# *Capítulo III*

# *Desarrollo del Proyecto*

# 1. Introducción

En este capítulo se va a describir todo el trabajo realizado durante este proyecto. A modo de introducción se exponen los puntos más importantes que se van a desarrollar posteriormente:

- **Base de Datos.** Para el funcionamiento de la aplicación es necesario que se almacene información acerca de los pacientes, cuidadores, ambulancias, etc. por lo que se pensó en crear una pequeña base de datos, y que la aplicación la gestionara.
- **Configuración simulador y creación del entorno de simulación.** El simulador venía configurado con una serie de características estáticas, como un mapa propio, una serie de lugares y terminales, etc. que se han modificado para adecuarlas a nuestros requisitos. Por otro lado, para realizar cualquier simulación, hay que aportar información dinámica al simulador: como el número de abonados que van a participar (especificando nombres y números de terminal), sus movimientos y otras características. Esto se realiza mediante un fichero XML, con una serie de etiquetas predefinidas, como ya se comentó en el capítulo anterior.
- **Programación.** Se describirá aquí todo el código java desarrollado utilizando la representación en UML de las distintas clases implementadas. Se mostrarán

también las relaciones principales entre las clases. No se entrará en detalles, pues para eso se aporta todo el código comentado en los anexos.

Antes de comenzar con los apartados mencionados más arriba, se van a enumerar los SCF's que emplea la aplicación desarrollada, explicando cuál es la funcionalidad que se busca con ellos:

- **Framework (FW):** simula el armazón, pieza necesaria en este tipo de arquitectura de servicios, y con la que es necesaria establecer la comunicación inicial para poder obtener el resto de funcionalidades.
- **User Location (UL):** ofrece información acerca de las posiciones de los terminales siguiendo distintos procedimientos (cuando se solicite, periódicamente o por eventos previamente especificados); es la base de la aplicación creada, pues de esta manera nuestros pacientes están totalmente vigilados, ya que su posición es comprobada periódicamente.
- **User Status (US):** ofrece información sobre el estado de los terminales (ocupado, apagado o encendido), y al igual que el anterior, también de diversas maneras (cuando se solicite o por eventos programados). En el servicio implementado se ha utilizado para detectar cuando el terminal de un paciente se apaga (o queda fuera de cobertura), pues esto supone que no se pueda seguir comprobando su posición.
- **Call Control (CC):** ofrece todas las funcionalidades relacionadas con una llamada telefónica. Es necesaria para poder establecer criterios y poder capturar las llamadas requeridas (las llamadas son tratadas como objetos) para así realizar la actividad que en cada momento se precise: desviarla a otro número, rechazarla, reproducir mensajes pregrabados,...
- **User Interaction (UI):** simulador que ofrece la capacidad de reproducir mensajes predefinidos a través de una llamada y poder obtener respuesta del oyente (a través de la pulsación o pulsaciones de las teclas del terminal). En el servicio de Teleasistencia se ha utilizado para que los cuidadores informen a través de un menú predefinido de la situación de su paciente, cuando una alarma por posible pérdida se produce.

- **Multi-Media Messaging (MMM):** ofrece la capacidad de envío y detección de llegada de mensajes (MMS). En la aplicación creada se utiliza cuando es necesario transmitir información sin la necesidad de obtener una respuesta del destinatario de la misma. Por ejemplo, cuando un cuidador se encuentra con su paciente, el servicio de localización periódica se desactiva, si salen del radio de seguridad, pues el paciente se encuentra atendido. Cuando, el paciente se vaya a quedar sólo, el cuidador, a través de un MMS, informa a la central de que es necesario activar de nuevo dicho servicio de localización para el paciente.

# 2. Base de datos

Se han creado sólo aquellas tablas necesarias para la aplicación, y con las columnas más imprescindibles. Desde el punto de vista de la aplicación final sería necesario conocer muchos más datos de los pacientes, como por ejemplo enfermedades, medicamentos y otro tipo de información médica. También sería interesante tener almacenada más información acerca de cuidadores y ambulancias, pero, como se ha comentado más arriba, sólo se han creado aquellas tablas y columnas necesarias para hacer correr la aplicación.

La base de datos está realizada con SQL, más concretamente con MySQL. El servidor de base de datos empleado ha sido *MySQL Server 5.0*. Como se comentó en la introducción general, se ha hecho uso también de *MySQL Tools for 5.0*, que incluye la herramienta *MySQL Administrator*. Con ella el proceso de creación de bases de datos se acelera enormemente. La base de datos creada se llama ***teleasistencia*** y tienen acceso a ella los usuarios *root* y *user* (ambos tienen como contraseña *pass*). El usuario *user* es el que utiliza la aplicación para realizar consultas y actualizaciones de la base de datos cuando se está ejecutando. Para el acceso a la base de datos se ha utilizado la API JDBC, para lo cual se empleó el driver *MySQL Connector/J*, que es el driver de JDBC para MySQL. El driver lo suministra libremente *MySQL AB*, creadora del resto de software antes mencionado.

Para representar las distintas tablas que conforman la base de datos se ha hecho uso de otra herramienta del paquete *MySQL Tools*: *MySQL Workbench*. Esta herramienta permite crear base de datos de una manera gráfica. Pero lo que se ha utilizado de ellas es la capacidad de realizar ingeniería inversa: a partir de una base de datos creada, es capaz de obtener su representación gráfica. Es esta representación la que se muestra a continuación. En primer lugar se muestran las tablas *cuidadores*, *pacientes* y *servicios_contratados* y las relaciones existentes entre ellas.

