- Título: Desarrollo de una herramienta software para el manejo de un teléfono móvil adaptada a personas con discapacidad física severa
- Autor: Alberto Egido Ros
- Programa de Postgrado: Máster en Ingeniería de Sistemas y Servicios Accesibles para la Sociedad de la Información
- Tutor: Juan Ignacio Godino Llorente
- Director: Dr. César Sanz Álvaro
- Tribunal:
    - Presidente: Juana Mª Gutíerrez Arriola
    - Vocal: Luis Narvarte Fernández
    - Vocal Secretario: Nicolás Sáenz Lechón
- Fecha de lectura: 13 de Julio de 2012

Índice

Índice de figuras

Índice de tablas

Resumen

El presente proyecto desarrolla una aplicación residente en un terminal móvil, que pretende proporcionar un valor añadido al actual proyecto *Localiza*, sistema de localización bajo demanda para personas con discapacidad severa.

Mediante el desarrollo de este proyecto se pretende facilitar el acceso al teléfono móvil y al ordenador a las personas con discapacidad motriz.

El objetivo final es ser capaz de controlar un teléfono móvil por medio de control remoto, mediante el uso de un ordenador personal. Para ello se establece una conexión remota entre el terminal móvil y el ordenador personal, a través del protocolo de comunicación *Bluetooth*. De este modo, a través de la aplicación móvil se transmite la información de posición de las coordenadas, proporcionada por el acelerómetro del terminal, a un servicio instalado en el ordenador que se encarga de gestionar la información recibida, y así crear las interrupciones pertinentes en el sistema operativo para mover el puntero del ratón.

Para controlar el teléfono móvil de forma remota se dispondrá de un emulador de telefonía móvil instalado en el ordenador que implemente las funciones básicas de control de llamadas. Por medio de comunicación *Bluetooth*, las acciones que realice el usuario en emulador serán invocadas en el propio terminal móvil.

Summary

The project presented develops a mobile application, which is intended to provide an added value to the already existing project *Localiza*, on-demand position system for people with severe disabilities.

This project aims to facilitate the access to the personal computer and to the mobile telephony environment for disabled people.

The main goal is to be able to control a mobile phone by remote control, using a personal computer. Thus, a remote connexion will to be established between the mobile device and the personal computer, through *Bluetooth* communication protocol. Thus, the mobile application will transmit the coordinate's position, provided by the accelerometer of the mobile device, to a *Bluetooth* service running in the personal computer. That service will be in charge of managing the information received in order to create the interruptions on the operational system for moving the mouse pointer.

The remote controlling of the mobile device is carried out using a mobile telephony emulator installed in the personal computer, which will implement the basic functionality of calling control. Using *Bluetooth* communication, the user actions done in the emulator interface will be invoked on the mobile device itself.

1 Introducción

1.1. Objetivos del Proyecto

El objetivo principal de este proyecto es colaborar con el proceso de integración de las personas con discapacidad en la nueva era de las tecnologías, mediante el desarrollo de un sistema que facilite el acceso, tanto a un ordenador como a un teléfono móvil, a aquellas personas con una discapacidad motriz de carácter severo.

El sistema se comprende de dos partes (ver esquema Figura 1). Una consiste en una aplicación para el teléfono móvil, y la otra en un servicio que se ejecutará en un ordenador.

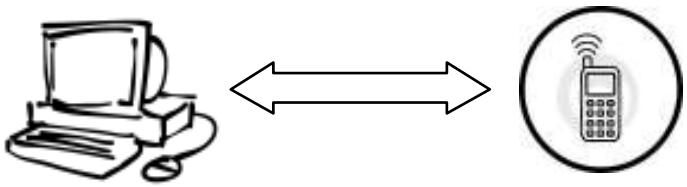

Figura 1. Esquema global del sistema

Este proyecto sólo se encarga del desarrollo del software de la aplicación móvil, llamada *"Control Remoto"*. La aplicación móvil va a enviar al ordenador, a través de un canal inalámbrico, las coordenadas del sensor de aceleración, integrado en el teléfono móvil.

El servicio que se ejecutará en el ordenador, ha sido desarrollado en el proyecto "*Herramienta de Control de Dispositivo Móvil mediante un Ordenador*" [1], y se denomina "*Ratón Localiza*". Éste va a ser el encargado de gestionar los datos enviados desde el dispositivo móvil, para así generar las interrupciones en el sistema operativo del ordenador, que permitan mover el ratón.

Una vez que se es capaz de mover el ratón, mediante el uso de un emulador de un terminal móvil instalado en el ordenador, se podrá cumplir el objetivo de controlar de forma remota las funciones básicas del móvil.

A continuación se enumeran los objetivos concernientes al desarrollo de la aplicación móvil:

- Establecer una comunicación inalámbrica entre la aplicación móvil y la aplicación del ordenador.

- Obtener la información proporcionada, por la interfaz del sensor de aceleración del teléfono móvil.

- Detectar las variaciones susceptibles al movimiento real, en la información obtenida del acelerómetro

- Enviar las coordenadas del acelerómetro, en el caso de detección de movimiento, hacia la aplicación del ordenador.

1.2. La discapacidad y sus grados

Ya que el sistema a desarrollar se encuentra dentro del ámbito de la discapacidad, en este apartado se realiza una introducción a lo que ésta representa, describiendo así su significado y los grados en que se clasifica.

La discapacidad se entiende como un concepto que evoluciona, y que resulta de la interacción entre las personas con deficiencias y las barreras debidas a la actitud y al entorno, que evitan su participación plena y efectiva en la sociedad en igualdad de condiciones con las demás.

Según la Organización Mundial de la Salud (OMS), el 15 por ciento de la población mundial se ve afectado por alguna discapacidad física, psíquica o sensorial, que dificulta su desarrollo personal y su integración social, educativa o laboral. Este porcentaje se corresponde a que 900 millones de personas padecen minusvalías de diverso grado.

Según la normativa vigente en el Estado Español, se reconocen cinco categorías o grados de discapacidad, ordenados de menor a mayor porcentaje. Cada categoría reconoce un grado de dificultad para realizar las actividades de la vida diaria, de manera que el primer grupo incluye las deficiencias permanentes que no producen discapacidad y el último grupo, las deficiencias permanentes severas que suponen, incluso, la dependencia de otras personas.

- Grado 1, discapacidad nula. Aunque la persona presente alguna discapacidad, ésta no le impide realizar las actividades de la vida diaria. La calificación de esta clase es del 0%.

- Grado 2, discapacidad leve. Existe dificultad para realizar algunas actividades de la vida diaria, pero el porcentaje de minusvalía está entre el 1% y el 24%.

- Grado 3, discapacidad moderada. Hay una gran dificultad o imposibilidad para llevar a cabo algunas actividades, aunque la persona se puede cuidar a sí misma. El grado de minusvalía está comprendido entre un 25% y un 49%.

- Grado 4, discapacidad grave. Existe dificultad para algunas actividades de autocuidado y un porcentaje de minusvalía que oscila entre un 50% y un 70%.

- Grado 5, discapacidad muy grave. Es el grado más severo. Las personas afectadas no pueden realizar por sí mismas las actividades de la vida diaria. El porcentaje de minusvalía es del 75%.

En esta clasificación se incluye la evaluación de los sistemas musculoesquelético, nervioso, cardiovascular, hematopoyético y endocrino, así como los aparatos respiratorio, digestivo, genitourinario, visual, la piel, el lenguaje, las neoplasias, el oído, la garganta y las estructuras relacionadas. No se incluye a las personas con retraso mental, ya que esta circunstancia cuenta con criterios específicos de evaluación al considerarse que las deficiencias cognitivas, por leves que sean, ocasionan siempre un cierto grado de interferencia en la realización de las actividades de la vida diaria.

Las personas con discapacidad física se encuentran con múltiples limitaciones a la hora de acceder a la tecnología, como por ejemplo a un ordenador o a un teléfono móvil. Sin embargo, actualmente existe en el mercado una amplia gama de dispositivos para facilitar el acceso a los mismos.

Hoy en día, los sistemas operativos más utilizados ya incluyen en sus prestaciones “Opciones de Accesibilidad”, que cubren algunas de las necesidades de las personas con discapacidad [2] [3].

1.3. Motivaciones

El foco de motivación de este proyecto se centra, como punto de partida, en la realidad sobre la discapacidad en España. Para ello, se presenta el último estudio realizado por el Instituto Nacional de Estadística sobre las personas discapacitadas [4], emitido en 2008 (se trata de estudios no periódicos), que refleja la panorámica actual de la discapacidad en nuestro país (ver Figura 2).

Los cambios demográficos sufridos en las últimas décadas, como el envejecimiento del país, ha supuesto una profunda transformación en la pirámide

poblacional. Uno de los posibles efectos producidos, es el aumento de las personas discapacitadas, siendo la edad un factor determinante en la aparición de este fenómeno.

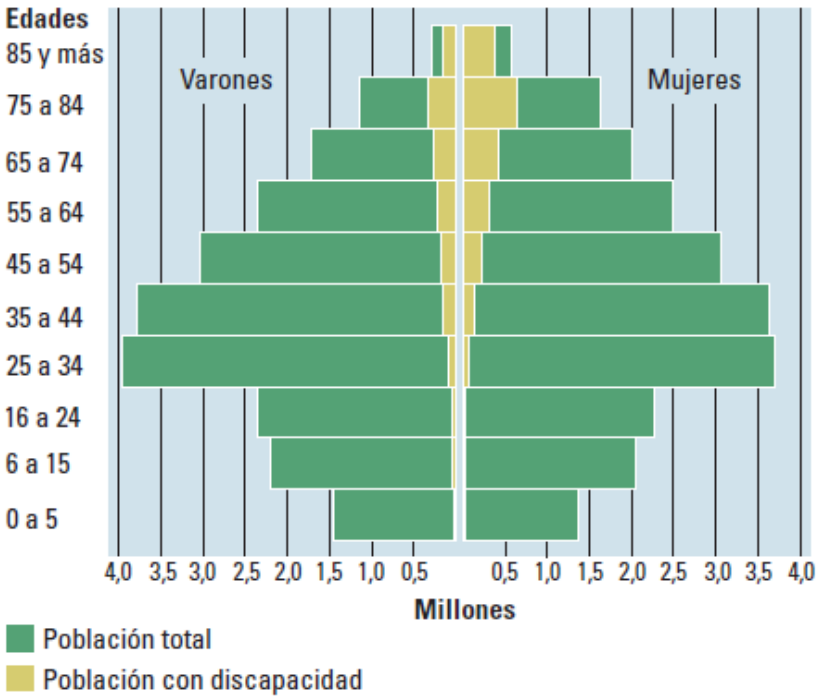

Figura 2. Pirámides de población
[4]

Conjuntamente con el aumento de la longevidad, el país ha experimentado importantes cambios sociales, que han producido que las instituciones sociales y políticas, deban ajustar sus objetivos en base a la nueva realidad, reflejándose en una demanda de mayor protección social y de un mayor apoyo a las personas que se encuentran en situación de dependencia.

Es imprescindible conocer las circunstancias de las personas discapacitadas, para entender sus necesidades, cuántos son, cuáles y con qué severidad son sus limitaciones, y si disponen de ayuda y cuidados. Por otra parte, es importante también conocer y comprender la realidad social y calidad de vida de estas personas, así como su acceso al empleo, el apoyo familiar, la discriminación o las barreras a las que se enfrentan.

En 2008 existían 3,85 millones de personas, residentes en hogares, que afirmaban tener una discapacidad o limitación, lo que supone una tasa del 85,5 por mil habitantes. El estudio realizado en [4], se centraba en personas de 6 o más años, con

una tasa del 89,7 por cada mil habitantes. En la Figura 3, se puede observar la distribución demográfica porcentual, de las personas con discapacidad en España.

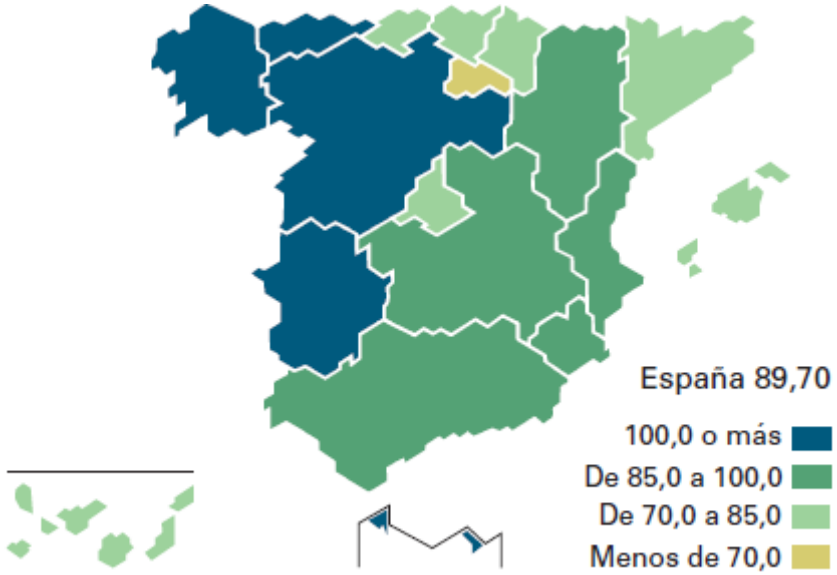

Figura 3. Personas de 6 y más años con discapacidad por mil habitantes [4]

Del estudio [4] se extrae que de estas personas, el 67 % presentan limitaciones para moverse o trasladar objetos, el 55,3 % tienen problemas relacionados con las tareas domésticas y el 48,4 % tienen problemas con las tareas del cuidado y la higiene personal.

