk.framework.FrameworkAdapter;
19  import com.lucent.isg.appsdk.userlocation.UserLocation;
20  import com.lucent.isg.appsdk.userlocation.UserLocationAdapter;
21  import com.lucent.isg.appsdk.userlocation.UserLocationException;
22  import com.lucent.isg.appsdk.userstatus.UserStatus;
23
24  /**
25   * Clase encargada de la gestión de los objetos ambulancia.
26   *
27   * @author Raúl Parras Eliche
28   */
29  public class GestionAmbulancias {
30
31          /**
32           * Vector que contiene todas las ambulancias disponibles para la aplicación.
33           */
34          public static Vector<Ambulancia> Ambulancias = new Vector<Ambulancia>();
35
36          /**
37           * Método que inicializa el vector Ambulancias
38           */
39          public static void listaAmbulancias(){
```

```
40
41                  Ambulancia amb;
42
43                  String sentencia_SQL = "SELECT MSISDN FROM ambulancias";
44                  ResultSet rs = BBDD.consulta (sentencia_SQL);
45
46                  try {
47
48                          while (rs.next()){
49                                  // las ambulancias están inicialmente todas libres
50                                  amb = new Ambulancia (rs.getLong(1), false);
51                                  Ambulancias.addElement(amb);
52
53                          }
54
55                  } catch (SQLException e) {
56                          // TODO Auto-generated catch block
57                          e.printStackTrace();
58                  }
59          }
60
61          /**
62           * Método que devuelve el MSISDN de la ambulancia disponible más cercana a la
63           * posición que se le pasa como parámetro. Se comprueba también que el terminal
64           * móvil de la ambulancia esté encendido (REACHABLE).
65           *
66           * @param locPac - posición del paciente que necesita la ambulancia.
67           * @param ulAdapter - Adaptador del servicio de localización.
68           * @param fwAdapter - FrameworkAdapter del servicio de tele asistencia, necesario
69           * para obtener el resto de adaptadores.
70           * @return MSISDN - Número del terminal de la ambulancia disponible y más cercana
71           *  al paciente.
72           */
73          public static String ambulanciaMasCercana(UserLocation locPac,
74                          UserLocationAdapter ulAdapter, FrameworkAdapter fwAdapter){
75
76                  // Creo un array de String con todos los MSISDN de las ambulancias que no
77                  // están ocupadas en este momento y el estado de su terminal sea REACHABLE.
78                  // Primero creo un vector, porque puede que el número total de ambulancias
79                  // no coincida con las disponibles.
80                  //*************************************************************************
81                  Vector <String> MSISDNamb= new Vector<String>();
82
83                  EstadoUsuario eu = new EstadoUsuario(fwAdapter);
```

```
84
85                      Iterator it = Ambulancias.iterator();
86                      Ambulancia a;
87                      Long l;
88
89                      while (it.hasNext()){
90                              a = (Ambulancia)it.next();
91
92                              // si la ambulancia no está ocupada y su estado es REACHABLE,
93                              // copio su MSISDN al Vector MSISDNamb
94
95                              if (!a.ocupado){
96
97                                      l = a.MSISDN;
98
99                                      if (eu.estadoTerminal(l.toString()) ==
    UserStatus.US_REACHABLE){
100
101                                             MSISDNamb.addElement(l.toString());
102                                     }
103                             }
104                     }
105
106                     // elimino el adaptador de servicio UserStatus
107                     eu.cleanup();
108
109
110                     String MSISDN = null;
111
112                     switch (MSISDNamb.size()){
113
114                     case 0:
115                             // no hay ambulancias disponibles
116                             // se devolverá null
117                             break;
118
119                     case 1:
120                             // sólo hay una ambulancia disponible
121                             // no hay que calcular cual es la más cercana
122                             MSISDN = MSISDNamb.firstElement();
123
124                             break;
125
126                     default:
127                             String[] amb = new String[MSISDNamb.size()];
```

```
128
129                 it = MSISDNamb.iterator();
130                 int i =0;
131                 String s;
132
133                 while (it.hasNext()){
134
135                         s = (String)it.next();
136                         amb[i] = s;
137                         i++;
138                 }
139
140                 // Obtengo las posiciones de las ambulancias
141                 // ********************************************************************************
142                 UserLocation[] locAmb = null;
143
144                 try {
145
146                         locAmb = ulAdapter.requestLocation(amb);
147
148                 } catch (UserLocationException e) {
149                         // TODO Auto-generated catch block
150                         e.printStackTrace();
151                 }
152
153                 // Calculo la distancia entre cada ambulancia y el paciente
154                 // ********************************************************************************
155
156                 double d1 = 0;
157                 double d2 = 0;
158
159                 for (int j= 0; j<amb.length; j++){
160
161                         if (j == 0){  // para la primera vez
162
163                                 d1 = locAmb[j].distanceTo(locPac);
164                                 MSISDN = locAmb[j].getUser();
165
166                         } else{
167
168                                 d2 = locAmb[j].distanceTo(locPac);
169                                 if (d2 < d1){
170                                         d1 = d2;
171                                         MSISDN = locAmb[j].getUser();
```