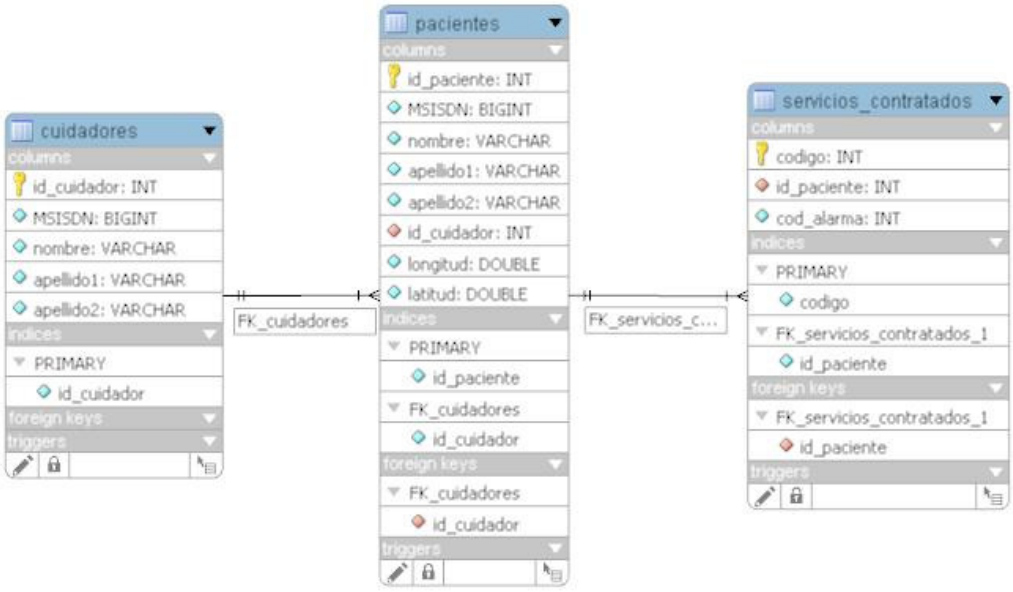


**Figura 37. Tablas pacientes, cuidadores y servicios_contratados**

La tabla *pacientes* contiene, como su nombre indica, a los pacientes. En ella se almacena el DNI, MSISDN y nombre y apellidos de cada uno de ellos. Además, se almacena el DNI del cuidador que tiene asociado, así como su longitud y latitud de referencia. Todos los pacientes almacenados en esta tabla no tienen porqué tener algún servicio contratado. Para ello se emplea la tabla *servicios_contratados*. En ella, cada entrada asocia el DNI de un paciente con el código de una alarma (el índice es un entero que se auto incrementa cuando se añade una entrada en la tabla). En la aplicación implementada se tienen dos tipos de servicios: el de localización y el de caída. Evidentemente, el DNI del paciente es una *foreign key* (relación de uno a muchos), al

igual que lo es el DNI del cuidador en la tabla *pacientes*. En la tabla cuidadores se almacena la información relativa a estos, identificándolos también mediante su DNI.

A continuación se muestran las dos tablas que restan para completar la base de datos: *estadopacloc* y *ambulancias*.


**Figura 38.** ***Tabla ambulancias***


**Figura 39.** ***Tablas estadopacloc***

En la tabla *ambulancias*, como su propio nombre indica, se almacenan las ambulancias de las que dispone el servicio. Por ahora sólo se almacenan de ellas su

MSISDN, para poder contactar con ellas (el índice es un entero que se auto incrementa cuando se añade una entrada en la tabla). En cuanto *estadopacloc*, es una tabla necesaria en tiempo de ejecución de la aplicación. Cada entrada almacena una asociación entre un MSISDN (el de un paciente) y su estado en el servicio de localización, que se representa mediante un entero. Estos estados son necesarios para la aplicación para saber si un paciente se encuentra atendido o no, porque esto condicionará la necesidad de generar una alarma según ciertas condiciones.

# 3. Configuración del simulador

En lo que se refiere a la configuración del simulador, ha sido necesario modificar ciertos archivos de configuración para adaptarlo al entorno que se buscaba. Los principales cambios se han llevado a cabo en simulador UL (*User Location*).

El software MiLife suministra mapas de las principales ciudades europeas. Sobre estos mapas, hay que definir la longitud y latitud tanto máxima como mínima, que se corresponden con las esquinas de los mismos. Por parte de España, el mapa que se aporta es el de Madrid, pero con muy poca resolución; por eso se pensó en utilizar otro de mayores dimensiones. Como había que buscar otro, pues se aprovechó para usar uno de Sevilla. Una vez conseguido el mapa en Internet, y para conseguir un mayor realismo, se decidió no usar las coordenadas por defecto que suministra el programa, pues eso supondría obtener unas distancias que en nada se corresponderían con lo que se visualizase en el mapa. Para obtener tanto la longitud como la latitud del mapa que se poseía, se recurrió al programa ***Google Earth***, de distribución gratuita. Una vez definido el escenario, y con unas coordenadas bastantes próximas a la realidad, se pasó a definir en el mapa lugares fijos, como la sede central desde donde se supone que se presta el servicio de teleasistencia como los lugares de residencia de cada uno de los pacientes. Estos lugares se especifican en el fichero *isgsimgui.properties*, y a cada uno de ellos se les asocia una imagen para que aparezca en el mapa. Estos lugares son independientes del mapa que se utilice, aparecerán en todos, por lo que los lugares de ejemplo que

especificaba el simulador se eliminaron. El resto de características de configuración apenas se han modificado.

En cuanto a la creación del entorno de simulación, se refiere a la creación del fichero XML que proporciona al simulador toda la información que necesita para llevar a cabo la simulación del servicio de telasistencia. El fichero creado, denominado también ***teleasistencia***, define los abonados (con nombres y MSISDN) en cada uno de los simuladores menores que conforman el simulador al completo, define los comportamientos en algunos de los simuladores (comportamiento en cuanto movimiento sobre el mapa en el simulador UL, comportamiento a la hora de actuar en caso de recibir una llamada en el simulador, etc.). También se definen caraterísticas técnicas de los simuladores, como características temporales de respuesta, retrasos, tipos de red, probabilidad de error en las peticiones o respuestas, etc. Se proporciona dicho archivo en los anexos, para consultarlo con mayor detalle.

# *4. Clases de la aplicación*

Como ya se ha comentado en la introducción de este capítulo, en este punto se va a describir todo el código desarrollado. Se va a abordar desde un punto de vista gráfico, es decir, no se va a entrar en describir de forma detallada el código, si no que se va a recurrir a esquemas y representaciones en UML. Para alguna duda más concreta, se aporta todo el código comentado y el Javadoc en los anexos.

En la imagen que se muestra a continuación se muestra el esquema del proyecto al completo. Todo el código fuente se encuentra en la carpeta *src*. En la carpeta *bin* se encuentran todos los archivos ejecutables y en *doc* toda la documentación (*Javadoc*) del proyecto (que también se suministra en los anexos).

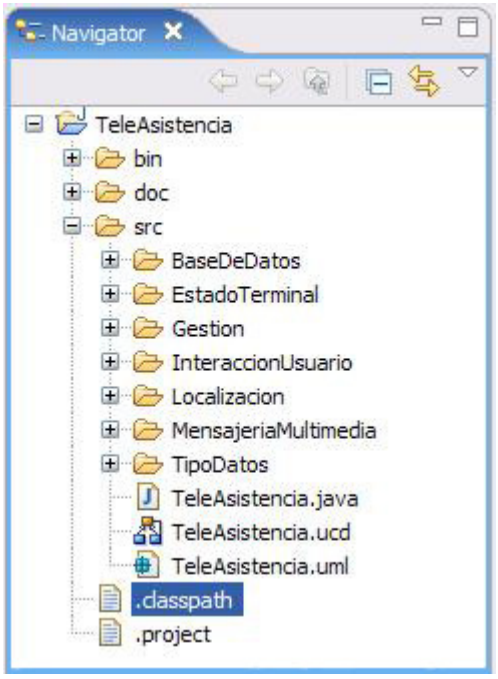


**Figura 40.** ***Esquema completo del proyecto***

La estructura que se va a seguir en este punto es la misma que la que se sigue en el proyecto. Se va a ir analizando las clases de cada paquete por separado. La clase ***TeleAsistencia***, cuya representación en UML se muestra a continuación (figura 41) no pertenece a ninguno de los paquetes creados, pues se trata de la clase principal que contiene el método Main().