La deficiencia que aparece con mayor frecuencia, con un 42%, es la osteoarticular, que implica un problema de en huesos y articulaciones. Sin embargo, la que causa mayor número de discapacidades por persona es la deficiencia mental.

En la Figura 4, se presenta una gráfica de discapacidades por edades y sexo. El 59,8 % de las personas discapacitadas son mujeres, mientras que por edades, la tasa de discapacidad de los hombres es ligeramente superior hasta los 44 años, y a partir de los 45 se revierte la situación, creciendo la diferencia según se aumenta la edad.

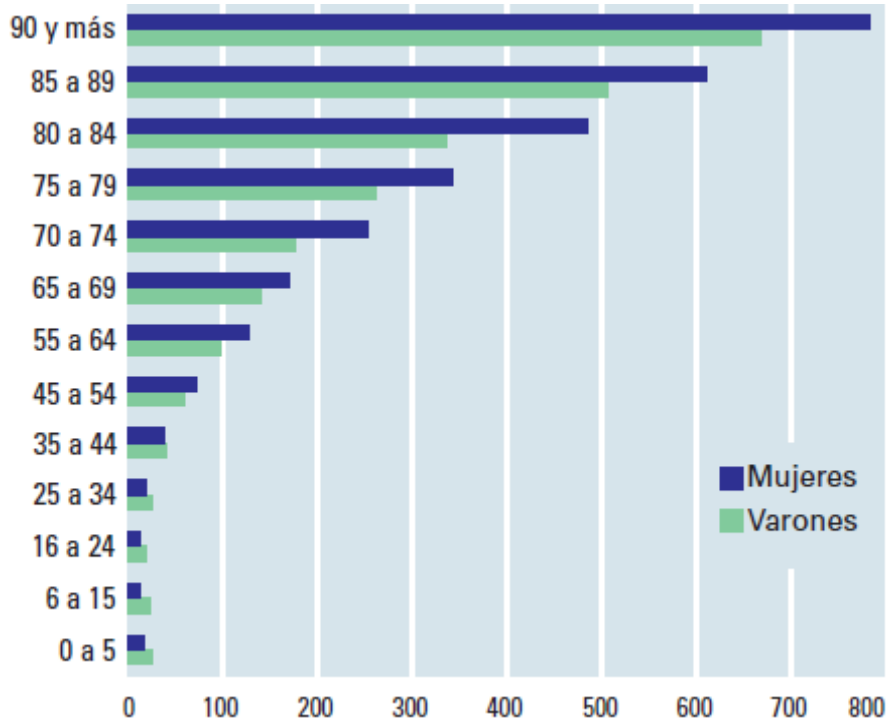

Figura 4. Estadística de discapacidad por edades y sexo [4]

En el estudio [4] se cita que más de 2,5 millones de las personas con discapacidad (71, 4 %) hacen uso de algún tipo de ayuda técnica, reciben asistencia personal, o combinan ambos tipos de asistencia (ver Figura 5). Las que acceden a más ayudas son el colectivo de las mujeres, con el 75,4% frente al colectivo masculino, con un 65,3%. Estas ayudas facilitan a disminuir la severidad de la discapacidad de las personas que acceden a ellas.

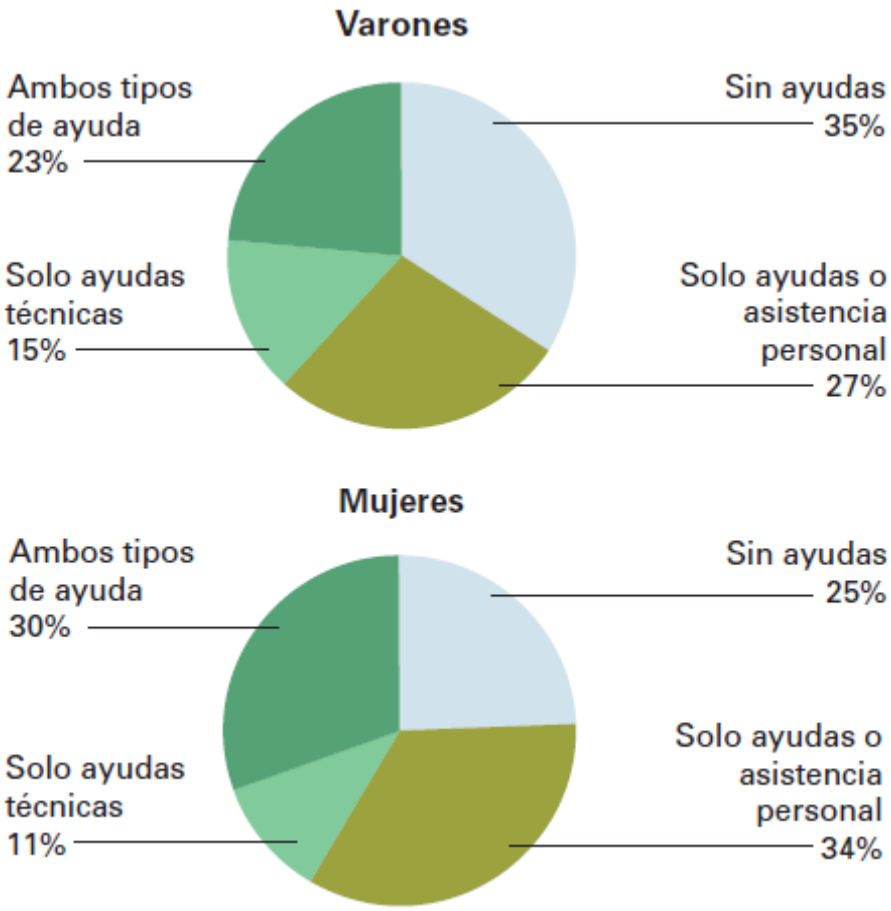

Figura 5. Tipo de ayuda recibida
[4]

Es especialmente importante el nivel de acceso a las ayudas, por parte del grupo que tiene un grado de discapacidad muy severo. Éste está formado por 1,8 millones de personas, que se ven totalmente incapacitados para realizar alguna de sus actividades sin ayuda. Dentro de este grupo, el 88,9 % recibe algún tipo de ayuda.

El estudio [4] cifra que más de seis de cada diez personas con discapacidad que reciben alguna ayuda, ven cubiertas sus necesidades. Por otro lado, las personas que no las disfrutan, expresan que tienen mayor necesidad de ayudas técnicas (27,3%) que de asistencia personal (16,9%).

En cuanto al ámbito laboral, en 2008 existía un número de 1,48 millones de personas con discapacidad en edad de trabajar. De entre las que había 419.300 tenían un puesto de trabajo. En la gráfica de la Figura 6, se observa que de entre éstas, el mayor porcentaje de trabajadores eran personas con discapacidades auditivas y visuales, con un 45,8 % y 32, 8 % respectivamente. El menor porcentaje, se encuentra en los trabajadores que tenían limitaciones de aprendizaje, aplicación de

conocimientos y desarrollo de tareas, con un 8,2%. Por otro lado, de las personas discapacitadas sin empleo, la mayoría afirman que no encuentran trabajo debido a su discapacidad.

	Personas (en miles)	% que trabajaban
Audición	291,3	42,8
Visión	295,6	32,8
Movilidad	902,7	21,6
Vida doméstica	693,3	17,2
Autocuidado	504,9	13,5
Comunicación	247,8	13,3
Interacciones y relaciones personales	280,2	11,1
Aprendizaje y aplicación de conocimientos y desarrollo de tareas	217,7	8,2

Figura 6. Trabajo según tipo de de discapacidad.2008
[4]

En otro de los campos donde también se aprecia una diferencia porcentual, es en el nivel de estudios alcanzados por la población, con respecto a la población con discapacidad. Como se ve puede apreciar en la tabla I, uno de los datos más significativos es un porcentaje muy inferior en cuanto al acceso a estudios superiores de las personas con discapacidad.

	Total Población (%)	Población con discapacidad (%)
No sabe leer o escribir	0,9	8,6
Estudios primarios incompletos	2,1	11,5
Estudios primarios o equivalentes	7,7	23,3
Estudios secundaria de primera etapa	29,0	19,2
Estudios bachillerato	15,8	11,6
Enseñanzas profesionales de grado medio o equivalentes	9,1	9,7
Enseñanzas profesionales de grado superior o equivalentes	11,4	5,6
Estudios universitarios o equivalentes	24,1	10,5

Tabla I Nivel de estudios alcanzados por la población
(Adaptado de [4])

Tras los datos analizados y extraídos del estudio [4], se puede ver que existe una realidad de la que la sociedad se ha hecho eco, donde se reclama una mayor integración de aquellas personas que sufren discapacidad.

Por lo que respecta al usuario final de este proyecto, se trata de personas con discapacidad que presenten algún tipo de limitación motriz. Como se citado anteriormente, el 67 % de las personas con discapacidad presentan dificultades para moverse o para mover objetos. De este modo, el usuario final al que está enfocado este proyecto, forma parte de un grupo mayoritario dentro de las personas con discapacidad. Por lo que se considera que la funcionalidad desarrollada en el proyecto tiene una amplía expectativa de uso dentro del sector de personas con discapacidad.

1.4. Sistema *Localiza*

El software desarrollado en este proyecto va a estar embebido dentro del sistema *Localiza* [5]. De esta forma, se pretende aportar un valor añadido al sistema *Localiza*, desarrollando una nueva funcionalidad para la aplicación *paciente.*

Localiza es un sistema desarrollado para la localización bajo demanda de personas que requieran un cierto grado de supervisión.

Esta herramienta, ofrece un servicio de localización de personas sencillo, fiable, seguro y gratuito, basado en el intercambio de mensajes de telefonía móvil que facilite la localización exacta de usuarios. Es independiente de las empresas operadoras de telefonía móvil, y no necesita de la intervención de un centro intermedio de coordinación entre el paciente y su entorno socio-familiar.

El sistema está compuesto por dos aplicaciones. La aplicación denominada *paciente*, que se ejecuta en el dispositivo móvil que se va a monitorizar, y requiere de un dispositivo GPS integrado, y la aplicación denominada *supervisor*, que se ejecuta en el dispositivo móvil del que coordina la monitorización.

El menú de opciones del que dispone el sistema permite realizar las siguientes acciones:

- Localización Puntual: el supervisor realiza una consulta sobre la localización del paciente. El paciente responde con un mensaje de texto, vía SMS, con su posición actual (latitud y longitud).

- Modo Seguimiento: este es un modo especial de localización, en el que el supervisor indica un período, marcado por la hora de inicio, hora final e intervalo en minutos, durante los cuales el paciente enviará mensajes al supervisor indicándole su posición actual.

- Visualizar en un mapa la localización del paciente.

- Ver el recorrido completo del paciente en un mapa digital.

- Avisar al supervisor cuando el paciente sobrepasa los límites de un área determinada, previamente definidos por él. Si llegara este caso, el sistema pasaría automáticamente a modo seguimiento.

- Avisar al supervisor cuando se detecta una caída del paciente.

- Avisar al supervisor en el caso de que el paciente permanezca inmóvil durante un período excesivo de tiempo.

- Avisar al supervisor en el caso de que el terminal del paciente se encuentre bajo de batería.

1.5. Fases del proyecto

Para realizar el proyecto se ha creído pertinente definir una serie de fases, para llevar a cabo el desarrollo del mismo. Las fases identificadas son las siguientes:

- Estudiar las diversas soluciones comerciales de periféricos, para discapacitados.
- Estudiar las soluciones adoptadas en otros proyectos, para la comunicación inalámbrica entre dispositivos móviles.
- Evaluar los requisitos tecnológicos para llevar a cabo el proyecto.
- Seleccionar la arquitectura más adecuada en función de los requisitos requeridos.
- Analizar las tecnologías disponibles según la arquitectura a desarrollar.
- Diseñar e implementar la solución adoptada.

2 Estado de la cuestión

2.1. Introducción

En este capítulo se presenta el estado de la cuestión que engloba el desarrollo de este proyecto. Para ello se han realizado una serie de estudios en diferentes áreas, como se detallan a continuación:

- Estudio de las soluciones tecnológicas para personas con discapacidad, focalizando en los sistemas de acceso al ordenador.
- Estudio de las soluciones disponibles, hoy en día en el mercado, para la implementación de ratones inalámbricos.
- Estudio de las soluciones desarrolladas en otros proyectos, que abordan la implementación de la comunicación entre dispositivos inalámbricos.