```
172                                                     }
173                                                 }
174                                             }
175                                         }
176                                         return MSISDN;
177                 }
178 }
```

2.7.3 GestionCaida.java

```
1   /*
2    * Proyecto Fin de Carrera: Servicio de Teleasistencia basado en OSA/Parlay
3    *
4    * GestionCaida.java
5    */
6
7   package Gestion;
8
9   import TipoDatos.Alarma;
10
11  import com.lucent.isg.appsdk.framework.FrameworkAdapter;
12  import com.lucent.isg.appsdk.framework.FrameworkException;
13  import com.lucent.isg.appsdk.userlocation.UserLocation;
14  import com.lucent.isg.appsdk.userlocation.UserLocationAdapter;
15  import com.lucent.isg.appsdk.userlocation.UserLocationException;
16
17  /**
18   * Clase que se encarga del tratamiento de las alarmas originadas por caída
19   * de un paciente.
20   *
21   * @author Raúl Parras Eliche
22   *
23   */
24  public class GestionCaida {
25
26          private FrameworkAdapter fwAdapter = null;
27          private Script script = null;
28          private UserLocationAdapter ulAdapter = null;
29
30          /**
31           * Construye un objeto de la clase GestionCaida e inicializa el adaptador
32           * UserLocation mediante el FramworkAdapter.
33           *
34           * @param fwAdapter - FrameworkAdapter del servicio de tele asistencia, necesario
```

```
35          * para obtener el resto de adaptadores.
36          * @param script - Clase para cargar y ejecutar programas en el simulador.
37          */
38         public GestionCaida(FrameworkAdapter fwAdapter, Script script){
39
40                 this.fwAdapter = fwAdapter;
41                 this.script = script;
42
43                 try {
44                         ulAdapter = fwAdapter.selectUserLocationService();
45
46                 } catch (FrameworkException e) {
47                         // TODO Auto-generated catch block
48                         e.printStackTrace();
49                 }
50         }
51
52
53         /**
54          * Método encargado del procesamiento de la alarma por caída.
55          *
56          * @param alarma - Objeto alarma a tratar.
57          */
58         public void procesaAlarmaCaida(Alarma alarma){
59
60                 // necesito la posición del paciente que ha generado la alarma, para ver
61                 // cual es la ambulancia más cercana
62                 Long MSISDNpac = alarma.MSISDN;
63
64                 // obtengo la posicion de los abonados
65                 UserLocation locPac = null;
66
67                 try {
68
69                         locPac = ulAdapter.requestLocation(MSISDNpac.toString());
70
71
72                 } catch (UserLocationException e1) {
73                         // TODO Auto-generated catch block
74                         System.out.println("\n--- UserLocationException caught ---\n");
75                         e1.printStackTrace();
76                 }
77
78
79                 String MSISDNamb = GestionAmbulancias.ambulanciaMasCercana(locPac,
```

```
80                                      ulAdapter,fwAdapter);
81
82                      if (MSISDNamb != null){
83
84                              String programa = "LLamada_Central_" + MSISDNamb;
85                              script.ejecutaPrograma(programa, false);
86                              GestionAlarmas.eliminaAlarma(alarma);
87                      }
88              }
89      }
```

2.7.4 GestionPacientes.java

```
1       /*
2        * Proyecto Fin de Carrera: Servicio de Teleasistencia basado en OSA/Parlay
3        *
4        * GestionPacientes.java
5        */
6
7       package Gestion;
8
9       import java.sql.ResultSet;
10      import java.sql.SQLException;
11      import java.util.Iterator;
12      import java.util.Vector;
13
14      import BaseDeDatos.BBDD;
15      import TipoDatos.Alarma;
16      import TipoDatos.Paciente;
17
18      /**
19       * Clase que define los métodos necesarios para gestionar los objetos pacientes.
20       *
21       * @author Raúl Parras Eliche.
22       */
23      public class GestionPacientes {
24
25              /**
26               * Metodo que devuelve un Vector con todos los objetos pacientes que tenemos en la
        BBDD,
27               * pero sólo aquellos que tienen contratado algún servicio.
28               *
29               * @return Pacientes - Listado de pacientes con algún servicio contratado.
30               */
```