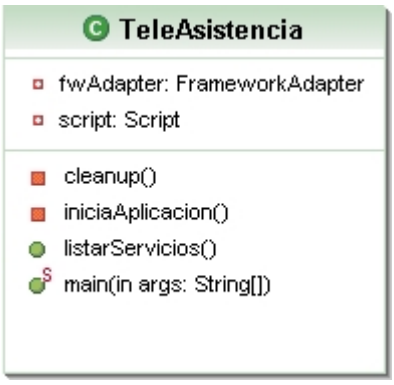


**Figura 41.** ***Clase TeleAsistencia***

Esta clase es la encargada de iniciar el servicio de TeleAsistencia. Mediante el método *iniciaAplicacion()* se identifica con el Framework (contiene el identificador de

aplicación y la contraseña) y a partir de ahí obtiene el resto de adaptadotes de servicio necesarios en la aplicación. Por último, en el método Main(), existe un bucle infinito que se encarga de procesar constantemente el vector de alarmas. Una vez iniciado el resto de adaptadores (que implica crear hilos hijos para el procesamiento del estado de los pacientes), el hilo principal, el del Main(), se duerme. Cuando alguno de estos hilos hijos crea una alarma, lo despierta, y entra en el bucle entes mencionado para el procesamiento de la alarma en cuestión.

A continuación se muestra un diagrama con las relaciones más importantes entre esta clase y las clases que se encargan de ofrecer las funcionalidades de red más importantes. Estas clases se analizarán más adelante. Podemos comprobar como podemos hacer una distinción en dos grupos de clases a las que accede *TeleAsistencia*: las que lanzan el resto de servicios que componen la aplicación y las que se encargan de la gestión (clases que definen métodos para el tratamiento de objetos específicos). Se enumeran a continuación las que lanzan el resto de funcionalidades del servicio:

***1.*** *LocalizacionUsuario*: se encarga de crear un nuevo hilo que comprueba y analiza periódicamente las posiciones de los pacientes.

***2.*** *InteraccionUsuarioLoc*: implementa una interfaz *Listener*; captura las llamadas provocadas por una alarma en el servicio de localización y realiza la interacción con los cuidadores.

***3.*** *EstadoUsuarioLis*: implementa una interfaz *Listener*; tiene como misión detectar los cambios de estado de los terminales de los pacientes (cuando un terminal se apaga o queda fuera de cobertura, se genera un evento procedente de la red, previamente configurado, y que esta clase captura y procesa).

**4.** *MensajeMultimediaLis:* implementa una interfaz *Listener*; está continuamente detectando la llegada de un mensaje a los terminales de cambio de estado del paciente o alarma por caída.

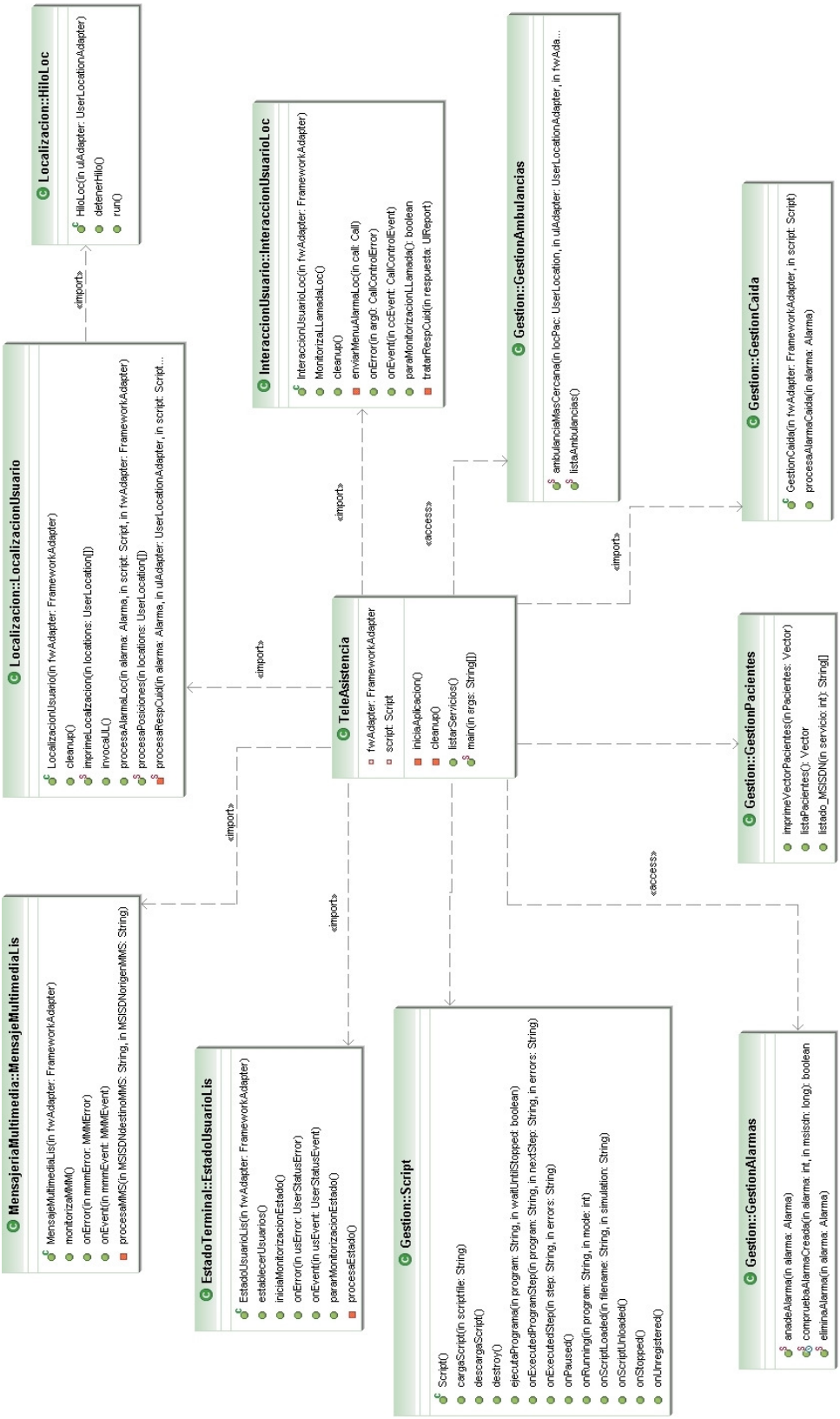


**Figura 42. Principales relaciones de la clase TeleAsistencia**

## 4.1 Paquete BaseDeDatos

Este paquete, como se ve en el esquema del paquete mostrado a continuación, sólo contiene una clase: ***BaseDeDatos***. El resto de archivos corresponden a los diagramas UML de la clase.


**Figura 43.** ***Esquema del paquete BaseDeDatos***

En la siguiente figura podemos observar la representación UML de la clase. Esta clase define los métodos necesarios para acceder a la base de datos de la aplicación. El método *reiniciaBBDDloc()* es llamado por el método Main() cuando se inicia la aplicación, y reinicia la tabla *estadopacloc*, asignando el estado normal (en el que se chequea la posición periódicamente) a todos los pacientes. Los métodos restantes de la clase sirven para actualizar datos de la base de datos (*actualiza(sentencia_SQL)*) o para realizar alguna consulta (*consulta(sentencia_SQL)*), y pueden ser llamados desde cualquier lugar.