2.2. Tecnología para discapacitados

Los avances tecnológicos de ayuda a las personas con discapacidad se pueden agrupar según las siguientes categorías:

- Sistemas alternativos y aumentativos de acceso a la información: son ayudas para personas con discapacidad visual y/o auditiva, como son las tecnologías del habla, los sistemas multimedia interactivos, las comunicaciones avanzadas y la rehabilitación cognitiva.
- Sistemas de acceso: son interfaces adaptativas que permiten a las personas con una discapacidad física o sensorial utilizar un ordenador, como por ejemplo los sintetizadores braille, los teclados de conceptos y las pantallas táctiles.
- Sistemas alternativos y aumentativos de comunicación: son sistemas para personas que por su discapacidad no pueden acceder a un código verbal-oral de comunicación
- Sistemas de movilidad: son aquellos sistemas relacionados a la movilidad personal y las barreras arquitectónicas, como son los brazos o soportes articulados, los conmutadores adosados a sillas de ruedas, los emuladores de ratones, las varillas o los micro-robots.

- Sistemas de control de entornos: son los sistemas que, con fines comunicativos, permiten la manipulación de dispositivos que ayudan a controlar un entorno. Se dividen en dos tipos:
    - Sistemas de control ambiental: son interfaces que permiten controlar dispositivos de uso doméstico, para personas con discapacidad motora.
    - Sistemas de realidad virtual: son dispositivos como los guantes sensitivos, los posicionadores en el espacio o las gafas virtuales.

2.2.1. Sistemas de Acceso

El proyecto presentado se enmarca dentro de los sistemas de acceso, por ello se van a describir algunas de las soluciones que existen hoy en día, para facilitar el acceso a un ordenador a las personas con discapacidad.

Para personas con dificultades en el manejo del ratón existen diversas adaptaciones, caso de "joysticks", o el denominado *Headmaster*, una especie de casco en el que los movimientos de la cabeza sustituyen a los de un ratón. En el caso de personas que presentan dificultades para manejar el teclado, existen teclados adaptados o incluso sistemas de teclado virtual en pantalla, como el programa *WiVik* [6], para el entorno Windows®[1], donde el usuario selecciona las letras y los comandos del teclado, en pantalla, mediante el ratón.

También existen las pantallas táctiles que vienen preparadas para emular la función del ratón (ver Figura 7). De esta forma no se necesitan programas o configuraciones especiales, sino que cualquier entorno o programa donde funcione el ratón convencional también será accesible desde la pantalla táctil. La pantalla táctil puede venir incorporada en el monitor o ser independiente. Cuando es independiente se debe colocar superpuesta a la pantalla del monitor, de forma similar a un filtro. El usuario toca o arrastra su dedo sobre la pantalla táctil para mover el puntero del ratón. El clic se realiza al dar un golpe suave sobre la pantalla táctil. El doble clic se lleva a cabo con dos golpes, y el arrastre ejerciendo mayor presión y deslizando a la vez el dedo.

[1] Windows® es marca registrada o marca comercial de Microsoft Corporation en los Estados Unidos y/o en otros países

Figura 7. Pantalla táctil
[7]

Por otra parte, se encuentran los sistemas de síntesis de voz que hacen posible que las personas con dificultades en el habla, puedan comunicarse por teléfono al traducir el ordenador el texto escrito en voz. Además, también es posible manejar todo el entorno de un ordenador gracias a programas como *Dragon NaturallySpeaking* [8], que permiten manejar con la voz todo el entorno Windows.

2.2.1.1. Adaptaciones del ratón para discapacitados

Son múltiples las posibilidades que existen (para las personas con una limitación motriz) que permiten disfrutar de mayor accesibilidad al ratón de un ordenador.

Se pueden encontrar soluciones integradas dentro del propio sistema operativo, como por ejemplo el teclado numérico, que se puede configurar para emular el ratón y acceder a las funciones de éste, desde el propio teclado. Otra solución, es configurar el propio ratón, de manera que modificando algún parámetro facilite el manejo del dispositivo, como pudiera ser la velocidad del puntero, la velocidad del doble clic o la programación de funciones en los botones del ratón.

Por otro lado, entre las soluciones hardware, existe una amplia variedad de dispositivos ya comercializados para controlar el puntero del ratón. Entre ellos cabe destacar los que se enumeran a continuación:

- **Ratón de bola**: Es un ratón estático. Tiene los dos o tres botones del ratón convencional y una bola integrada con la que se pueden controlar los desplazamientos del cursor. Algunos modelos permiten, mediante una sencilla adaptación y un soporte adecuado, su utilización con la barbilla.

Figura 8. Ratón de bola
[9]

- **Ratón de joystick**: Son parecidos al joystick de una silla de ruedas eléctrica. Algunos modelos pueden manejarse o están diseñados específicamente para controlarlo con la cabeza o con la boca.

Figura 9. Ratón de joystick
[9]

Figura 10. Ratón de joystick para ser controlado con el mentón
[9]

- **Ratón de cabeza**: Consiste en un dispositivo tipo diadema que el usuario se coloca en la cabeza, mediante el cual con ligeros movimientos de cabeza, se consigue el desplazamiento del cursor del ratón en la pantalla.

Figura 11. Ratón de cabeza
[10]

- **Ratón tipo orbitrack**: Es un ratón que dispone de un anillo sensible al tacto que permite, con una muy ligera presión, escoger la dirección de movimiento del ratón. Orbitrack está recomendado para aquellos usuarios con problemas de motricidad en las extremidades superiores, especialmente para aquellos con buena movilidad en algún dedo y con muy poca movilidad y/o fuerza en las manos.

Figura 12. Ratón tipo orbitrack
[9]

- **Ratón contralado por la mirada***:* Movimiento del mouse mediante el seguimiento de la mirada. Resulta ideal para personas con movilidad muy limitada. La ventaja que presenta respecto a otros sistemas, es que reduce el esfuerzo que tiene que realizar el usuario para acceder al ordenador. El usuario sólo tiene que desplazar los ojos por la pantalla, de esta forma, el movimiento de la mirada actúa como ratón y el mecanismo se activa cuando la vista permanece fija en un punto durante un determinado periodo de tiempo. La gran desventaja que tiene esta solución es el precio de los dispositivos, que se venden por más de 6.000 €.

- **Emuladores de ratón:** Permiten emular las funciones del ratón estándar, pero con terminales adaptados a los usuarios. Los más utilizados son los de "pulsadores" que vienen acompañados de un software instalable.

Figura 13. Ratón controlado por pulsadores
[9]

2.1. El ratón inalámbrico

Vistos algunos de los dispositivos que existen para la adaptación del ratón hacia personas con discapacidad, en esta sección se van a analizar las tecnologías de comunicación inalámbricas disponibles para los ratones comercializados hoy en día. Es de suma importancia conocer con qué tecnologías se está trabajando, debido a que se va a implementar una de ellas en el desarrollo de este proyecto.

Existen tres tecnologías posibles para establecer una conexión inalámbrica de un ratón con un ordenador. Las dos opciones más comercializadas actualmente son las tecnologías *Bluetooth*®[2] [11] y *Wi-Fi*®[3] [12], si bien existe una tercera ya obsoleta, como son los sensores infrarrojos.

2.1.1. Ratón *Wi-Fi* Vs Ratón *Bluetooth*

Dado que *Wi-Fi* y *Bluetooth* son las dos tecnologías empleadas en los ratones inalámbricos, en esta sección se describen las diferencias que existen entre ellas.

La tecnología *Bluetooth* ha sido denominada como el equivalente inalámbrico de la conectividad USB, mientras que la tecnología *Wi-Fi* lo es a la red Ethernet. Por otro lado, *Bluetooth* ofrece una mayor versatilidad, mientras que *Wi-Fi* está orientada

[2]Bluetooth® es marca registrada o marca comercial de Bluetooth SIG, Inc.
[3]Wi-Fi® es marca registrada o marca comercial de Wireless Fidelity y Alliance, Inc.

a una transferencia de datos de un mayor volumen. En cuanto a la cobertura ofrecida, *Bluetooth* funciona en un rango de 50 a 250 metros y *Wi-Fi* funciona en un rango de 300 a 500 metros.

Como semejanza, cabe destacar que ambas operan en el mismo rango de frecuencia, 2,4 Ghz, y proporcionan una conectividad rápida y apropiada para la transferencia de datos de un ratón inalámbrico.

Una de las diferencias importantes entre ambas tecnologías viene determinada por el consumo, donde *Bluetooth* consume cinco veces menos que una conexión *Wi-Fi*. Por otro lado, en cuanto a infraestructura se refiere, los ratones *Wi-Fi* necesitan de un receptor inalámbrico, que suele conectarse vía USB al ordenador, y actúa como transcodificador de los comandos que manda el ratón. Los ratones *Bluetooth* no requieren de dicho receptor externo, sino que hacen uso del propio soporte *Bluetooth* del ordenador, dejando libre el puerto USB para otros posibles usos.

Además, *Bluetooth* posee la ventaja de simplificar el descubrimiento y configuración de los dispositivos, donde éstos pueden indicar a otros los servicios que ofrecen, lo que redunda en la accesibilidad de los mismos sin un control explícito de direcciones de red, permisos y otros aspectos típicos de redes comunes.

Por los motivos expuestos, resulta más eficiente y transparente para el usuario final la utilización de la tecnología *Bluetooth*.

2.1. Soluciones de comunicación entre aplicaciones para dispositivos *Bluetooth*

En este apartado, se describe el estudio realizado sobre proyectos existentes, que actualmente implementan una solución software para la interacción de dispositivos, a través del protocolo de comunicaciones *Bluetooth*.

2.1.1. Proyecto Bluelight

El proyecto Bluelight fue desarrollado en la facultad de ingenierías de la Universidad Antonio José Camacho, de Santiago de Cali, en julio de 2009 [13]. Bluelight implementa un sistema para el control de intensidad de luz de lámparas incandescentes.

Según se ve en la Figura 14, el modelo de hardware utilizado consta de un módulo *Bluetooth* y un microcontrolador. El microcontrolador es configurable por control remoto a través una aplicación móvil. De este modo envía la información hacia el microcontrolador a través del modulo *Bluetooth* del hardware.

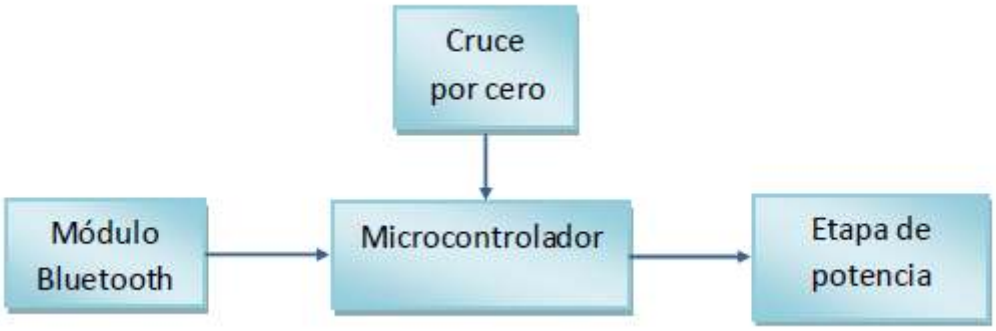

Figura 14. Diagrama de bloques del hardware Bluelight
Fuente: [13]

El sistema Bluelight permite el control de las luces de una vivienda, mediante el uso de un teléfono móvil. El diseño software desarrollado, que se puede ver en el diagrama de bloques de la Figura 15, ha sido implementado a través de la tecnología *J2ME*® (actualmente conocida como *Java*® *ME*) [4]. El modelo de diseño utilizado es el cliente-servidor. El cliente *Bluetooth* está implementado por la aplicación móvil y el servidor por el módulo *Bluetooth* del hardware (ver Figura 13).

Figura 15. Diagrama de bloques del software Bluelight V2.0
[13]

Como justificación de la elección de dicha tecnología, el autor argumenta que *J2ME* es la tecnología más extendida, para este tipo de desarrollo ya que está integrada en la gran mayoría de teléfonos móviles del mercado.

[4] Java®, J2ME® y otros productos Java son marcas comerciales o marcas registradas. de Sun Microsystems, Inc. en los Estados Unidos y en otros países.

2.1.2. Desarrollo de una aplicación de comunicación Bluetooth entre una PDA y sensores

El proyecto que se detalla a continuación, fue desarrollado en la Universidad Carlos III de Madrid en julio de 2009 [14]. En este proyecto se implementó una aplicación para estudiar los movimientos que realiza un ser humano en un escenario determinado. Se marca como objetivo poder registrar los movimientos de un cuerpo sobre un escenario móvil.

Para su realización, se proveyó de una red de sensores de aceleración distribuidos por el cuerpo de una persona, y de sensores *RFID* para registrar los objetos que toca el sujeto.