```
31          public Vector listaPacientes(){
32
33                  // vector que devuelve el método
34                  Vector<Paciente> Pacientes = new Vector<Paciente>();
35
36                  String sentencia_SQL = "SELECT id_paciente,MSISDN FROM
    pacientes";
37                  ResultSet rs = BBDD.consulta (sentencia_SQL);
38
39                  Vector<Integer> id_pacientes = new Vector<Integer>();
40                  Vector<Long> MSISDN = new Vector<Long>();
41
42                  // obtengo en un vector todos los pacientes almacenados en la BBDD
43                  // y otro con sus MSISDN
44
45                  try {
46
47                      while (rs.next()){
48
49                          id_pacientes.addElement(rs.getInt(1));
50                          MSISDN.addElement(rs.getLong(2));
51                      }
52                  } catch (SQLException ex) {
53                      // TODO Auto-generated catch block
54                      System.out.println("\n--- SQLException caught ---\n");
55                      while (ex != null) {
56                          System.out.println("Message:   " + ex.getMessage
    ());
57                          System.out.println("SQLState:  " + ex.getSQLState
    ());
58                          System.out.println("ErrorCode: " + ex.getErrorCode
    ());
59                          ex = ex.getNextException();
60                          System.out.println("");
61                      }
62                  }
63
64                  for (int i=0; i < id_pacientes.size(); i++){
65
66
67                      // esta consulta me va a devolver todas las alarmas que tengo
    que
68                      // tener en cuanta para el paciente anterior
69
70                      sentencia_SQL = "SELECT cod_alarma FROM" +
71                                      "servicios_contratados WHERE " +
72                                      "id_paciente='" +
```

```
73                                      id_pacientes.elementAt(i) + "'";
74                                      rs = BBDD.consulta(sentencia_SQL);
75
76                                      try {
77                                              if (rs.next()){
78                                                      //por lo menos ha contratado un servicio
79                                                      Paciente pac = new Paciente();
80                                                      pac.id_paciente =
    id_pacientes.elementAt(i);
81                                                      pac.MSISDN = MSISDN.elementAt(i);
82                                                      pac.servicios.addElement (new
    Integer(rs.getInt(1)));
83
84                                                      // Bucle para almacenar en el vector el
    resto de
85                                                      // los servicios  contratados
86                                                      while (rs.next()){
87                                                              pac.servicios.addElement(new
    Integer(rs.getInt(1)));
88                                                      }
89
90                                                      // Introduzco el objeto paciente recién
    creado en el vector Pacientes
91                                                      Pacientes.addElement(pac);
92
93                                              }
94
95                                      } catch (SQLException ex) {
96                                              // TODO Auto-generated catch block
97                                              System.out.println("\n--- SQLException caught ---
    \n");
98                                              while (ex != null) {
99                                                      System.out.println("Message:   " +
    ex.getMessage ());
100                                                     System.out.println("SQLState:  " +
    ex.getSQLState ());
101                                                     System.out.println("ErrorCode: " +
    ex.getErrorCode ());
102                                                     ex = ex.getNextException();
103                                                     System.out.println("");
104                                             }
105                                     }
106
107                             }
108
109                             // vacío los vectores pacientes y MSISDN
110                             id_pacientes.clear();
111                             MSISDN.clear();
```

```
112
113                         return Pacientes;
114                 }
115
116         /**
117          * Metodo que imprime por la salida estandar un listado de pacientes.
118          *
119          * @param Pacientes - Listado de pacientes a imprimir.
120          */
121         public void imprimeVectorPacientes(Vector Pacientes){
122
123                 Iterator it1 = Pacientes.iterator();
124
125                 System.out.println("");
126                 System.out.println("************ Listado de pacientes ************");
127                 System.out.println("Id_paciente******MSISDN******Alarmas Contratadas**");
128
129                 Paciente pac = null;
130
131                 while (it1.hasNext()){
132
133                         pac = (Paciente) it1.next();
134                         System.out.print("  " + pac.id_paciente + "    " + pac.MSISDN +"  ");
135                         Vector serv = pac.servicios;
136
137                         Iterator it2 = serv.iterator();
138                         while (it2.hasNext()){
139
140                                 Integer alarma = (Integer)it2.next();
141                                 System.out.print("  " + alarma.toString());
142                         }
143                         System.out.println();
144                 }
145                 System.out.println("");
146
147         }
148
149         /**
150          * Método que devuelve un array de string con los MSISDN de los pacientes que tienen
151          * contratado el servicio que recibe como parámetro.
152          *
153          * @param servicio - Código del servicio del que queremos saber qué pacientes lo
154          * lo tienen contratado.
155          * @return UL_abonados - lista de MSISDN de los pacientes que tienen el servicio
156          * contratado
```

```
157         */
158
159         public String[] listado_MSISDN (int servicio){
160
161                 //  creo un vector y no un array de String directamente porque el numero de
162                 //   abonados a este servicio puede variar
163
164                 Vector<Long> abonados = new Vector<Long>();
165
166                 // primero obtengo un listado de todos los pacientes que tienen algún servicio
167                 // contratado
168                 Vector Pacientes = listaPacientes();
169
170                 Iterator it1 = Pacientes.iterator();
171                 Paciente p = null;
172                 Vector serv = null;
173                 Iterator it2 = null;
174                 Integer alarma;
175
176                 while (it1.hasNext()){
177
178                         p = (Paciente) it1.next();
179                         serv = p.servicios;
180
181                         it2 = serv.iterator();
182
183                         while (it2.hasNext()){
184                                 alarma = (Integer)it2.next();
185
186                                 if (alarma == servicio)
187                                         abonados.addElement(new Long(p.MSISDN));
188                         }
189
190                 }
191
192                 // convierto el vector de abonados UL a un array de String
193
194                 String[] usuarios = new String[abonados.size()];
195
196                 System.out.println("El vector de abonados para el servicio de "
197                                 + Alarma.tipoAlarma_toString(servicio) + ": ");
198
199                 Iterator it3 = abonados.iterator();
200                 int i =0;
```