**Figura 44.** ***Clase BBDD***

## 4.2 Paquete EstadoTerminal

El contenido de este paquete se puede visualizar en la siguiente figura. Este paquete agrupa las clases que tienen una funcionalidad común: la de averiguar cuál es el estado en el que se encuentra un terminal de abonado, aunque según la clase, se hace de una manera u otra. Este estado puede ser: REACHABLE, BUSY o NOT REACHABLE. Se definen dos clases en el paquete *EstadoTerminal*: ***EstadoUsuario*** y ***EstadoUsuarioLis***.

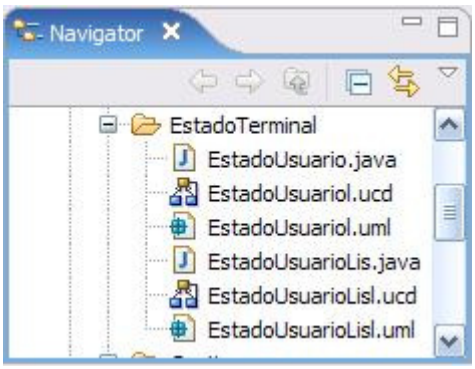


**Figura 45.** ***Esquema del paquete EstadoTerminal***

A continuación se puede observar la representación en UML (figura 46) de las clases antes mencionadas. Vemos como *EstadoUsuarioLis* implementa una interfaz *Listener* y accede a *EstadoUsuario*.

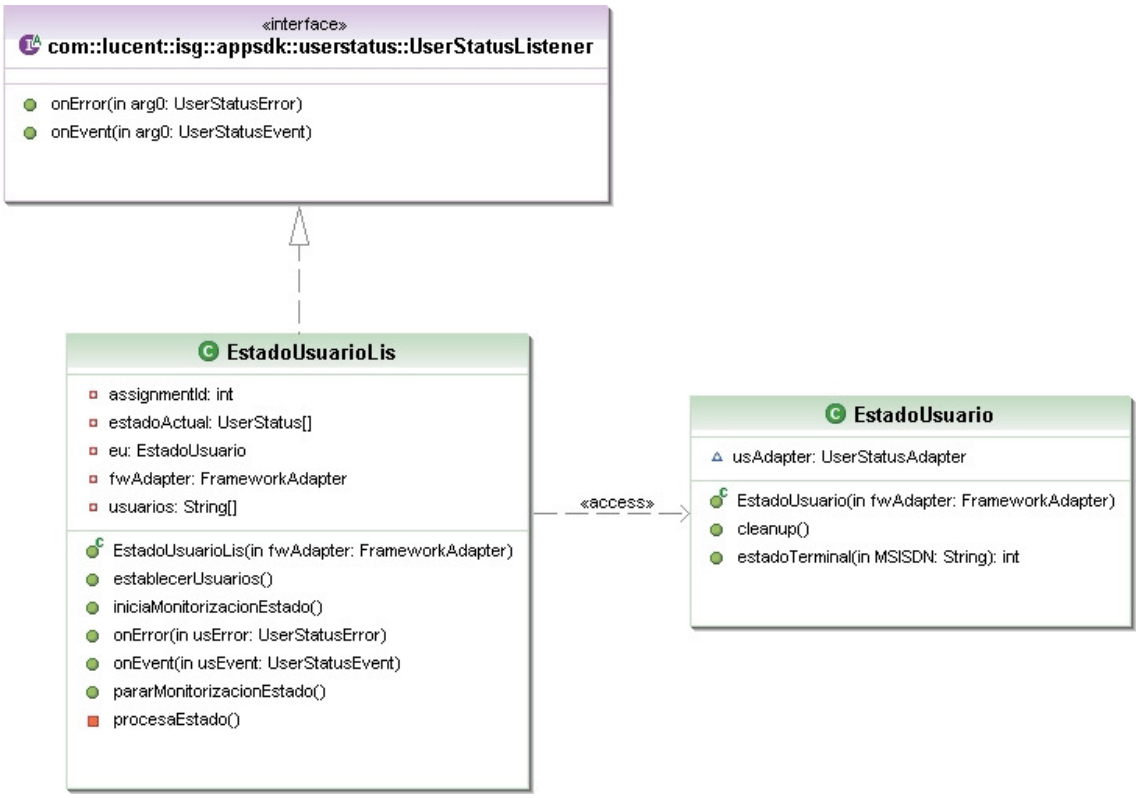


**Figura 46.** ***Clases EstadoUsuarioLis y EstadoUsuario***

La diferencia entre ambas clases radica en que *EstadoUsuarioLis*, al implementar la interfaz *Listener* correspondiente (suministrada por las *Convenience Classes* de MiLife) está continuamente 'escuchando' el estado de los terminales que se le especifique, en nuestro caso, los de los pacientes. Con el método *establecerUsuarios()* le especificamos los números de abonado (MSISDN) que queremos monitorizar y con el método *iniciaMonitorizacionEstado()* informamos a la red y solicitamos el inicio de la escucha. Cuando algún terminal de los especificados cambie su estado, la red lo notificará mediante un evento, que será capturado por el método *onEvent()*, y se dará paso a *procesaEstado()*, que tomará las medidas oportunas. Mientras que *EstadoUsuario()* ofrece la funcionalidad de usar el método *estadoTerminal(MSISDN)*, que devuelve el estado en el que se encuentra el terminal especificado como parámetro. El uso que hace la clase *Listener* de *EstadoUsuario* es la de usar el constructor de ésta, para no repetir código.

A la clase *EstadoUsuario* se accede desde cualquier punto de la aplicación, es decir, la usará cualquier clase que necesite de la información del estado de un terminal concreto, mientras que la clase *EstadoUsuarioLis*, la ejecuta el método Main() al iniciarse la aplicación, para que esté continuamente ejecutándose.

## 4.3 Paquete Localizacion

Este paquete almacena las clases relacionadas con el servicio de localización. Como se puede ver en el esquema del mismo, define dos clases: ***LocalizacionUsuario*** y ***HiloLoc***. El servicio de localización de pacientes, como ya se explicó en la introducción general de este documento, consiste en comprobar periódicamente cuáles son las posiciones de los pacientes y analizar qué distancia les separa de sus posiciones de referencia, que normalmente corresponderá a sus casas. Se ha establecido un intervalo de chequeo de las posiciones de 30 segundos, aunque esto es configurable. Si esas distancias superan el kilómetro, se generará una alarma por cada paciente que la haya superado.

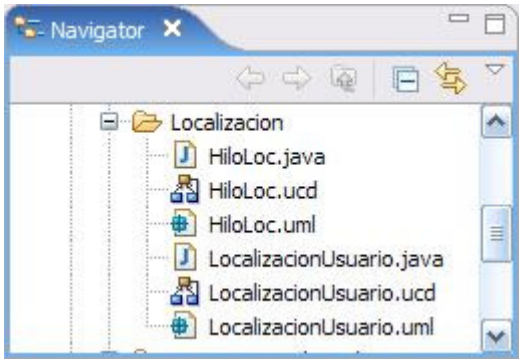


**Figura 47.** ***Esquema del paquete Localizacion***

La representación en UML de las clases se muestra más abajo. La clase *LocalizacionUsuario* la utiliza el método Main() cuando se inicia la aplicación. Mediante el método *invocaUL()*, se crea un objeto de la clase *HiloLoc*, que extiende a la clase *Thread*, creándose un nuevo hilo que es el encargado de solicitar a la red las posiciones de los pacientes que tengan contratado el servicio (método *run(),* que consiste en un bucle infinito). Cuando obtenga esas posiciones, el método

*procesaPosiciones(UserLocations)* de la clase *LocalizacionUsuario* se encargará de procesarlas.