Los sensores se encargan de enviar la información, a través de *Bluetooth*, a una aplicación móvil instalada en una *PDA*, que centraliza y envía periódicamente los datos almacenados a un ordenador personal, que contiene una aplicación que se encarga de generar los informes acerca del sujeto en cuestión.

Para determinar que plataforma de programación se debía utilizar, el proyecto llevó a cabo un estudio sobre los diferentes lenguajes de programación con *Bluetooth* para un dispositivo *PDA*. En el estudio se evaluaron tres posibles soluciones, *Java*, *C/C++* y *Python®*[5].

Java se descartó debido a que el dispositivo *PDA* utilizado no disponía de la maquina virtual de *Java*, y además la versión de la maquina virtual disponible compatible con la *PDA*, no incluía las librerías de *Java* necesarias para el manejo de *Bluetooth*.

El lenguaje escogido finalmente, fue *C* por delante de *Python*. Como justificación, el autor argumenta que el lenguaje de programación C es el más habitual y más cómodo, ya que se le facilita al desarrollador un entorno de programación, junto con las librerías necesarias para el desarrollo de aplicaciones para dispositivos *Palm OS®*[6].

[5] Python® es marca registrada o marca comercial de Python Software Foundation.

[6] Palm OS® es marca registrada o marca comercial de Palm, Inc.

2.1.3. Proyecto Ulfsark

El proyecto Ulfsark fue desarrollado en la Escuela Técnica Superior de Ingenierías Informáticas y Telecomunicación de la Universidad de Granada, en el curso 2005/2006 [15]. Ulfsark consiste en una plataforma que pretende facilitar la comunicación entre dispositivos *Bluetooth*, como por ejemplo ordenadores, teléfonos móviles, PDAs, etc.

Ulfsark es un protocolo en sí, que permite la creación de aplicaciones móviles que usan *Bluetooth* para comunicarse entre ellas. Como se ve en la Figura 16, Ulfsark actúa como una capa intermedia entre el API de *Bluetooth* y la aplicación móvil.

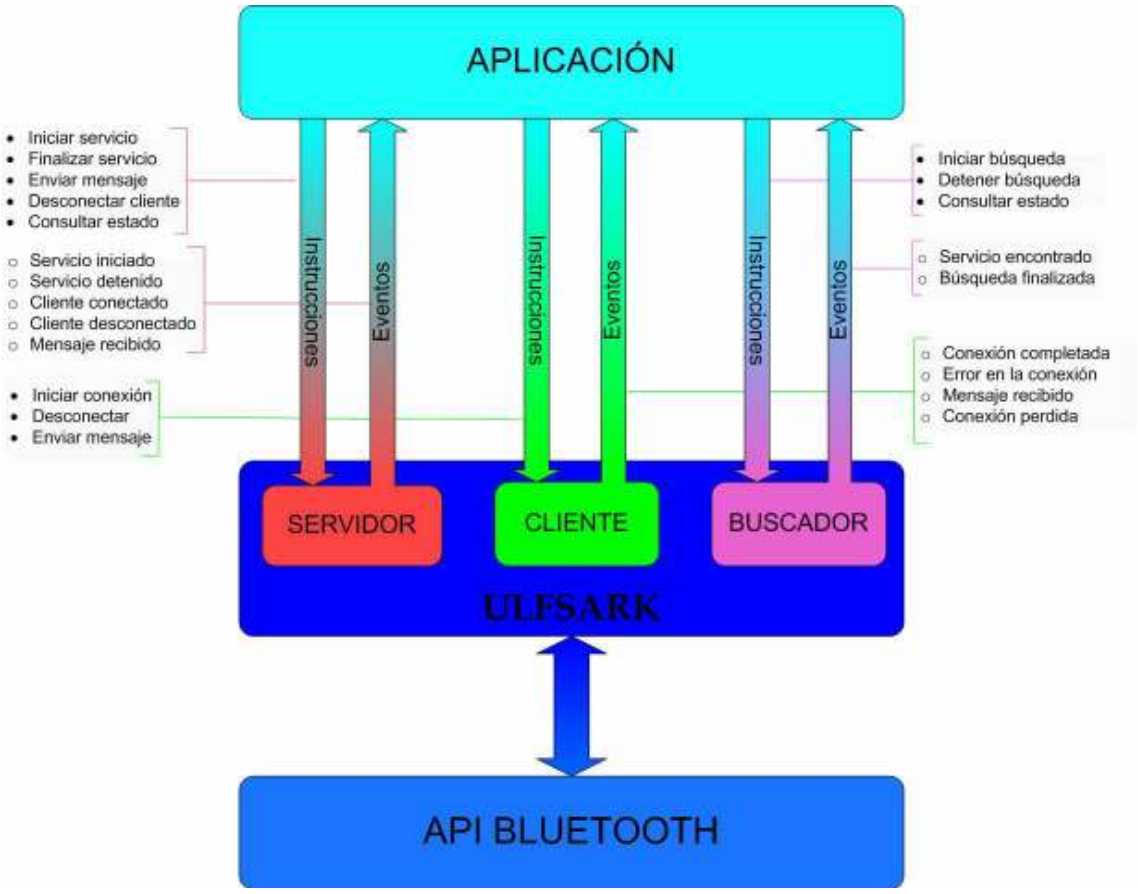

Figura 16. Diagrama de actividad protocolo Ulfsark
[15]

La plataforma seleccionada para desarrollar el proyecto fue *J2ME*, ya que según argumenta el autor, la plataforma *Java* es de muy fácil manejo y ampliamente extendida en la gran mayoría de dispositivos móviles.

De este modo, Ulfsark crea una API que enmascara la propia API de *Java* para *Bluetooth*, con la única intención de simplificar la ya existente, para facilitar su uso a desarrolladores no expertos, teniendo utilidad únicamente desde el punto de vista académico.

2.1.4. Pasarela Bluetooth/GPRS para dispositivos móviles

El proyecto que se describe a continuación, fue desarrollado en la Escuela Técnica Superior de Ingeniería de Telecomunicación de la Universidad de Málaga en junio de 2007 [16]. El proyecto, enmarcado dentro del ámbito de la *e-health*, desarrolló un software encargado de centralizar la información de monitorización procedente de varios dispositivos electrónicos, portados por un paciente.

El desarrollo del proyecto formó parte de un proyecto global (ver Figura 15), que complementa al sistema desarrollado en el proyecto que tiene como título "Desarrollo de una aplicación de telemedicina para el acceso remoto a bioseñales desde dispositivos móviles".

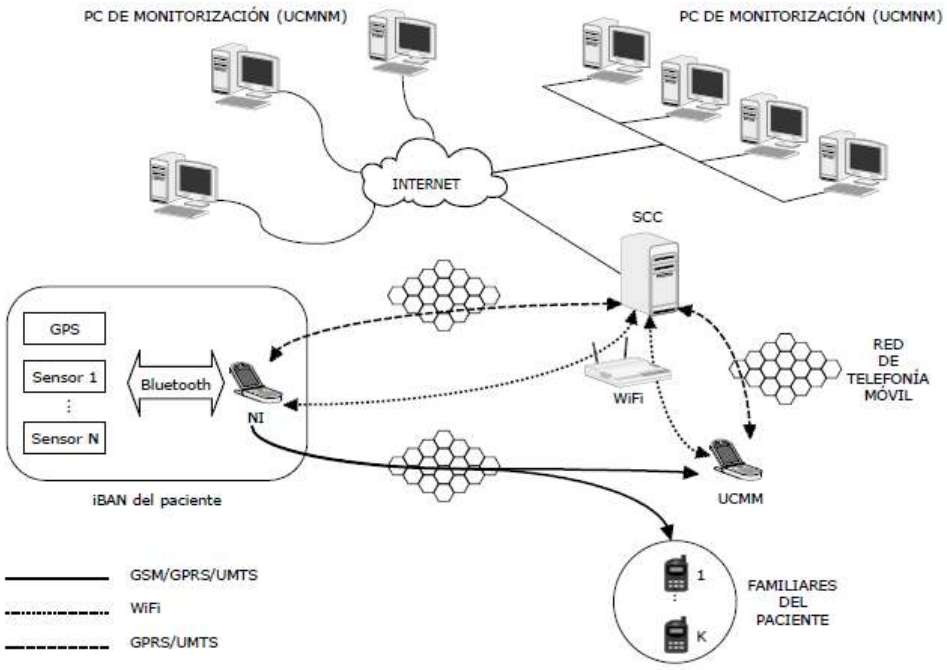

Figura 17. Sistema de monitorización de pacientes
[16]

Según se ve en el esquema presentado en la Figura 17, el sistema global está formado por los siguientes elementos:

1. EL *SCC* (Servidor de Control Central): se trata de un servidor que centraliza la información de los pacientes monitorizados, el cual permite el acceso a dicha información vía Internet, bien desde un dispositivo móvil, o bien desde un ordenador.

2. La *iBAN* (intelligent Body Area Network): es un sistema que se compone de sensores que monitorizan al paciente, y de un nodo central denominado *NI* (Nodo Inteligente), que recoge toda la información a través de *Bluetooth*. El nodo central es el encargado de enviar la información del paciente al *SCC*, y también de enviar alarmas hacia la *UCMM* y a los terminales de los familiares del paciente.

3. La *UCMM*, es un terminal móvil que es capaz de comunicarse con el *SCC* y recibir alarmas por parte del NI.

En este proyecto sólo se desarrollo el software de control de la *iBAN* del paciente.

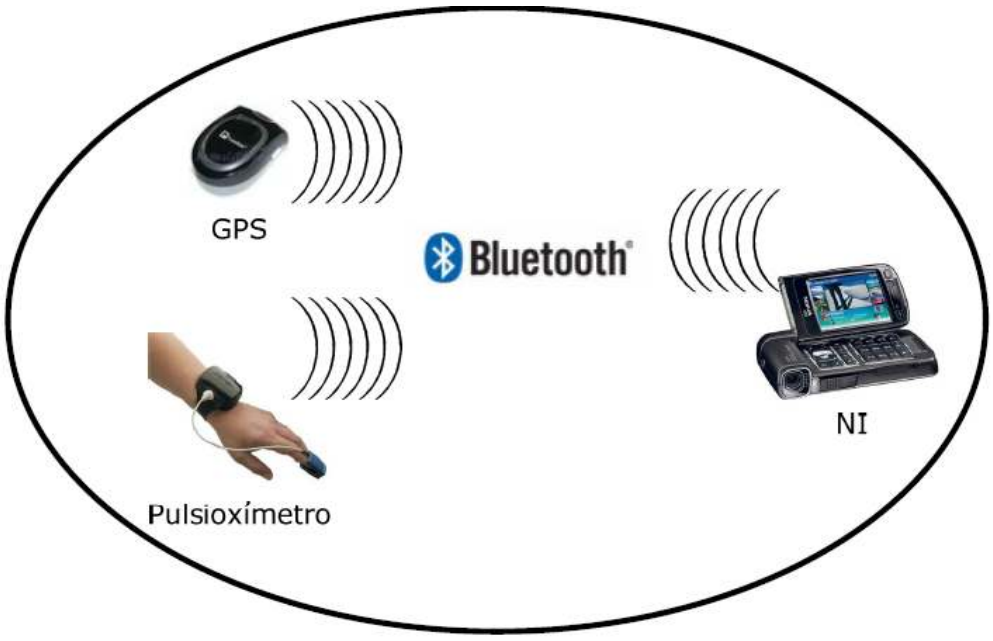

Figura 18. Esquema de actores del sistema *iBAN*
[16]

Como se aprecia en la Figura 18, el sistema *iBAN* consta de los siguientes componentes:

- Un pulsioxímetro: Es un dispositivo que monitoriza la saturación de oxigeno en sangre, el ritmo cardiaco y la información pletismográfica del paciente.

- Un terminal *GPS*: Es un dispositivo que proporciona la localización del paciente en entornos abiertos

- Un nodo central (*NI*): Este componente es el encargado de:

    - Localizar al resto de dispositivos de la *iBAN* y conectarse a ellos mediante *Bluetooth*.

    - Leer de datos de los sensores.

    - Procesar la información.

    - Detectar y enviar de alarmas.

    - Transmitir la información del paciente al *SCC*.

Para el desarrollo del *NI*, el autor escogió el lenguaje de programación *Java*, argumentando que la principal ventaja de éste es la portabilidad que ofrece con respecto a otras plataformas, permitiendo el funcionamiento de la aplicación en dispositivos móviles de diferentes fabricantes, como *Nokia*®[7], *Sony-Ericsson*®[8], etc., así posibilita al desarrollador abstraerse del sistema operativo del terminal móvil.

2.1.5. Gestor de contraseñas mediante un dispositivo móvil accesible por Bluetooth

El proyecto que se describe en esta sección fue desarrollado en la Universidad Autónoma de Barcelona en Septiembre de 2008 [17]. El proyecto describe la solución adoptada para un sistema, que proporciona los datos del formulario de acceso (usuario/contraseña) para cualquier página web, mediante la información guardada en un teléfono móvil.