```
201                     Long l;
202                     while (it3.hasNext()){
203                             l = (Long)it3.next();
204                             usuarios[i] = l.toString();
205                             System.out.println("  ---->> " + usuarios[i]);
206                             i++;
207                     }
208                     System.out.println("");
209
210                     abonados.clear();
211                     Pacientes.clear();
212                     serv.clear();
213
214                     return usuarios;
215             }
216     }
```

2.7.5 Script.java

```
1   /*
2    * Proyecto Fin de Carrera: Servicio de Teleasistencia basado en OSA/Parlay
3    *
4    * Script.java
5    */
6
7
8   package Gestion;
9
10  import com.lucent.isg.appsdk.script.*;
11  /**
12   * Clase que implementa la interfaz ScriptCallback. Define los métodos necesarios
13   * para cargar y descargar scripts y para ejecutar programas en el simulador.
14   *
15   * @author Raúl
16   *
17   */
18  public class Script implements ScriptCallback {
19
20
21          // Clase para acceder a la interfaz ScriptProcessor.
22      private ScriptAdapter scriptAdapter = null;
23
24      // variable que indicará si el programa debe esperar la ejecucion en el simulador de
25      // un programa del script.
```

```
26      private boolean waitUntilStopped = true;
27
28
29      /**
30       * Constructor que ejecuta la inicializacion del script adapter y registra
31       * la interfaz ScriptCallback, para que los métodos onXXX de la clase sean
32       * llamados cuando se produzcan eventos importantes.
33       */
34      public Script() {
35
36              System.out.println("Inicializando ScriptAdapter ...");
37          try {
38
39              // creamos el objeto SriptAdapter
40              scriptAdapter = ScriptAdapterFactory.createScriptAdapter();
41
42              // register a callback interface (= this), causing the onXXXX()
43              // methods to be called upon important events
44              ScriptStatus status = scriptAdapter.registerCallback(this);
45
46              // display the status of the Script Processor
47              System.out.println("Status: " + status);
48
49          } catch (ScriptException e) {
50              e.printStackTrace();
51          }
52      }
53
54      /**
55       * Carga el script que se le pase como parametro en el simulador.
56       * Muestra ,además, el nombre de la simulacion así como todos los programas
57       * y pasos de que consta dicho archivo XML.
58       *
59       * @param scriptfile - Archivo XML que se cargará en el simulador.
60       */
61      public void cargaScript(String scriptfile) {
62
63              System.out.println("Cargando script ...");
64          try {
65              // si una simulacion (script) ya esta cargada, se descarga:
66              if ( scriptAdapter.isScriptLoaded() ) {
67                  scriptAdapter.unloadScript();
68                  System.out.println("Anterior script descargado.");
69              }
70
```

```
71              // cargamos el script de teleasistencia:
72              scriptAdapter.loadScript(scriptfile);
73
74
75              // obtenemos los programas disponibles:
76              String[] programs = scriptAdapter.getPrograms();
77              System.out.println("Los programas son:");
78              for ( int i = 0; i < programs.length; i++ )
79                  System.out.println("   [" + i + "]=" + programs[i]);
80
81              // obtenemos los pasos disponibles:
82              String[] steps = scriptAdapter.getSteps();
83              System.out.println("Los pasos son:");
84              for ( int i = 0; i < steps.length; i++ )
85                  System.out.println("   [" + i + "]=" + steps[i]);
86
87
88          } catch (ScriptException e) {
89                  e.printStackTrace();
90          }
91
92      }
93
94
95      /**
96       * Ejecuta un programa en el simulador.
97       *
98       * @param program - Nombre del programa a ejecutar en el simulador.
99       * @param waitUntilStopped - Boolean para indicar si queremos esperar que termine
100      *       la ejecución del programa en el simulador (true para que espere).
101      */
102     public void ejecutaPrograma(String program, boolean waitUntilStopped) {
103
104             this.waitUntilStopped = waitUntilStopped;
105
106         System.out.println("Going to run program \"" + program + "\"");
107         try {
108
109             scriptAdapter.runProgram(program, ScriptStatus.CONTINUOUS);
110
111         } catch (ScriptException e) {
112             e.printStackTrace();
113
114         }
115
```

```
116         if ( waitUntilStopped ) {
117            synchronized (this) {
118               try {
119                  // espera el notify() en onStopped()
120                  wait();
121               } catch (Exception e) { }
122            }
123         }
124
125      }
126
127
128      /**
129       * Descarga el script que haya en el simulador
130       *
131       */
132      public void descargaScript(){
133
134                  try {
135                      scriptAdapter.unloadScript();
136
137                   } catch (ScriptException e) {
138                      e.printStackTrace();
139
140                   }
141      }
142
143
144      /**
145       * Termina el ScriptAdapter.
146       */
147
148              public void destroy() {
149
150         try {
151            // destroy the Script Adapter:
152            scriptAdapter.destroy();
153
154         } catch (ScriptException e) {
155            e.printStackTrace();
156         }
157      }
158
159      //===========================================
160      // Implementacion de la interfaz ScriptCallback
```