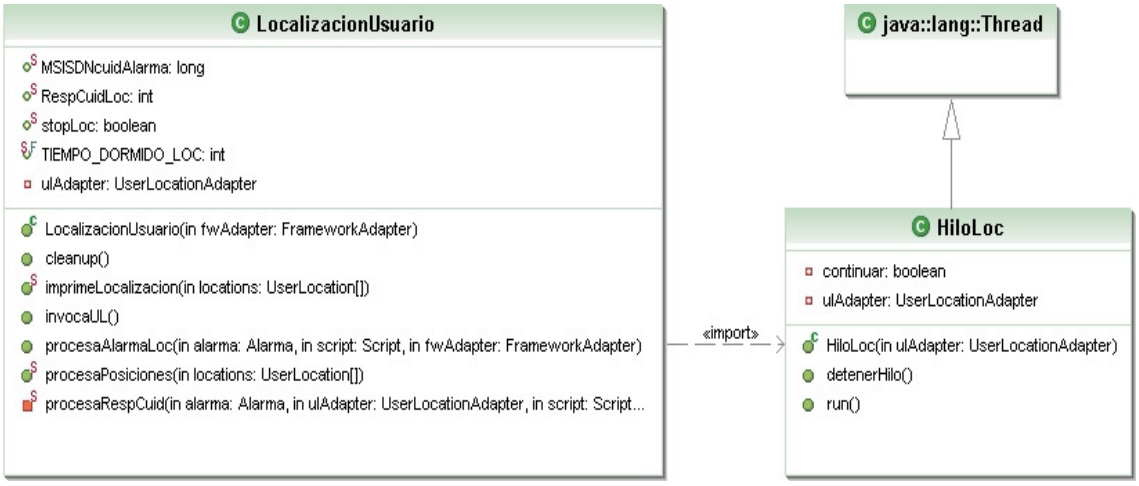


**Figura 48.** ***Clases LocalizacionUsuario e HiloLoc***

El procesamiento de esas posiciones puede provocar que se cree una alarma, lo que originará de inmediato que el hilo principal se despierte. Este hilo analizará la alarma mediante el método *procesaAlarmaLoc(...)*, que básicamente lo que hace es averiguar cuál es la distancia entre el paciente y su cuidador. Si la distancia es menor a 20 metros, se procede a enviar un mensaje al cuidador (utilizando la clase MensajeMultimedia) informándole del hecho. Si es mayor a 20 metros, se establece comunicación con el cuidador mediante una llamada telefónica. El cuidador responderá con información que será analizada con el método *procesaRespCuid(...)*, y según su respuesta, se llamará a una ambulancia para que vaya en auxilio del paciente o simplemente se cambiará su estado.

## 4.4 Paquete InteracciónUsuario

Este paquete pretende agrupar a las clases que ofrecen la funcionalidad de la red denominada *UserInteraction*. La interacción con el usuario se refiere a que sobre una llamada telefónica se reproduzcan mensajes pregrabados, normalmente menús, y el oyente pulse las teclas asociadas a la opción que desee o las elija mediante voz. Para

realizar esta actividad, es necesario 'capturar' de entre todas las llamadas que se produzcan por la interfaz OSA/Parlay aquellas que nos interesen, por lo que hay que establecer una serie de criterios (esto implica realizar una escucha continua de las llamadas que se están produciendo). Después de que esa llamada sea capturada, se iniciará el envío de los mensajes pregrabados y la posterior captura de las respuestas del oyente.

Pues bien, en la aplicación realizada, sólo en una ocasión se establece interacción con un usuario, más concretamente con los cuidadores. Como se ha comentado en el apartado anterior, cuando se produce una alarma por localización y la distancia del paciente respecto al cuidador supera los 20 metros, se establece comunicación con el cuidador correspondiente. Se supone que, en realidad, el ordenador donde esté corriendo esta aplicación estaría conectado a un teléfono o similar, y en el momento que hubiera que establecer comunicación con el cuidador se ejecutaría la orden pertinente sobre dicho aparato. En nuestro caso, puesto que se trata de una simulación, teníamos que conseguir que el teléfono del simulador asociado al servicio de teleasistencia (denominado central) realizara la llamada al cuidador en cuestión de manera automática, es decir, que la aplicación la realizara, y no manualmente desde el simulador. Para conseguir esto se ha realizado lo siguiente:

- En los scripts que cargan información en el simulador se pueden definir también programas (acciones a realizar por los abonados definidos dentro del simulador). Se han creado entonces todas las posibles llamadas que se pueden producir entra la central y los cuidadores (siendo cada llamada un programa independiente).
- Se han empleado unas clases definidas en la API de MiLife que permiten comunicarse con el simulador desde la aplicación, de tal manera que se pueden ejecutar programas definidos en los scripts, conociendo de ante mano el nombre de los ficheros XML y el de los programas a ejecutar. Se ha creado una clase que engloba a estas funcionalidades, denominada ***Script***, que pertenece al paquete de Gestión, y que se explicará más adelante.

Teniendo en cuenta lo anterior, el funcionamiento es el siguiente: cuando se produce la alarma por localización, y la distancia entre cuidador y paciente supera los 20 metros, se hace uso de los métodos definidos en la clase antes mencionada, se ejecuta una llamada desde la central al cuidador, deteniendo el hilo a continuación. Entonces esta llamada será capturada (con el método *onEvent(CallControlEvent)*) por el hilo que está a la escucha de las llamadas, es decir, por la clase *UserInteraction*, pues implementa la interfaz *CallControlListener*. Una vez capturada, es cuando se hace uso de las características de esta funcionalidad. Se reproduce un mensaje al cuidador mediante el método *enviarMenuAlarmaLoc(Call)*, informándole de que su paciente ha excedido el radio de seguridad, y se le pide que informe si conoce la situación del cuidador (puede que está acompañado de familiares y él lo sepa) o no. El cuidador pulsará una tecla, se comprobará la respuesta mediante el método *tratarRespCuid(UIReport)* (se ha tenido en cuenta que su pulsación sea errónea o fuera de tiempo y en tal caso se le avisa y se le da un nuevo intento) y se grabará para que sea analizada por el hilo principal que previamente se durmió (por lo que en este punto se vuelve a despertar).

Aunque se ha comentado más arriba que lo que se hace es ejecutar una llamada desde la central al cuidador, la llamada que se ejecuta en realidad en el simulador es en el sentido contrario: desde el cuidador a la central. Esto es debido a lo que ya se explicó en el apartado 3 del capítulo II: la interacción con el usuario en el simulador sólo se permite en el sentido del llamante, por lo que para poder establecer comunicación con los cuidadores, reproducirles un mensaje y obtener una respuesta de ellos, la llamada deben iniciarla los cuidadores. Esto en realidad no sería así, pues la interacción se establece independientemente de quién inicie la llamada.