El proyecto implementa una solución por la cual el navegador de Internet *Firefox* es capaz de rellenar los formularios del tipo login/password a través de una extensión del navegador, que se conecta por *Bluetooth* al móvil del usuario y obtiene las claves del formulario a rellenar.

[7] Nokia® es marca registrada o marca comercial de Nokia Corporation.

[8] Sony-Ericsson® es marca registrada o marca comercial de Sony Ericsson Mobile Communications AB.

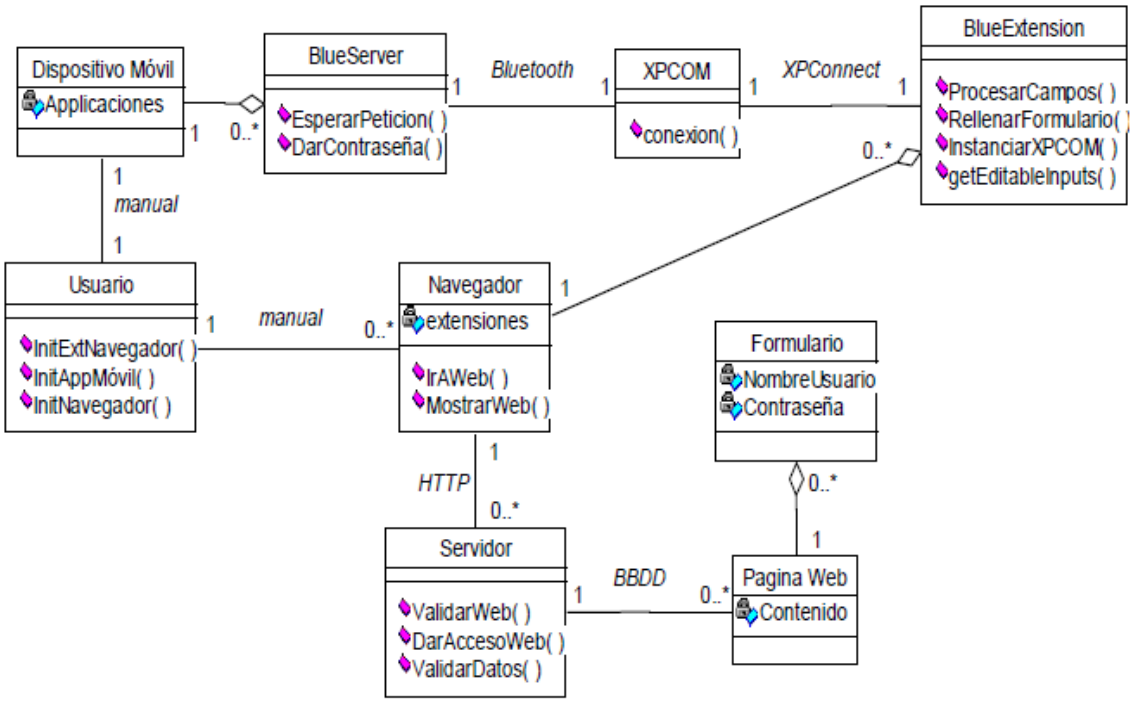

Figura 19. Diagrama de clases. Sistema gestor de contraseñas [17]

Mediante el diagrama de clases de la Figura 19, el autor describe el funcionamiento del sistema, representando a cada una de las partes como si de una clase se tratara, y mapeando las acciones como métodos de las clases:

- El usuario se encarga de iniciar una sesión en el navegador, y de iniciar la aplicación de gestión de contraseñas en su dispositivo móvil.
- El navegador accede al servidor web mediante el protocolo http para poder acceder a la pagina deseada.
- El servidor se encarga de validar la página web introducida por el usuario, y en caso de encontrarla en su base de datos, le da acceso a la misma.
- La extensión *BlueExtension* del navegador instancia el componente *XPCOM* mediante la interfaz *XPConnect*.
- El componente *XPCOM* establece la conexión *Bluetooth* con la aplicación del móvil, que realiza la función de servidor *Bluetooth*, y obtiene los datos requeridos para rellenar el formulario.
- La extensión rellena el formulario con los datos obtenidos desde la aplicación móvil.

La tecnología empleada en el desarrollo de toda esta arquitectura, difiere según el componente: la extensión *BlueExtension* del navegador *Firefox®*[9] se creó mediante *Javascript®*[10] y *XUL* (XML-based User-interface Language), el componente *XPCOM* se implementó mediante el lenguaje *C++* y la aplicación del dispositivo móvil se desarrollado con *J2ME*.

La justificación de la elección de las tecnologías viene marcada por la imposición de las herramientas utilizadas. En este caso, *Mozilla®*[11] proporciona las herramientas para crear extensiones, que obligan a utilizar *JavaScript* y *XUL*. En cuanto al componente *XPCOM*, es otra de las herramientas proporcionada por *Mozilla* para generar funciones más complejas de lo que permite el propio *JavaScript*, de ahí que estos componentes se escriban en código *C++*.

J2ME es la tecnología elegida por el autor para desarrollar la aplicación del dispositivo móvil. Para argumentar dicha elección, el autor redunda en lo expresado anteriormente por otros, considerando a *J2ME* la tecnología del futuro para dispositivos móviles, debido a que ésta ofrece una plataforma estándar para el desarrollo de aplicaciones y permite la portabilidad entre diferentes fabricantes.

2.1.6. Avisador médico en tecnología Bluetooth

El proyecto, desarrollado en la Escuela Politécnica Superior de Castelldefels de la Universidad Politécnica de Cataluña, en julio de 2009 [18], describe una solución semejante a la ya desarrollada en otro de los proyectos nombrados anteriormente.

En este proyecto se implementó un sistema de comunicación entre un dispositivo móvil y un sensor médico. Para el dispositivo móvil se desarrolló una aplicación capaz de recibir alertas vía *Bluetooth* por parte de dispositivos médicos. Los requisitos para el desarrollo de la aplicación móvil fueron:

- Permitir al personal médico conocer el estado del paciente con sólo acercarse al entorno de éste.

9 Firefox® es marca registrada o marca comercial de la Fundación Mozilla

10 JavaScript® es marca registrada o marca comercial de Oracle Corporation.

11 Mozilla® es marca registrada o marca comercial de la Fundación Mozilla.

- Consumir pocos recursos. Pensando en el uso de dispositivos con una autonomía reducida y alimentados mediante una batería.

- Ofrecer seguridad. Solo el personal acreditado debe poder comunicarse con el sensor, por lo que debe ser segura frente a ataques intencionados o no intencionados de otros dispositivos móviles.

Al no disponer de un señor comercial, como parte del proyecto se decidió emular el funcionamiento de un sensor mediante una aplicación que se ejecutase desde un ordenador.

Una vez más, la tecnología escogida para desarrollar la aplicación fue *J2ME*. El autor expone que de entre los diferentes lenguajes posibles: *C*, *Cocoa* y *Java*, la programación en *C* se usa para dispositivos con un sistema operativo *Symbian*®[12], *Cocoa* es un lenguaje de programación bastante reciente utilizado en dispositivos con sistemas operativos *Apple*®[13], y finalmente *Java* es un lenguaje multiplataforma que permite ejecutar el mismo código en plataformas hardware diferentes y que está presente en la mayoría de dispositivos móviles.

Para implementar la aplicación del ordenador, se diseñó un script que se ejecuta sobre un sistema operativo *Linux*®[14]. Donde el script es capaz de emular las funciones requeridas para el dispositivo médico.

2.1.7. Sistema de posicionamiento en interiores mediante balizas Bluetooth

El sistema que se detalla a continuación fue desarrollado en la Escuela Técnica Superior de Ingeniería (ICAI) de la Universidad Pontificia Comillas, en septiembre de 2010 [19].

El proyecto fue enfocado dentro del marco de ambientes inteligentes, ambientes que son capaces de detectar la presencia humana e interaccionar con ella, en función de su comportamiento. Se implementó una alternativa al posicionamiento

[12] Symbian® es una marca comercial o marca registrada de Symbian Software Ltd.

[13] Apple® es marca registrada o marca comercial de Apple Inc.

[14] Linux® es marca registrada o marca comercial de Linux Torvalds en los Estados Unidos.

en espacios cerrados, a través del desarrollo de una aplicación móvil, que permite la interacción con un entorno inteligente *ad-hoc*, formado por balizas pasivas (ver Figura 20).

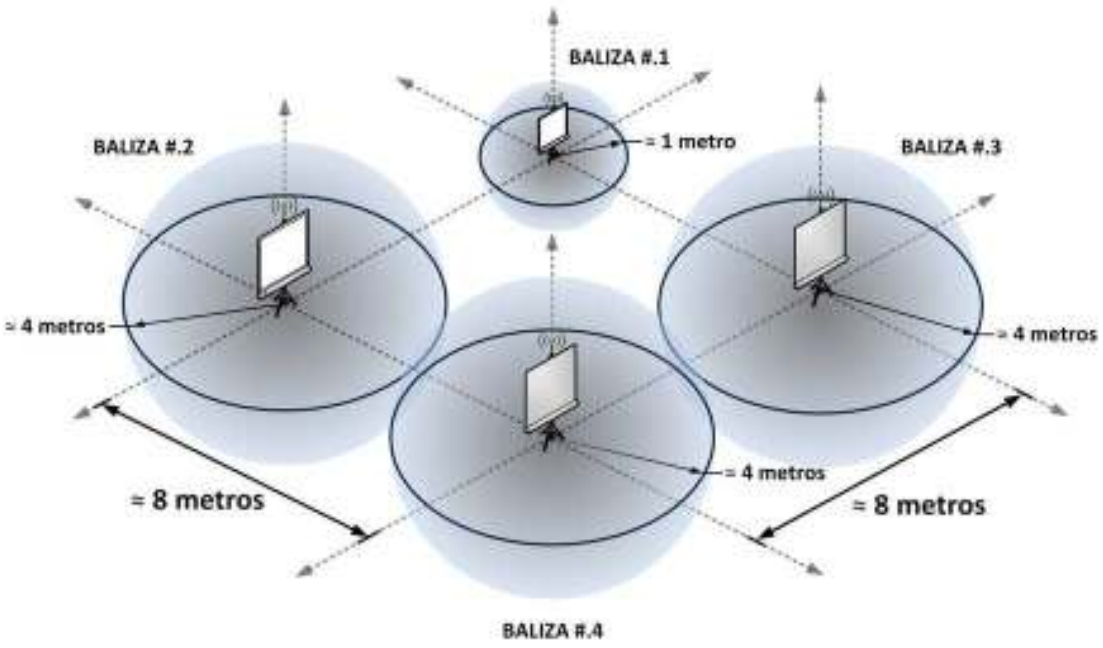

Figura 20. Disposición de las balizas en el entorno propuesto [19]

El sistema desarrollado, denominado *BlueBrowse,* realizó un estudio sobre la tecnología *Bluetooth*, proponiendo como balizas pasivas de bajo coste auriculares *Bluetooth*, y planteó una lógica de posicionamiento mediante un sistema no distribuido, presente de forma activa en un dispositivo móvil localizador.

El cometido de la aplicación móvil, conociendo los datos de las balizas presentes, es filtrar qué dispositivos *Bluetooth* debe encontrar, y mostrar en tiempo real información de baliza localizada y su posición exacta en el entorno.

La tecnología elegida para el desarrollo de la aplicación del dispositivo móvil fue nuevamente *J2ME*.

2.2. Conclusiones

De los proyectos descritos en este capítulo, se concluye que en la fecha de presentación de los mismos (la mayoría entre 2007 y 2009), la tecnología con mayor proyección y con más adeptos para el desarrollo de aplicaciones móviles era *J2ME*.

La justificación mayoritaria del porqué elegir *J2ME* en lugar de otras tecnologías, se sustenta en que *Java* ofrece una solución multiplataforma gracias al uso de su *máquina virtual*. De modo que para la plataforma *J2ME* se implementaron una *máquina virtual* y una *API* específicas para el desarrollo de aplicaciones móviles, lo que posibilitó el poder codificar aplicaciones para cualquier plataforma hardware.

Sin embargo, hoy en día *J2ME* no ha cubierto las expectativas creadas en sus inicios, quedándose cada vez más en desuso. El porqué, se debe a la aparición del nuevo sistema operativo *Android*®[15] [20]. Éste ha irrumpido de forma devastadora dentro de los sistemas operativos (de software libre) orientados a teléfonos móviles, siendo el más extendido dentro de los actuales *smartphones*. Su único competidor es el sistema operativo *iOS*® (sistema operativo móvil de *Apple*) [16], que aún teniendo un gran volumen de mercado se ve limitado a un solo fabricante, teniendo como gran hándicap el ser un sistema propietario no abierto al libre desarrollo.