```
161     //=============================================
162
163     /**
164      * Método llamado cuando el proceso de registro es cancelado.
165      */
166     public void onUnregistered()
167     {
168             // System.out.println("onUnregistered() called");
169     }
170
171     /**
172      * Método llamado cuando un script se carga con éxito en el simulador.
173      *
174      * @param filename      name of script file
175      * @param simulation    name of simulation
176      */
177     public void onScriptLoaded(String filename, String simulation)
178     {
179        //System.out.println("onScriptLoaded() called");
180        //System.out.println("   Archivo cargado = " + filename);
181        //System.out.println("   Nombre de la simulacion = " + simulation);
182     }
183
184     /**
185      * Método llamado cuando se descarga un script del simulador.
186      */
187     public void onScriptUnloaded()
188     {
189        //System.out.println("onScriptUnloaded() called");
190     }
191
192     /**
193      * Método llamado cuando un programa del script comienza a ejecutarse.
194      */
195     public void onRunning(String program, int mode)
196     {
197        //System.out.println("onRunning() called");
198        //System.out.println("   Nombre del programa = " + program);
199        //System.out.println("   Modo de ejecución = " + ScriptStatus.modeName(mode));
200     }
201
202     /**
203      * Método llamado cuando se detiene la ejecución de un  programa.
204      * Si al ejecutar un programa le dimos el valor true a la variable waitUntilStopped,
205      * este método hará un Notify() avisando al hilo que llamó a ejecutaPrograma, que
```

```
206      * estaba esperando, para avisarle de que el simulador ha terminado la ejecución.
207      */
208     public void onStopped()
209     {
210        //System.out.println("onStopped() called en clase Script.");
211
212        if ( waitUntilStopped ) {
213           synchronized (this) {
214              try {
215                 // notify wait() in runProgram()
216                 notify();
217              } catch (Exception e) { }
218           }
219        }
220     }
221
222     /**
223      * Método llamado cuando de pausa la ejecución de un programa.
224      */
225     public void onPaused()
226     {
227        System.out.println("onPaused() called");
228     }
229
230     /**
231      * Método llamado cuando se ha terminado la ejecución de un paso de un programa.
232      */
233     public void onExecutedProgramStep(String program, String nextStep, String errors)
234     {
235        //System.out.println("onExecutedProgramStep() called");
236        //System.out.println("   program = " + program);
237        //System.out.println("   nextStep = " + nextStep);
238        //System.out.println("   errors = " + errors);
239     }
240
241     /**
242      * Método llamado cuando un paso (fuera del contexto de un programa) ha sido ejecutado.
243      */
244     public void onExecutedStep(String step, String errors)
245     {
246        //System.out.println("onExecutedStep() called");
247        //System.out.println("   step = " + step);
248        //System.out.println("   errors = " + errors);
249     }
250  }
```

2.8 Paquete TipoDatos

2.8.1 Alarma.java

```
1   /*
2    * Proyecto Fin de Carrera: Servicio de Teleasistencia basado en OSA/Parlay
3    *
4    * Alarma.java
5    */
6
7   package TipoDatos;
8
9   /**
10   * Clase que define las características de las alarmas.
11   *
12   * @author Raúl Parras Eliche
13   *
14   */
15  public class Alarma {
16
17          /**
18           * Tipo de alarma.
19           */
20          public static final int CAIDA = 1;
21          /**
22           * Tipo de alarma.
23           */
24      public static final int PERSONA_PERDIDA = 2;
25
26          /**
27           * Código de identificación de la alarma.
28           */
29          public int id_alarma;
30
31          /**
32           * MSISDN del paciente que genera la alarma.
33           */
34          public long MSISDN;
35
36          /**
37           * Prioridad asociada a la naturaleza de la alarma.
38           */
39          public int priAlarma;
40
41          /**
42           * Prioridad asociada al tiempo de reacción requerido frente a la alarma.
43           */
44          public int priReaccion;
45
46          /**
47           * Estado actual de la alarma, que a su vez indicará si la alarma está o no
48           * activa y  si está pendiente o no de ser atendida por el sistema.
49           */
```