A continuación se muestra el esquema de este paquete. Sólo contiene una clase, ***InterccionUsuarioLoc***, pues sólo se establece interacción en el servicio de localización. Dicha clase se representa también más abajo en UML.

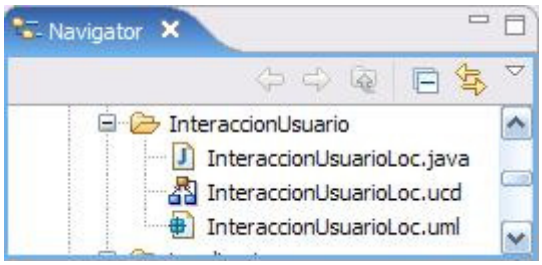


**Figura 49.** ***Esquema del paquete InteraccionUsuario***

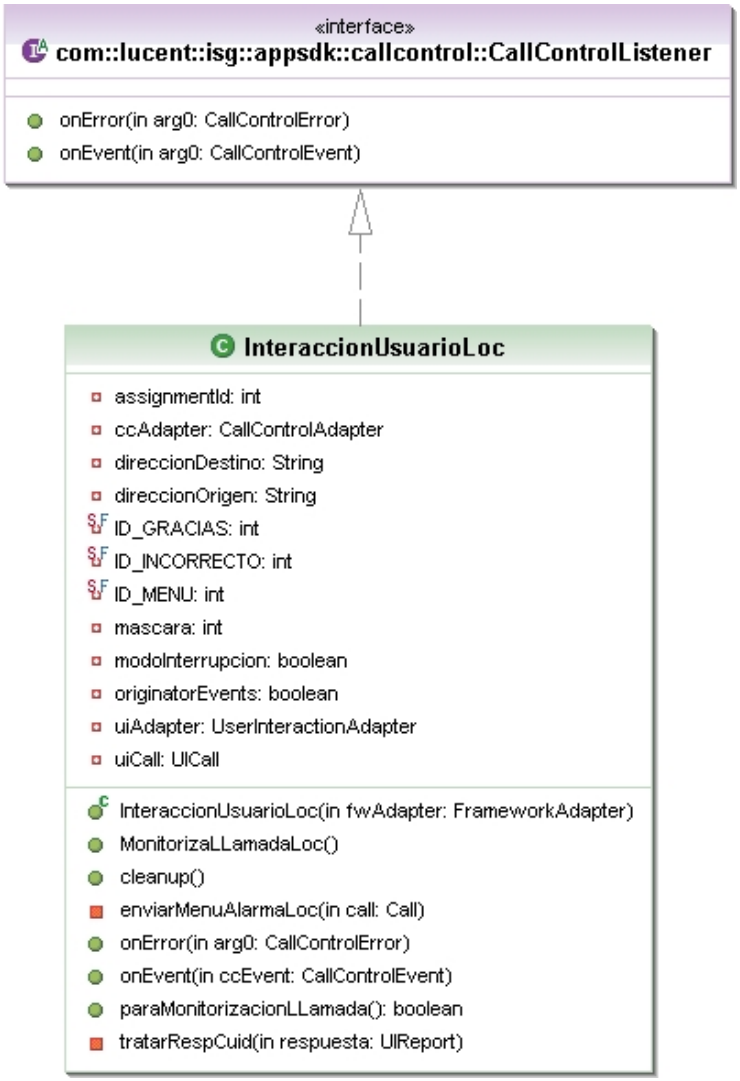


**Figura 50.** ***Clase InteraccionUsuarioLoc***

## 4.5 Paquete MensajeriaMultimedia

Este paquete contiene las clases cuya funcionalidad está relacionada con el envío y recepción de mensajes multimedia. A pesar de ser mensajes multimedia, el uso que se hace de ellos es sólo para texto, aunque se podría enviar todo tipo de datos: mensajes de voz, imágenes, etc. Al igual que en el paquete *UserStatus*, se definen dos clases: ***MensajeMultimedia***, que define métodos para enviar un mensaje multimedia estableciendo las direcciones de emisor y receptor y especificando el fichero de texto que contiene la información a enviar y ***MensajeMultimediaLis***, que implementa la interfaz *MMMListener*, y cuya funcionalidad es la de estar continuamente 'escuchando' si llegan mensajes a dos terminales específicos provenientes de los pacientes o los cuidadores (los de reopción de alarmas de caída y cambios de estado de los pacientes en el servicio de localización). El contenido del paquete se muestra en la imagen siguiente.


**Figura 51.** ***Esquema del paquete MensajeriaMultimedia***

La representación en UML de ambas clases se muestra más adelante, en la figura 52. La clase *MensajeMultimediaLis* la utiliza el método Main() cuando se inicia la ejecución de la aplicación. Crea un objeto de ese tipo, y mediante el método *monitorizaMMM()* establece los criterios y solicita a la interfaz OSA/Parlay el inicio de la escucha para la recepción de mensajes multimedia según los criterios establecidos. Cuando algún mensaje cumpla los criterios, se produce un evento desde la interfaz, que la aplicación captura en forma de objeto del tipo *MMMEvent* en el método *onEvent(MMMEvent)*, y éste llamará al método *procesaMMS(...)* que analizará el mensaje y tomará las medidas oportunas (según sea una alarma por caída o un aviso para cambiar el estado de un paciente).

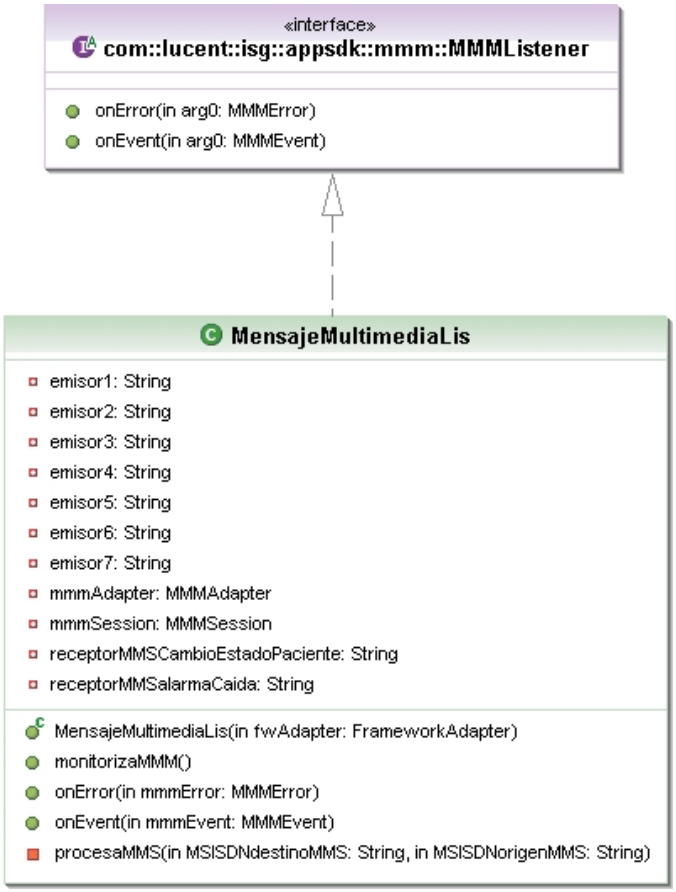


**Figura 52.** ***Clase MensajeriaMultimediaLis***

En cuanto a la clase *MensajeMultimedia*, decir que una vez que hemos construido un objeto de la misma, mediante el método *establecerDirecciones(receptores,emisor)* especificamos el número de abonado del terminal que va a enviar el MMS y el conjunto de números de los terminales que reciben dicho mensaje y con el método *enviarMensaje(...)* se produce el envío del mensaje. A este último método se le pasa el fichero de texto que contiene la información a enviar.