Aún siendo *Java* el lenguaje de programación empleado para desarrollar aplicaciones sobre la plataforma *Android*, se ha creado una máquina virtual y una *API* exclusivas para Android, desbancando a la plataforma *J2ME*.

Por otro lado, hay que mencionar que la aplicación desarrollada en el proyecto resulta muy interesante para el usuario final de la misma, ya que supone un coste mucho menor que el resto de dispositivos comerciales presentados en este capítulo.

[15] Android® es marca registrada o marca comercial de Google Inc.

[16] iOS® es marca registrada o marca comercial de Apple Inc.

3 Tecnologías

3.1. Introducción

En este capítulo se realiza una descripción acerca de las tecnologías y las herramientas utilizadas tanto para el desarrollo como el testeo del software desarrollado.

Debido a las ventajas que se han expuesto en el capítulo anterior, *Bluetooth* es la tecnología implementada para establecer la comunicación inalámbrica entre la aplicación móvil y el servicio del ordenador.

Siendo Java *ME* la plataforma utilizada como base de diseño del sistema *Localiza*, el desarrollo del proyecto se basa en esta tecnología.

Al final del capítulo se hace una breve mención tanto de la herramienta de programación, como del software de pruebas utilizados.

3.2. *Bluetooth*

Bluetooth [11] es un estándar global de comunicación inalámbrica que posibilita la transmisión de voz y datos entre diferentes equipos mediante un enlace por radiofrecuencia. Los principales objetivos que pretende conseguir son:

- Facilitar las comunicaciones entre equipos móviles y fijos.
- Eliminar cables y conectores entre los dispositivos a conectar.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre dispositivos.

3.2.1. Tecnología

La especificación *Bluetooth* define un canal de comunicación con una tasa de transferencia máxima de 720 kb/s. El rango típico de operación de *Bluetooth* es menor a 10 m, sin embargo se pueden alcanzar distancias de hasta 100 m con el uso de amplificadores.

Bluetooth trabaja en un rango de frecuencia que comprende de 2,4 a 2,8 GHz. Utiliza una técnica llamada “Salto de Frecuencia en un Espectro Expandido” (“Spread Spectrum Frecuency Hopping”) para evitar interferencia entre dos dispositivos. Ésta consiste en cambiar al azar la frecuencia en la que los transmisores envían señales,

alrededor de 1600 veces por segundo, lo cual hace muy poco probable que dos transmisores decidan transmitir en exactamente la misma frecuencia al mismo tiempo, y de haber una colisión esta durará una fracción de segundo (1/1600 segundos), pudiendo resolverse el conflicto por medio de software creado para corrección de errores

Bluetooth utiliza 79 canales de radiofrecuencia con un ancho de banda de 1MHz cada uno y una rata máxima de símbolos de 1 MSímbolo/s. Después del envío de cada paquete en una determinada frecuencia de transmisión, ésta cambia a otra de las 79 frecuencias.

Las conexiones entre dispositivos *Bluetooth*, permiten establecer comunicaciones creando redes ad hoc de tipo *WPAN* (redes de área personal), con una topología de red conocida como *piconets*. Las *piconets* se establecen de forma dinámica y automática en el momento que un dispositivo *Bluetooth* activo accede y permite una proximidad radio con otro dispositivo *Bluetooth* activo. De este modo se define un espacio de operación personal omnidireccional en el que se permite la movilidad de los dispositivos.

3.2.2. Seguridad

Los dispositivos y servicios tienen diferentes niveles de seguridad. Para los dispositivos, hay dos niveles: “Dispositivo confiable” y “Dispositivo no confiable”. Un dispositivo confiable ya ha sido vinculado con otro dispositivo y tiene acceso ilimitado a todos los servicios.

Los servicios tienen tres niveles de seguridad:

- Servicios que requieren autorización y autenticación.
- Servicios que requieren autenticación solamente.
- Servicios que están abiertos a todos los dispositivos.

3.2.3. Arquitectura (Pila de protocolos)

La pila o *stack* de protocolos *Bluetooth* [21], como se puede observar en la Figura 21, se basa en el modelo de referencia OSI (Open System Interconnect) de ISO (Internacional Standard Organization) para interconexión de sistemas abiertos. La especificación *Bluetooth* utiliza una arquitectura de protocolos que divide las diversas

funciones de red en un sistema de niveles. En conjunto, permiten el intercambio transparente de información entre aplicaciones diseñadas de acuerdo con dicha especificación y fomentan la interoperabilidad entre los productos de diferentes fabricantes.

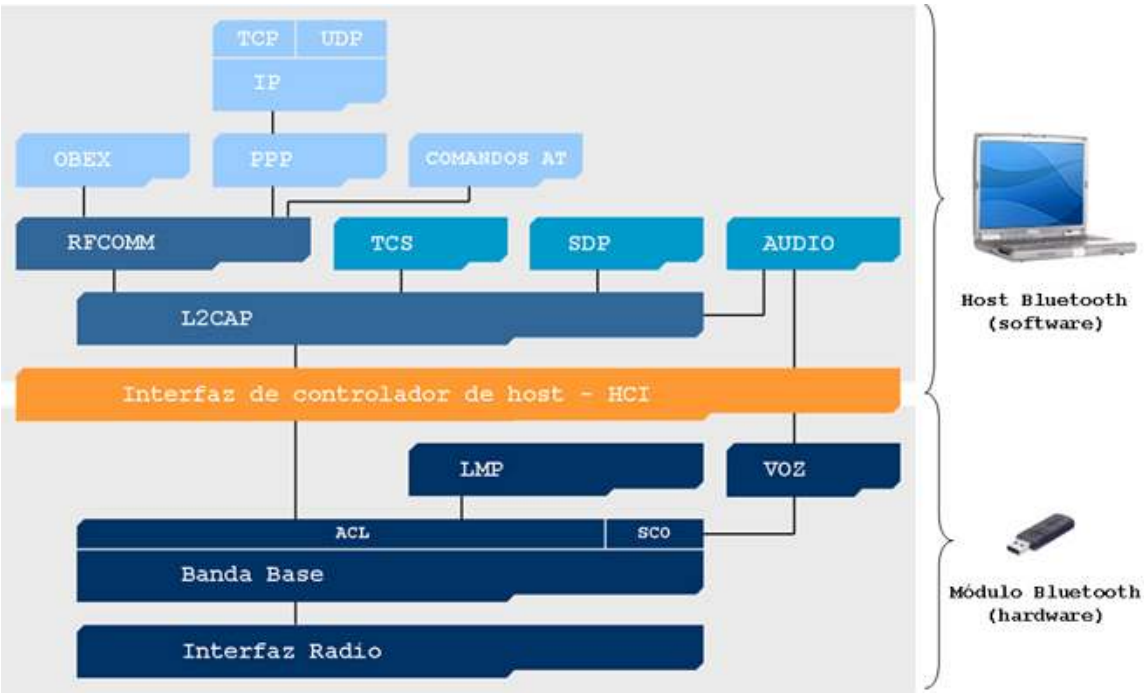

Figura 21. Pila de Protocolos *Bluetooth*
[22]

La pila está compuesta por numerosos protocolos que pueden ser clasificados según su propósito en los siguientes cuatro grupos:

- Protocolos centrales de *Bluetooth*: Banda Base, LMP, L2CAP, SDP.
- Protocolos de reemplazo de cable: RFCOMM.
- Protocolos de control de telefonía: TCS BIN, Comandos AT.
- Protocolos adaptados: PPP, UDP/TCP/IP, OBEX.

A su vez, la pila se divide en dos zonas:

- El Módulo *Bluetooth* (Hardware), encargado de las tareas relacionadas con el envío de información a través de la interfaz de radiofrecuencia.
- El Host *Bluetooth*, encargado de la parte relacionada con las capas superiores de enlace y aplicación.

Ambas zonas se comunican por la interfaz de Controlador de Host, HCI (Host Controller Interface). Ésta provee una interfaz estandarizada para ambos bloques. Sobre la capa de protocolos específicos de *Bluetooth*, cada fabricante puede implementar su capa de protocolos de aplicación propietarios. De esta forma, *Bluetooth* es considerada como una especificación abierta que se expande enormemente, con un gran número de aplicaciones que pueden beneficiarse de las capacidades que ofrece esta tecnología inalámbrica. Sin embargo, la especificación *Bluetooth* exige que a pesar de la existencia de diferentes pilas de protocolos de aplicación propietarios, se mantenga la interoperabilidad entre dispositivos que implementen diferentes pilas.

Como protocolos más reseñables de la pila se detallan los siguientes:

- ***Banda base:*** es el protocolo que especifica los procedimientos de acceso de medios y capa física entre dispositivos *Bluetooth*. Su función principal es permitir el enlace físico por radiofrecuencia (RF) entre unidades *Bluetooth* dentro de una *piconet*, realizando tareas de modulación y demodulación de los datos en señales RF que se transmiten por el aire.

- ***LMP (Link Manager Protocol):*** es el protocolo encargado del control y configuración del enlace, así como también el responsable de la autenticación y encriptación.

- ***L2CAP (Logical Link Control and Adaptation Protocol):*** es el protocolo que se corresponde a la capa de enlace de datos del modelo OSI. Proporciona servicios de datos orientados y no orientados a la conexión a capas superiores. Multiplexa los protocolos de capas superiores con el fin de enviar varios protocolos sobre un canal banda base. Con el fin de manipular paquetes de capas superiores más grandes que el máximo tamaño del paquete banda base, los segmenta en varios paquetes banda base, mientras que la capa L2CAP del receptor, los reensambla en paquetes más grandes para la capa superior.

- ***SDP (Sevice Discovery Protocol):*** es el protocolo que se usa para consultar información del dispositivo, servicios o características de servicios.

- ***RFCOMM:*** es el protocolo que se encarga de emular las señales de control y datos del puerto serie RS-232 sobre el protocolo L2CAP, ofreciendo capacidades de transporte a servicios de capas superiores, como OBEX, que usan la interfaz puerto serie como mecanismo de transporte.

3.2.1. Perfiles

Además de los protocolos, el SIG (Special Interest Group) *Bluetooth* ha definido los perfiles *Bluetooth* [23]. Un perfil *Bluetooth* define maneras estándar de usar los protocolos y sus características, para un uso en particular. Un dispositivo *Bluetooth* puede soportar uno o más perfiles.

Existe un amplio abanico de perfiles que detallan los diferentes tipos de uso y aplicaciones de la tecnología inalámbrica *Bluetooth*, estos se pueden agrupar en cuatro grupos:

- ***GAP (Generis Access Profile):*** Este perfil define los procedimientos generales para el descubrimiento y establecimiento de conexión entre dispositivos *Bluetooth*.

- ***SPP (Serial Port Profile):*** Este perfil define los requerimientos necesarios para dispositivos *Bluetooth*, para establecer conexiones puerto serie usando RFCOMM entre dos dispositivos semejantes. Así mismo, también define los protocolos y procedimientos a usar por dispositivos que emplean *Bluetooth* para la emulación del protocolo puerto serie RS-232 (o similar).

- ***GOEP (Generic Object Exchange Profile):*** Este perfil define protocolos y procedimientos usados por aplicaciones para ofrecer características de intercambio de objetos. Los dispositivos más comunes que usan este modelo son agendas electrónicas, *PDA*s, teléfonos celulares y teléfonos móviles. El GOEP es dependiente del perfil SPP.

- ***SDAP (Service Discovery Application Profile):*** Este perfil define los protocolos y procedimientos para una aplicación en un dispositivo *Bluetooth*, donde se desea descubrir y recuperar información relacionada con servicios localizados en otros dispositivos. El SDAP es dependiente del GAP.

3.3. *Java ME* (*Java Platform, Micro Edition*)

La plataforma *Java ME* [24] está enfocada a la aplicación de la tecnología *Java* en dispositivos electrónicos con capacidades computacionales y gráficas muy reducidas, tales como teléfonos móviles, *PDA*s o electrodomésticos inteligentes. Esta edición tiene unos componentes básicos que la diferencian de las otras versiones, destacando el uso de una máquina virtual denominada *KVM* (*Kilo Virtual Machine*), debido a que

requiere sólo unos pocos kilobytes de memoria para funcionar (en vez del uso de la clásica *JVM* (*Java Virtual Machine*)) y a que se incluye un pequeño y rápido recolector de basura.