```
50              public int estadoAlarma;
51
52              /**
53               * Identificador del operario que ha atendido la alarma.
54               */
55              public int idOperario;
56
57
58
59              /**
60               * Construye un objeto vacío.
61               */
62              public Alarma (){
63
64                      this.id_alarma = -1;
65                      this.MSISDN = -1;
66
67              }
68
69              /**
70               * Construye un nuevo objeto Alarma con los datos especificados id_alarma y MSISDN.
71               * @param id_alarma
72               * @param MSISDN
73               */
74               public Alarma(int id_alarma, long MSISDN){
75
76                      this.id_alarma = id_alarma;
77                      this.MSISDN = MSISDN;
78
79              }
80
81              /**
82               * Método que devuelve el nombre del servicio asociado al número de alarma
    especificado.
83               * @param alarma
84               * @return nombre_alarma
85               */
86              public static String tipoAlarma_toString(int alarma){
87
88                      String nombre_alarma;
89
90                      switch (alarma) {
91                      case CAIDA:
92                              nombre_alarma = "detección de caídas";
93                              break;
94
95                      case PERSONA_PERDIDA:
96                              nombre_alarma = "detección de personas perdidas";
97                              break;
98
99                      default:
100                             nombre_alarma = "servicio desconocido";
101                             break;
102                     }
```

```
103
104                     return nombre_alarma;
105             }
106
107     }
```

2.8.2 Ambulancia.java

```
1       /*
2        * Proyecto Fin de Carrera: Servicio de Teleasistencia basado en OSA/Parlay
3        *
4        * Ambulancia.java
5        */
6
7       package TipoDatos;
8
9       /**
10       * Clase que define las caraterísticas de los objetos Ambulancia.
11       *
12       * @author Raúl Parras Eliche
13       *
14       */
15      public class Ambulancia {
16
17              /**
18               * Número del terninal de la ambulancia.
19               */
20               public long MSISDN;
21
22              /**
23               * Estado en que se encuentra la ambulancia: si está libre o si se está utilizando
24               * con algún paciente.
25               */
26               public boolean ocupado;
27
28               /**
29                * Construye un objeto Ambulancia con los datos especificados MSISDN y ocupado.
30                *
31                * @param MSISDN
32                * @param ocupado
33                */
34              public Ambulancia(long MSISDN, boolean ocupado){
35
36                      this.MSISDN = MSISDN;
37                      this.ocupado = ocupado;
38              }
39
```

2.8.3 NumerosServicio.java

```
1       /*
```

```
2       * Proyecto Fin de Carrera: Servicio de Teleasistencia basado en OSA/Parlay
3       *
4       * NumerosServicio.java
5       */
6
7       package TipoDatos;
8
9       /**
10      * Interfaz que define los números de los terminales que prestan el servicio.
11      * Se definen los números de la central, de la operadora y de los servicios
12      * de recepción de alarmas de caídas y de peticiones de cambios del estado
13      * de un  paciente.
14      *
15      * @author Raúl Parras Eliche
16      *
17      */
18      public interface NumerosServicio {
19
20              /**
21               * Número de terminal principal del servicio.
22               */
23          public static final String MSISDNcentral = "160111111111";
24
25          /**
26           * Número de terminal de la operadora.
27           */
28          public static final String MSISDNoperadora = "160222222222";
29
30          /**
31           * Número de terminal para recepción de las alarmas de caída.
32           */
33          public static final String MSISDNalarmaCaida = "160333333333";
34
35          /**
36           * Número de terminal para la recepción de los cambios de estado de los
37           * pacientes en el servicio de localización.
38           */
39          public static final String MSISDNcambioEstPac = "160444444444";
40
41      }
```

2.8.4 Paciente.java

```
1       /*
2       * Proyecto Fin de Carrera: Servicio de Teleasistencia basado en OSA/Parlay
3       *
4       * Paciente.java
5       */
6
7       package TipoDatos;
8
```