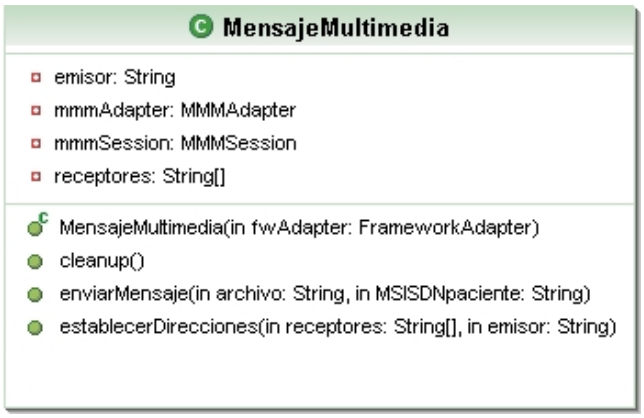


**Figura 53.** ***Clase MensajeriaMultimedia***

## 4.6 Paquete Gestion

Se muestra a continuación el contenido de este paquete. Como se puede comprobar engloba a cinco clases encargadas de gestionar distintos aspectos.

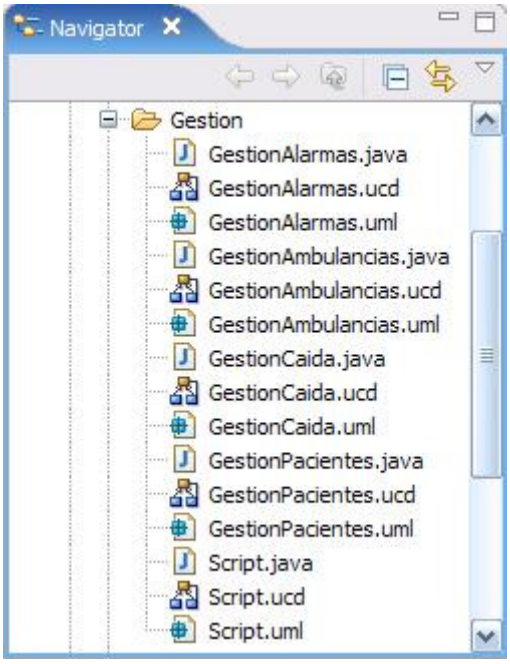


**Figura 54.** ***Esquema del paquete Gestion***

Las clases que conforman este paquete, a diferencia de lo que ocurre en los anteriores, no están asociadas por la funcionalidad de la red con la que trabajan. En este paquete se han incluido clases que ofrecen métodos para manejar los distintos tipos de

datos creados y para manejar la comunicación con el simulador. A continuación se enumeraran sus principales características así como sus cometidos:

1. ***GestionAlarmas***. Como se ha comentado en otros puntos del presente documento, cuando nos referimos a que se genera una alarma se quiere decir que se crea un objeto del tipo *Alarma*. Una vez creada esta alarma, se añade a un vector, de manera que lo que se hace es formar una cola FIFO: la primera que se cree será la primera en ser tratada, y este vector es continuamente leído por el hilo principal. Pues bien, para comprobar, añadir o eliminar alarmas del vector, hay que hacerlo de una manera sincronizada (pues son hilos hijos los que las van creando). Esta clase define estos métodos, de tal manera que sólo un hilo podrá acceder en un momento determinado al vector de alarmas.

2. ***GestionAmbulancias***. Clase que define métodos para tratar los objetos *Ambulancia*. El principal método es *ambulanciaMasCercana(...)* que devuelve el MSISDN de la ambulancia que está más cerca de la posición que se le pasa como parámetro, teniendo en cuenta también que esté libre.

3. ***GestionCaida***. Cada vez que se procese una alarma por caída de un paciente, se implementa un objeto de esta clase. Define el método *procesaAlarmaCaida()*, que lo que hace es averiguar la posición del paciente y con ella obtener cuál es la ambulancia libre y más cercana a él, empleando para ello el método *ambulanciaMasCercana()* explicado más arriba.

4. ***GestionPacientes***. Clase que define métodos para tratar los objetos *Paciente*, creados a partir de la información almacenada en la base de datos. En el comienzo de la aplicación, se lee la base de datos y se crean tantos objetos *Paciente* como pacientes tengan contratado el servicio de localización, para poder trasladar esa información al hilo que comprueba periódicamente sus posiciones.

5. ***Script***. Esta clase se ha explicado anteriormente. Utiliza un adaptador de servicio definido en la API del software MiLife utilizado que permite la comunicación con el simulador para cargar scripts, ejecutar programas, pararlos, etc. Implementa la interfaz *ScriptCallBack*. Al implementar esta interfaz se generan eventos en determinadas situaciones (cuando se inicia o se termina de ejecutar un paso, o un programa entero, cuando se completa la carga de un script, cuando se pausa la

ejecución, etc.) que son capturados por los métodos definidos por la interfaz, Esto permite estar continuamente informado del estado del simulador, y poder sincronizar la ejecución del programa con la simulación.

A continuación se representa el esquema UML de estas clases. Las relaciones que se aprecian son sólo las que existen entre las clases del paquete, que básicamente son las que tiene *GestionCaida* con el resto, puesto que ésta, al ser la que define la actuación en caso de caída, ha de tratar con objetos *Alarma* y *Ambulancia* y comunicarse con el simulador para establecer conexión telefónica con las ambulancias.