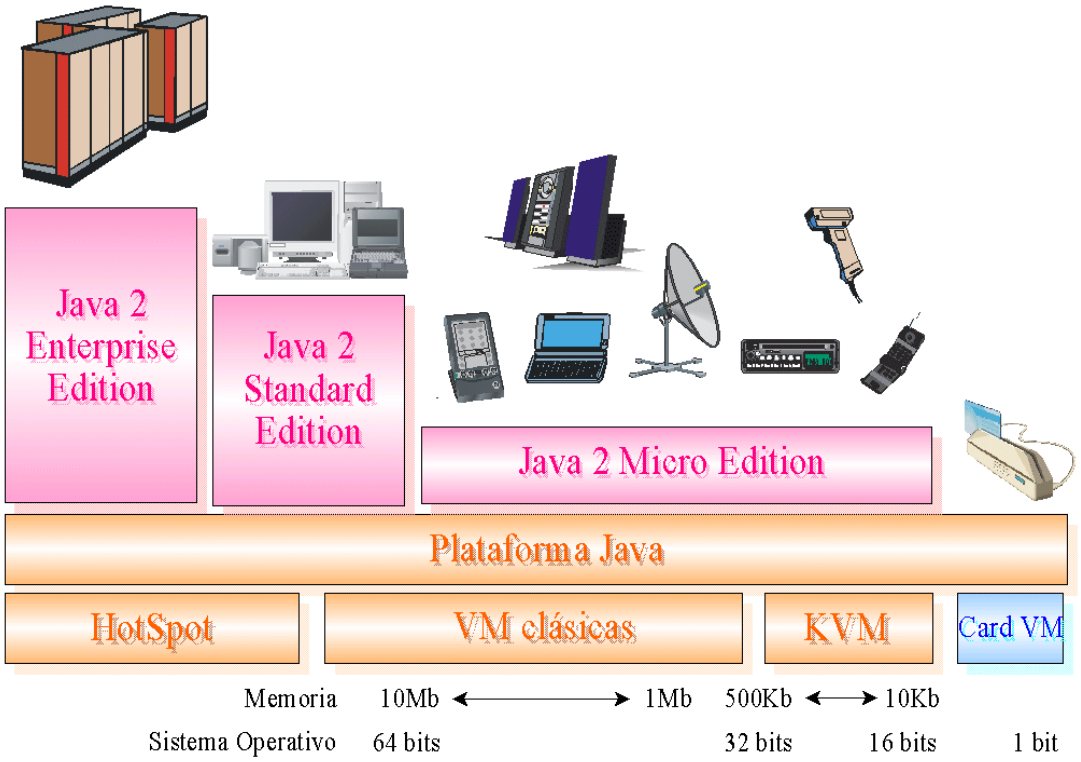

Figura 22. Arquitectura de la plataforma *Java 2* de *Sun* [24]

En la Figura 22 se muestra la arquitectura de la plataforma *Java* 2, aunque el nombre actual es simplemente *Java*. En la actualidad, no es realista ver *Java* como un simple lenguaje de programación, sino como un conjunto de tecnologías que abarca a todos los ámbitos de la computación con dos elementos en común que son:

- El código fuente en lenguaje *Java* es compilado a un código intermedio interpretado por una *JVM*, por lo que hace al código *Java* independiente de la plataforma.

- Todas las tecnologías comparten un conjunto más o menos amplio de *API*s básicas del lenguaje, agrupadas principalmente en los paquetes `java.lang` y `java.io`.

Java ME contiene una mínima parte de las *API*s de *Java*. Esto es debido a que la edición estándar de *API*s de *Java* ocupa 20 Mb, y los dispositivos pequeños disponen de una cantidad de memoria mucho más reducida. En concreto, *Java ME* usa 37 clases de la plataforma *Java SE* provenientes de los paquetes `java.lang`, `java.io`, `java.util`. Esta parte de la *API* que se mantiene fija forma parte de lo que se denomina "configuración".

Otras diferencias con la plataforma *J2SE* vienen dadas por el uso de una máquina virtual distinta de la clásica *JVM* denominada *KVM*. Esta *KVM* tiene unas restricciones que hacen que no posea todas las capacidades incluidas en la *JVM*. Se observa entonces, que *Java ME* representa una versión simplificada de *Java SE*. *Sun* separó estas dos versiones debido a que *Java ME* estaba orientado a dispositivos con limitaciones de proceso y de capacidad gráfica. Por lo tanto, separó ambos productos por razones de eficiencia. Hoy en día, *Java ME* es un subconjunto de *Java SE* (excepto por el paquete `javax.microedition`) ya que contiene varias limitaciones con respecto a *Java SE*.

Java ME comprende diferentes entornos de ejecución, en función del dispositivo que utilicemos. El entorno de ejecución se compone de una selección de los siguientes elementos:

- ***Maquina virtual:*** La *KVM* es la implementación utilizada para los terminales de telefonía móvil. Existe otra implementación, la *CVM*, diseñada para dispositivos de más potencia.

- ***Configuración:*** la configuración es un conjunto de clases básicas orientadas a conformar el corazón de las implementaciones para dispositivos de características específicas. Existen dos configuraciones definidas en *Java ME*: Connected Limited Device Configuration (CLDC) enfocada a dispositivos con restricciones de procesamiento y memoria, y Connected Device Configuration (CDC) enfocada a dispositivos con más recursos. Los teléfonos móviles usan la configuración CLDC.

- ***Perfil:*** los perfiles son unas bibliotecas *Java* de clases específicas, orientadas a implementar funcionalidades de más alto nivel para familias específicas de dispositivos. El perfil necesario para crear aplicaciones móviles es el MIDP (Mobile Information Device Profile), que está construido sobre la configuración CLDC.

- ***Paquetes Opcionales:*** Son los llamados *JSR (Java Specification Requests)*, constituyen un conjunto de documentos formales que describen las especificaciones y tecnologías propuestas para que sean añadidas a la plataforma *Java*. La implementación de los *JSR*s se presenta como una *API* accesible para el programador.

3.3.1. Configuraciones

Como ya se ha mencionado anteriormente, una configuración es el conjunto mínimo de *API*s *Java* que permiten desarrollar aplicaciones para un grupo de dispositivos. Estas *API*s describen las características básicas comunes a todos los dispositivos:

- Características soportadas del lenguaje de programación *Java*.
- Características soportadas por la *Máquina Virtual Java*.
- Bibliotecas básicas de *Java* y *API*s soportadas.

Como se ha visto con anterioridad, existen dos configuraciones en *Java ME*:

- CLDC, orientada a dispositivos con limitaciones computacionales y de memoria
- CDC, orientada a dispositivos con no tantas limitaciones.

Para el desarrollo de la aplicación a implementar en este proyecto se va a utilizar la configuración CLDC, por lo que en el apartado siguiente se procede a dar una breve descripción de la misa.

3.3.1.1. Configuración de dispositivos limitados con conexión

La configuración CLDC está orientada a dispositivos dotados de conexión y con limitaciones en cuanto a capacidad gráfica, cómputo y memoria. Un ejemplo de estos dispositivos son: teléfonos móviles, buscapersonas (pagers), PDAs, organizadores personales, etc. Algunas de estas restricciones vienen dadas por el uso de la KVM, necesaria al trabajar con la CLDC debido a su pequeño tamaño. Los dispositivos que usan CLDC deben cumplir los siguientes requisitos:

- Disponer entre 160 kb y 512 kb de memoria total disponible. Como mínimo se debe disponer de 128 kb de memoria no volátil para la *Máquina Virtual Java* y

las bibliotecas CLDC, y 32 kb de memoria volátil para la Máquina Virtual en tiempo de ejecución.

- Procesador de 16 ó 32 bits con al menos 25 MHz de velocidad.
- Ofrecer bajo consumo, debido a que estos dispositivos trabajan con suministro de energía limitado, normalmente baterías.
- Tener conexión a algún tipo de red, normalmente sin cable, con conexión intermitente y ancho de banda limitado (unos 9600 bps).

Por otro lado, la CLDC aporta las siguientes funcionalidades a los dispositivos:

- Un subconjunto del lenguaje *Java* y todas las restricciones de su *Máquina Virtual (KVM)*.
- Un subconjunto de las bibliotecas *Java* del núcleo.
- Soporte para E/S básica.
- Soporte para acceso a redes.
- Seguridad.

3.3.2. Perfiles

Los perfiles son los encargados del mantenimiento del ciclo de vida de la aplicación, de las interfaces de usuario y del manejo de eventos. El perfil se define como un conjunto de *API*s, que definen las características de un dispositivo, orientado a un ámbito de aplicación determinado. Los perfiles identifican un grupo de dispositivos por la funcionalidad que proporcionan (electrodomésticos, teléfonos móviles, etc.) y el tipo de aplicaciones que se ejecutarán en ellos. Las librerías de la interfaz gráfica son un componente muy importante en la definición de un perfil, donde se pueden encontrar grandes diferencias entre interfaces, desde el menú textual de los teléfonos móviles, hasta los táctiles de las *PDA*s.

A la hora de construir una aplicación, se cuenta tanto con las *API*s del perfil, como con las de la configuración, por lo que se ha de tener en cuenta que un perfil siempre se construye sobre una configuración determinada. De este modo, se puede pensar en un perfil como un conjunto de *API*s que dotan a una configuración de funcionalidad específica.

Como sucede en la configuración, donde se usa una *Máquina Virtual Java* diferente para cada una, en los perfiles sucede lo mismo. Existen unos perfiles que se construyen sobre la configuración CDC y otros que se construyen sobre la CLDC. Para la configuración CDC existen los siguientes perfiles: Foundation Profile; Personal Profile; RMI Profile. Para la configuración CLDC existen los perfiles: PDA Profile y Mobile Information Profile (MIDP)

De entre todos los perfiles, el que aplica en este proyecto es el MIDP. Este perfil está construido sobre la configuración CLDC. Al igual que CLDC fue la primera configuración definida para *Java ME*, MIDP fue el primer perfil definido para esta plataforma. Este perfil está orientado para dispositivos con las siguientes características:

- Reducida capacidad computacional y de memoria.
- Conectividad limitada (en torno a 9600 bps).
- Capacidad gráfica muy reducida (mínimo un display de 96x54 pixels monocromo).
- Entrada de datos alfanumérica reducida.
- 128 kb de memoria no volátil para componentes MIDP.
- 8 kb de memoria no volátil para datos persistentes de aplicaciones.
- 32 kb de memoria volátil en tiempo de ejecución para la pila *Java*.

Los tipos de dispositivos que se adaptan a estas características son: teléfonos móviles, buscapersonas (pagers) o *PDA*s de gama baja con conectividad.

El perfil MIDP establece las capacidades del dispositivo, por lo tanto, especifica las *API*s relacionadas con:

- La aplicación (semántica y control de la aplicación)
- Interfaz de usuario
- Almacenamiento persistente
- Trabajo en red.

- Temporizadores.

Las aplicaciones que se desarrollan utilizando el perfil MIDP, reciben el nombre de *MIDlets* (por simpatía con *APPlets*). Se dice así, que un *MIDlet* es una aplicación *Java* realizada con el perfil MIDP sobre la configuración CLDC.

3.3.3. MIDlet

Los *MIDlets* [24] [25] son aplicaciones creadas usando la especificación MIDP. Están diseñados para ser ejecutados en dispositivos con recursos limitados. Estos dispositivos en lugar de disponer de una interfaz de línea de comandos para ejecutar las aplicaciones, vienen provistos de un software llamado AMS (Application Management System), que es el encargado de ejecutar los *MIDlet*s y gestionar los recursos que éstos ocupan

Durante su ejecución, el *MIDlet* pasa por 3 estados diferentes (ver Figura 23). Estos tres estados son:

- Activo: El *MIDlet* está actualmente en ejecución.
- Pausa: El *MIDlet* no está actualmente en ejecución. En este estado, el *MIDlet* no debe usar ningún recurso compartido. Para volver a estar en ejecución, tiene que cambiar al estado Activo.
- Destruido: El *MIDlet* no está en ejecución, ni puede transitar a otro estado. Además se liberan todos los recursos ocupados por el *MIDlet*.

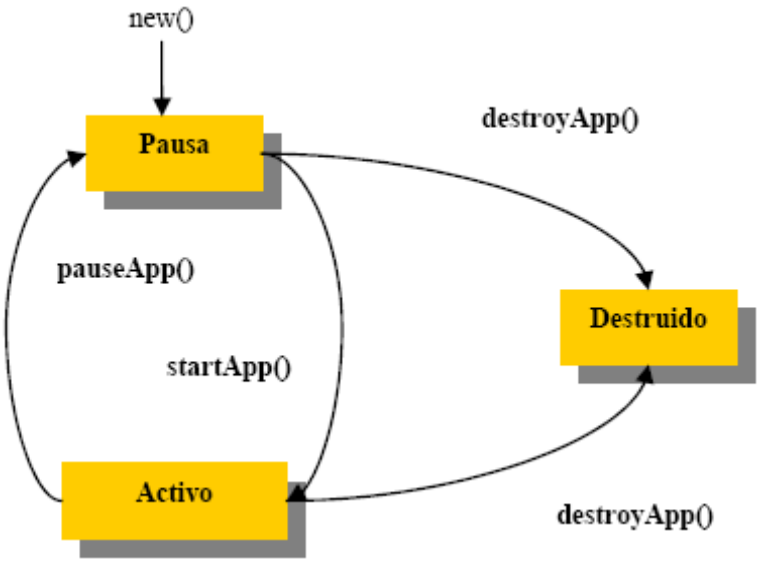

Figura 23. Diagrama de estados *MIDlet* en ejecución
[24]

El gestor de aplicaciones (AMS) se encarga de cambiar el estado de los *MIDlets*, cuando éste hace una llamada a uno de los métodos: `startApp()`, `pauseApp()` ó `destroyApp()`. El cambio de estado también puede estar provocado por la acción del usuario.