```
9   import java.util.Vector;
10  /**
11   * Clase que contiene los datos relacionados con un paciente.
12   *
13   * @author Raúl Parras Eliche
14   *
15   */
16  public class Paciente {
17
18      /**
19       * Posible estado de un paciente del servicio de localización
20       */
21      public static final int PacNORMAL = 1;
22
23      /**
24       * Posible estado de un paciente del servicio de localización
25       */
26      public static final int PacATENDIDO = 2;
27
28
29              /**
30               * Entero que almacenará el dni del paciente.
31               */
32      public int id_paciente;
33
34
35              /**
36               * Número del termninal del paciente.
37               */
38      public long MSISDN;
39
40
41              /**
42               * Vector para almacenar los servicios contratados por cada cliente.
43               * @uml.property name="servicios"
44               */
45              public Vector<Integer> servicios = new Vector<Integer>();
46
47  }
```

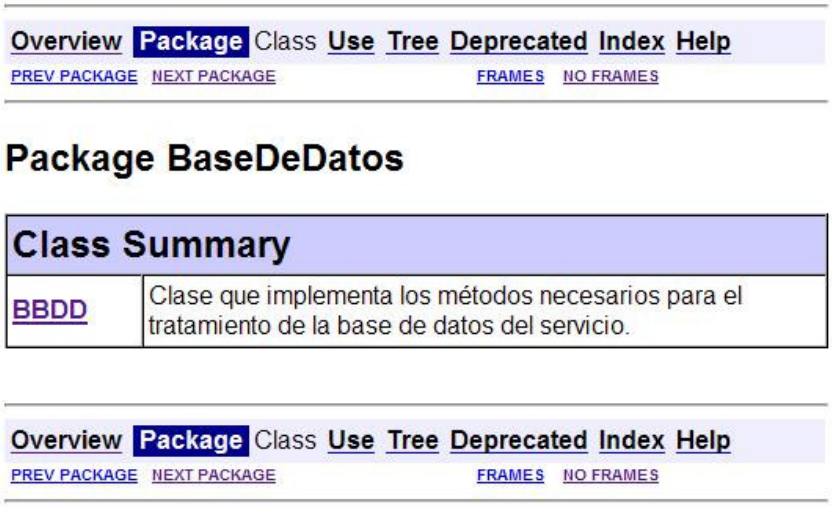
3. Anexo III: Javadoc

A continuación se muestran algunas capturas de la documentación de la aplicación generada mediante la herramienta JavaDoc, en las que se aprecian todos los paquetes que han sido necesarios elaborar.

Paquetes de la aplicación

Figura 62. ***Lista de Paquetes***

Paquete BaseDeDatos



Overview Package Class Use Tree Deprecated Index Help

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Package BaseDeDatos

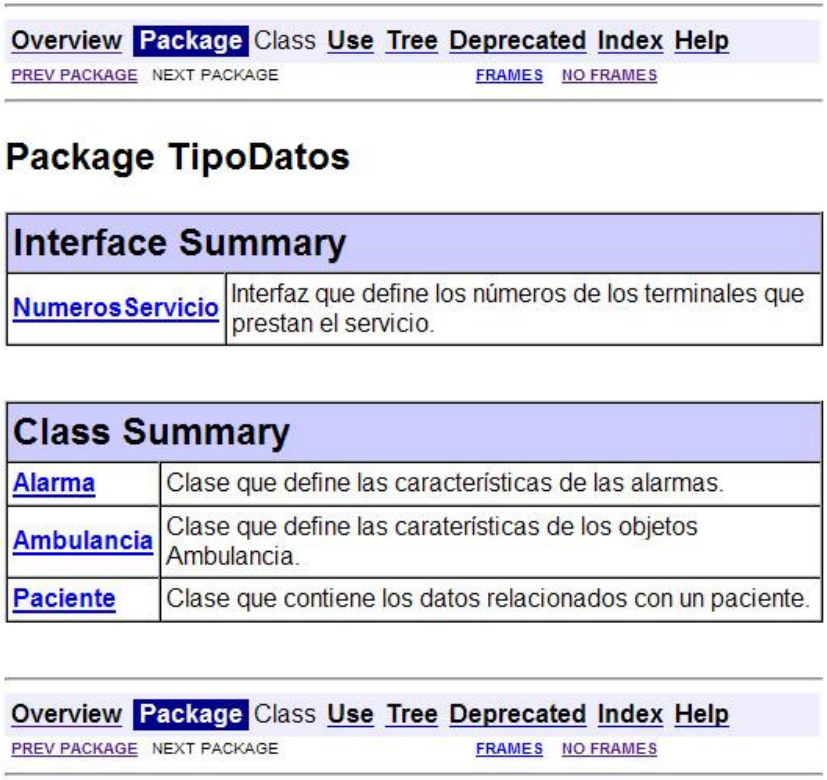
Class Summary	
BBDD	Clase que implementa los métodos necesarios para el tratamiento de la base de datos del servicio.

Overview Package Class Use Tree Deprecated Index Help

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Figura 63. ***Paquete BaseDeDatos***

Paquete TipoDatos



Overview Package Class Use Tree Deprecated Index Help

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Package TipoDatos

Interface Summary	
NumerosServicio	Interfaz que define los números de los terminales que prestan el servicio.

Class Summary	
Alarma	Clase que define las características de las alarmas.
Ambulancia	Clase que define las caraterísticas de los objetos Ambulancia.
Paciente	Clase que contiene los datos relacionados con un paciente.

Overview Package Class Use Tree Deprecated Index Help

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Figura 64. ***Paquete TipoDatos***

Paquete Localización

Overview Package Class Use Tree Deprecated Index Help

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Package Localizacion

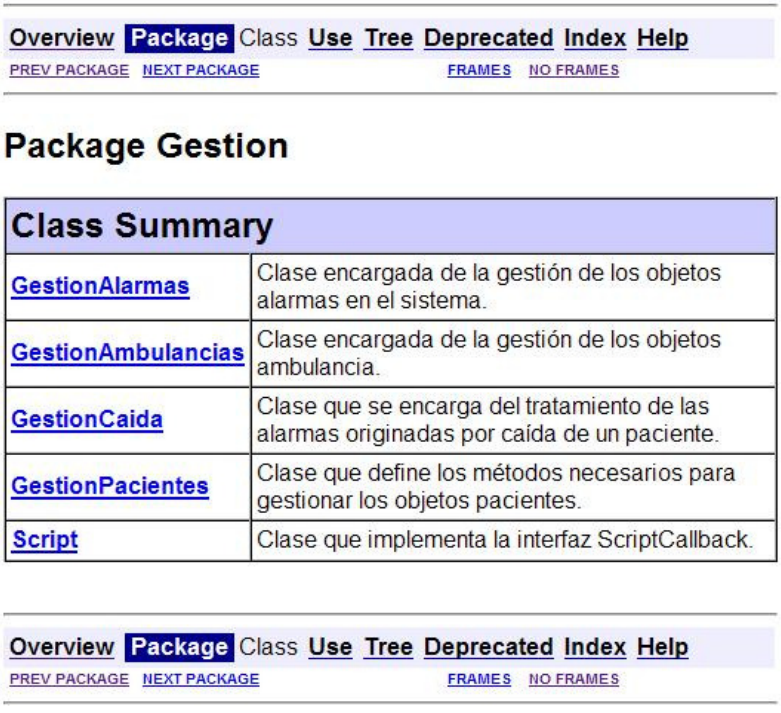
Class Summary	