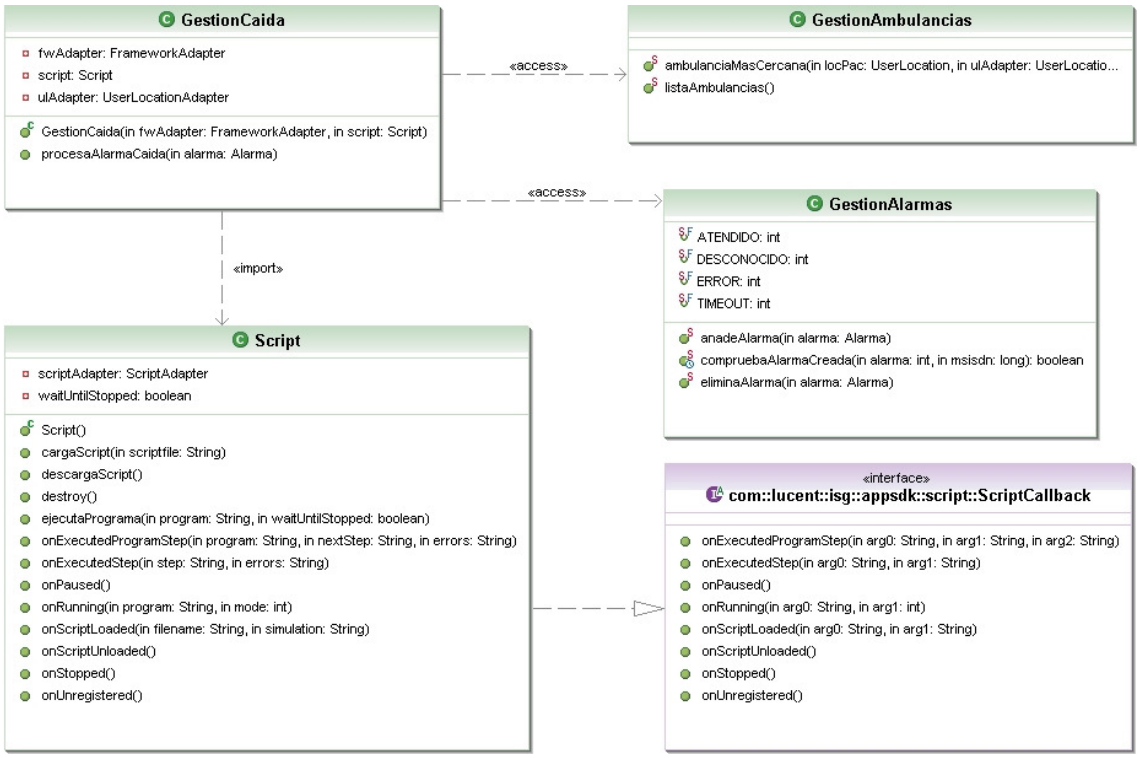


**Figura 55.** ***Clases GestionCaida, GestionAmbulancias,GestionAlarmas y Script***

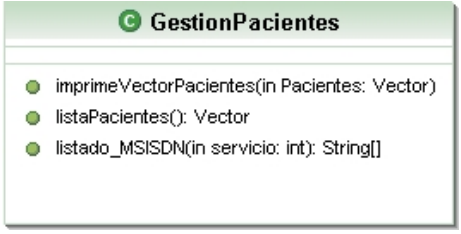


**Figura 56.** ***Clase GestionPacientes***

## 4.7 Paquete TipoDatos

En este paquete se aglutinan aquellas clases e interfaces que definen los tipos de datos así como constantes utilizadas por la aplicación. El contenido del mismo se muestra en el esquema de más abajo. Se compone de tres clases y una interfaz.

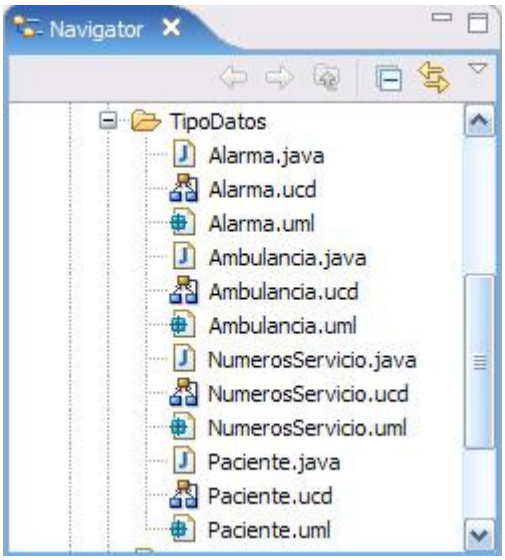


**Figura 57.** ***Esquema del paquete TipoDatos***

Se presenta a continuación la representación en UML de cada una de estas clases, comentando sus características principales. Con la clase ***Alarma*** se define el formato de las alarmas. Muchos de los campos que la componen no han sido utilizados en la aplicación desarrollada, pero se han incluido pensando en posibles ampliaciones, en las que se oferte una gama de servicios mayor y la gama de alarmas sea por tanto mucho más amplia y como consecuencia haya que establecer prioridades. En esta clase se definen también los valores constantes de los tipos de alarmas con las que trabaja la aplicación: caída y persona perdida.

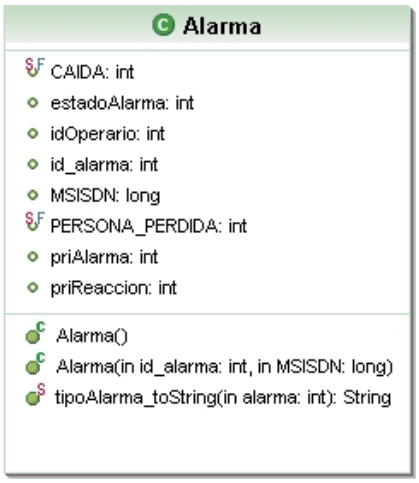


**Figura 58.** ***Clase Alarma***

En cuanto a la clase ***Paciente*** sirve para reunir en un mismo objeto toda la información necesaria de un paciente. Se usa principalmente cuando la aplicación empieza a correr, pues como se comentó anteriormente, antes de lanzar los servicios debemos de leer la base de datos para averiguar cuántos pacientes tienen contratado algún tipo de servicio. Como resultado de esa lectura, se crea un vector de objetos *Paciente*, que contiene toda la información de los pacientes (su identificación en forma de DNI y su número de abonado), así como un vector para cada uno de ellos con los servicios contratados. A partir de este momento, la aplicación tiene los datos necesarios para lanzar los servicios.

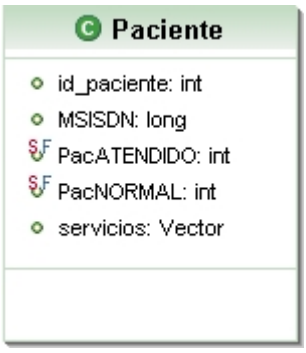


**Figura 59.** ***Clase Paciente***

Al igual que ocurre con los pacientes, cuando comienza a correr la aplicación, el método Main() crea un vector de objetos ***Ambulancia***, donde cada objeto se

corresponde con una ambulancia real (en este caso en el simulador). Para ello, el servicio tiene registradas a todas sus ambulancias en la base de datos, y la aplicación sólo tiene que realizar una consulta.

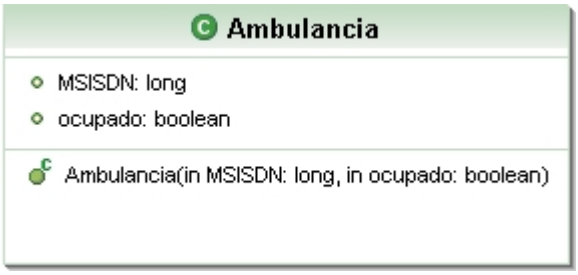


**Figura 60.** ***Clase Ambulancia***

Por último, el paquete contiene a la interfaz ***NumerosServicio***. En ella, se definen los MSISDN de los terminales que componen el servicio de teleasistencia.


**Figura 61.** ***Interfaz NumerosServicio***