HiloLoc	Clase que implementa un nuevo hilo para el servicio de localización.
LocalizacionUsuario	Clase que engloba todo lo relacionado con el servicio de localización de los pacientes.

Overview Package Class Use Tree Deprecated Index Help

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Figura 65. ***Paquete Localización***

Paquete Gestión



Overview Package Class Use Tree Deprecated Index Help

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Package Gestion

Class Summary	
GestionAlarmas	Clase encargada de la gestión de los objetos alarmas en el sistema.
GestionAmbulancias	Clase encargada de la gestión de los objetos ambulancia.
GestionCaida	Clase que se encarga del tratamiento de las alarmas originadas por caída de un paciente.
GestionPacientes	Clase que define los métodos necesarios para gestionar los objetos pacientes.
Script	Clase que implementa la interfaz ScriptCallback.

Overview Package Class Use Tree Deprecated Index Help

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Figura 66. ***Paquete Gestión***

Paquete InteraccionUsuario

Overview Package Class Use Tree Deprecated Index Help

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Package InteraccionUsuario

Class Summary	
InteraccionUsuarioLoc	Clase que engloba todo lo relacionado con la captura de las llamadas, pues implementa la interfaz CallControlListener, y establece la interacción con los usuarios.

Overview Package Class Use Tree Deprecated Index Help

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Figura 67. ***Paquete InteraccionUsuario***

Paquete EstadoTerminal

Overview Package Class Use Tree Deprecated Index Help

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Package EstadoTerminal

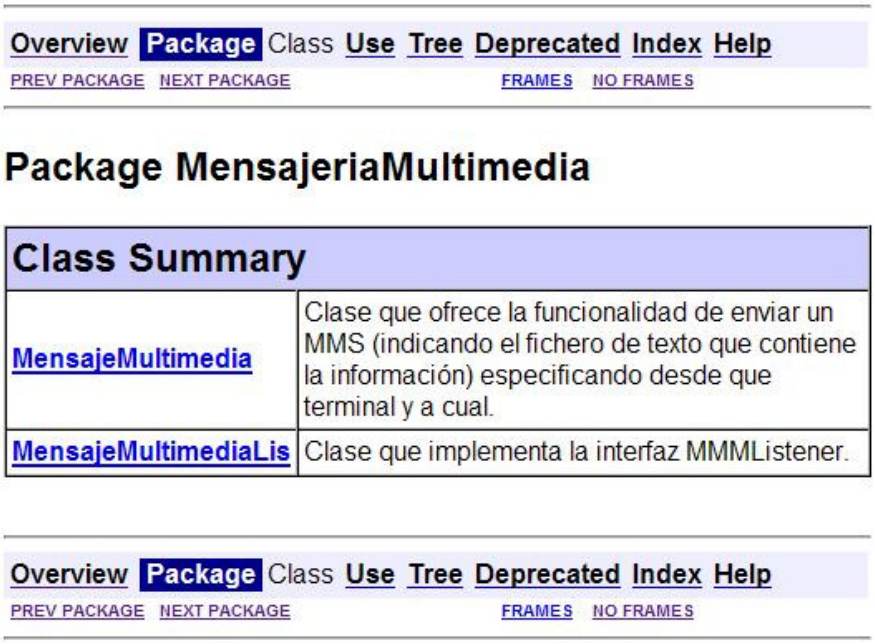
Class Summary	
EstadoUsuario	Clase que implementa los métodos necesarios para averiguar el estado de un terminal.
EstadoUsuarioLis	Clase que implementa la interfaz UserStatusListener.

Overview Package Class Use Tree Deprecated Index Help

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Figura 68. ***Paquete EstadoTerminal***

Paquete MensajeriaMultimedia



Overview Package Class Use Tree Deprecated Index Help

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Package MensajeriaMultimedia

Class Summary	
MensajeMultimedia	Clase que ofrece la funcionalidad de enviar un MMS (indicando el fichero de texto que contiene la información) especificando desde que terminal y a cual.
MensajeMultimediaLis	Clase que implementa la interfaz MMMListener.

Overview Package Class Use Tree Deprecated Index Help

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Figura 69. Paquete MensajeriaMultimedia