En una ejecución típica se realizan las siguientes acciones:

- En primer lugar se realiza la llamada al constructor del *MIDlet* pasando al estado de "Pausa", durante un corto período de tiempo.
- El AMS por su parte crea una nueva instancia del *MIDlet*.
- Cuándo el dispositivo está preparado para ejecutar el *MIDlet*, el AMS invoca al método `startApp()` para entrar en el estado de "Activo".
- El *MIDlet* entonces, ocupa todos los recursos que necesita para su ejecución.
- Durante este estado, el *MIDlet* puede pasar al estado de "Pausa" por una acción del usuario, o bien, por el AMS
- Tanto en el estado "Activo" como en el de "Pausa", el *MIDlet* puede pasar al estado "Destruido" realizando una llamada al método `destroyApp()`. Esto puede ocurrir porque el *MIDlet* haya finalizado su ejecución o porque una aplicación prioritaria necesite ser ejecutada en memoria en lugar del *MIDlet*.

- Una vez destruido el *MIDlet*, éste libera todos los recursos ocupados.

3.3.4. *API*s opcionales

Para la implementación del proyecto es necesario hacer uso de dos *API*s opcionales, una para el manejo del acelerómetro del teléfono móvil, la *API* JSR-256, y otra para implementar la interfaz *Bluetooth*, la *API* JSR-82.

3.3.4.1. JSR-256

La *API* JSR-256 [26] permite a los desarrolladores de aplicaciones *Java ME*, obtener de forma sencilla y uniforme los datos ofrecidos por los sensores contenidos en el móvil. Se considera un sensor como cualquier fuente de datos de medición. Existen múltiples tipos de sensor, desde sensores físicos (como los magnetómetros y acelerómetros), hasta los sensores virtuales (que combinan y manipulan los datos que reciben de otros tipos de sensores físicos).

La *API* también proporciona información sobre la monitorización de los datos medidos. La aplicación es capaz de establecer unos límites y rangos de medida para la monitorización. Si los valores obtenidos superan algunos de los límites o rangos establecidos, la aplicación es notificada.

La *API* no proporciona ningún método de control sobre el sensor, solo proporciona métodos para recibir la información del mismo. Posibles métodos de control serían comienzo, parada, calibrado y configuración del rango de medida. Esto queda fuera de la *API* por cuestiones de simplicidad, de modo que la implementación de la *API* se debe ocupar de esta funcionalidad de forma automática.

3.3.4.2. JSR-82

La especificación de *Bluetooth* cubre muchas capas y perfiles, la *API* JSR-82 [27] [28] se limita a las siguientes áreas:

- Transmisión de datos y no de voz
- Los protocolos: L2CAP, RFCOMM, SDP y OBEX
- Los perfiles: GAP, SPP y GOEP.

La plataforma principal de desarrollo de la *API* JSR-82 es *Java ME*. La *API* ha sido diseñada para depender de la configuración CLDC. Sin embargo existen implementaciones alternativas para poder hacer uso de esta *API* en *Java SE*.

La *API* queda dividida en dos partes: el paquete `javax.Bluetooth` y el paquete "javax.obex". Los dos paquetes son totalmente independientes. El primero de ellos define clases e interfaces básicas para el descubrimiento de dispositivos, descubrimiento de servicios, conexión y comunicación.

La comunicación a través de `javax.Bluetooth` es a bajo nivel: mediante flujos de datos o mediante la transmisión de *arrays* de bytes. Por el contrario el paquete `javax.obex` permite manejar el protocolo de alto nivel OBEX (Object Exchange).

El protocolo OBEX es muy similar a HTTP y es utilizado sobre todo para el intercambio de archivos. Consiste es un estándar desarrollado por IrDA y es utilizado también sobre otras tecnologías inalámbricas distintas de *Bluetooth*.

A partir del diagrama de clases expuesto en la Figura 24, se pasa a detallar el funcionamiento de los componentes principales del paquete `javax.Bluetooth`, desde el punto de vista de un *MIDlet* que implementa un cliente *Bluetooth*.

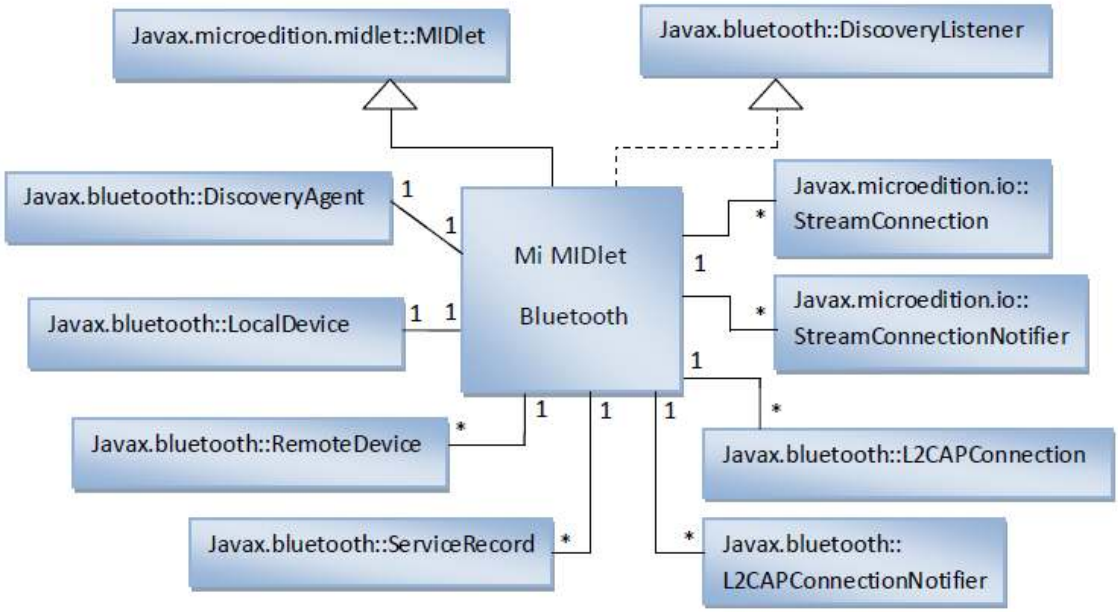

Figura 24. Diagrama de objetos de un *MIDlet Bluetooth*
(Adaptación de [29])

- ***Clase* `LocalDevice`.** Proporciona acceso a datos del dispositivo utilizado. Esta clase tiene un constructor privado que impide crear un objeto `LocalDevice` nuevo, por lo que solo se podrá obtener una referencia del mismo con el método `LocalDevice.getLocalDevice()`.

- ***Clase* `DiscoveryAgent`.** Las búsquedas de dispositivos y servicios *Bluetooth* se realizarán a través de un objeto `DiscoveryAgent`. Este objeto es único y se obtendrá a través del método `getDiscoveryAgent()` del objeto `LocalDevice`.

- ***Clase* `RemoteDevice`.** Permite obtener la dirección *Bluetooth* del dispositivo que representa (dispositivo remoto) a través del método `getBluetoothAddress()` en forma de *String*. El objeto `RemoteDevice` se incluirá como parámetro dentro del método `searchServices()` de la clase `DiscoveryAgent`.

- ***Clase* `UUID`.** La clase `UUID` (universally unique identifier) representa identificadores universales únicos. Se trata de enteros de 128 bits que identifican protocolos y servicios. Se puede crear un objeto `UUID` a partir de un *String* o de un entero largo. El `UUID` es utilizado como parámetro dentro del método `searchServices()` de la clase `DiscoveryAgent`.

- **Interfaz DiscoveryListener.** Esta es la interfaz que debe implementar cualquier cliente *Bluetooth*, ya que permite a la aplicación recibir eventos de descubrimiento de dispositivos y servicios. Esta interfaz provee cuatro métodos, dos para descubrir dispositivos y dos para descubrir servicios:

    - `deviceDiscovered()`: es llamado cada vez que se descubre un nuevo dispositivo durante una búsqueda.

    - `inquiryCompleted()`: es invocado cuando una búsqueda de dispositivos ha finalizado.

    - `servicesDiscovered()`: es llamado cada vez que se descubre un servicio nuevo durante una búsqueda de servicios.

    - `serviceSearchCompleted()`: es invocado cuando una búsqueda de servicios ha finalizado.

Conexión. El *API* `javax.Bluetooth` permite usar dos mecanismos de conexión: SPP (RFCOMM) y L2CAP. Mediante SPP se obtendrán datos orientados a *streams* de tipo `InputStream` y `OutputStream`, mientras que en L2CAP se enviarán y recibirán arrays de bytes. Para abrir cualquier tipo de conexión se hará uso de la clase `javax.microedition.io.Connector`, en concreto se usará el método estático `open()` que está sobrecargado; su versión más sencilla requiere un parámetro que es un *String* el cual contendrá la *URL* con los datos necesarios para realizar la conexión. La *URL* será diferente dependiendo si se desea ser cliente o servidor de una conexión L2CAP o SPP. Las diferentes interfaces a utilizar así como la línea de la *URL* que define el protocolo, están indicadas en la tabla II.

Protocolo	Diferencia en *URL*	Notifier	Connector
RFCOMM	btspp:	StreamConnectionNotifier	StreamConnection
L2CAP	Btl2cap:	L2CAPConnectionNotifier	L2CAPConnection
OBEX	Btgoep:	SessionNotifier	SessionConnection

Tabla II Protocolos de comunicación *Bluetooth*

3.4. Entorno de desarrollo y simulación

El entorno de desarrollo seleccionado para desarrollar el proyecto ha sido el *NetBeans®*[17] IDE 7.1. Es un entorno de desarrollo integrado open-source (software distribuido y desarrollado libre y gratuitamente), que soporta el desarrollo de todo tipo de aplicaciones *Java* y también de otros tipos, como *PHP, C, C++ y JavaScrip*, donde se permite escribir, compilar, depurar y ejecutar los programas.

La plataforma *NetBeans* permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo *Java* que contiene clases de *Java* escritas para interactuar con las *API*s de *NetBeans* y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma *NetBeans* pueden ser extendidas fácilmente por otros desarrolladores de software.

[17] NetBeans® es marca registrada o marca comercial de Oracle, Corp.

El módulo requerido para implementar la aplicación de este proyecto es el *click*e *Sun*®[18]. Este módulo contiene el conjunto de herramientas para la creación de aplicaciones o *MIDlets*, compatibles con la especificación MIDP, para ser instaladas en teléfonos móviles. Además contiene un emulador para distintos terminales.

En fase de desarrollo se utilizará un entorno simulado para correr la aplicación, para ello *NetBeans* permite integrar el *SDK* (Software Development Kit) de los principales fabricantes. Debido a que el terminal que se utilizará para probar la aplicación es de la marca Samsung®[19], el entorno *NetBeans+Samsung SDK* permitirá ejecutar la aplicación desarrollada en un emulador del modelo mismo modelo.

[18] Sun® es marca registrada o marca comercial de Sun Microsystems, Inc.

[19] Samsung® es marca registrada o marca comercial de Samsung Electronics.

4 Arquitectura

4.1. Introducción

Al enfrentar el desarrollo de este proyecto, se plantean dos posibles arquitecturas para desarrollar la funcionalidad requerida. Dentro de cada arquitectura se definen dos partes en común. Por un lado, está la aplicación del teléfono móvil, y por otro el lado el software que se ejecuta en el ordenador.

La primera arquitectura, denominada “Arquitectura Orientada a Driver”, implica el desarrollo de un driver para el ordenador. La segunda arquitectura, denominada “Arquitectura Orientada a Servicio”, supone el desarrollo de un servicio en lugar de un driver.

Ambas arquitecturas siguen el modelo cliente-servidor (ver Figura 25), donde existe un dispositivo que ofrece un servicio (servidor) y otros dispositivos acceden a él (clientes). Dependiendo de qué parte de la comunicación se deba programar es necesario realizar una serie de acciones diferentes.

En el caso del servidor, éste tiene que levantar un servicio *Bluetooth* mediante los siguientes pasos:

- Crear una conexión servidora.
- Especificar los atributos de servicio.
- Abrir la conexión para aceptar peticiones clientes.

Por otro lado, el cliente *Bluetooth* debe establecer una conexión con el servicio realizando las siguientes acciones:

- Búsqueda de dispositivos. La aplicación realizará una búsqueda de los dispositivos *Bluetooth* a su alcance que estén en modo conectable.
- Búsqueda de servicios. La aplicación realizará una búsqueda de servicios por cada dispositivo.
- Establecimiento de la conexión. Una vez encontrado un dispositivo que ofrece el servicio deseado nos conectaremos a él.
- Comunicación. Ya establecida la conexión se podrá leer y escribir en ella.

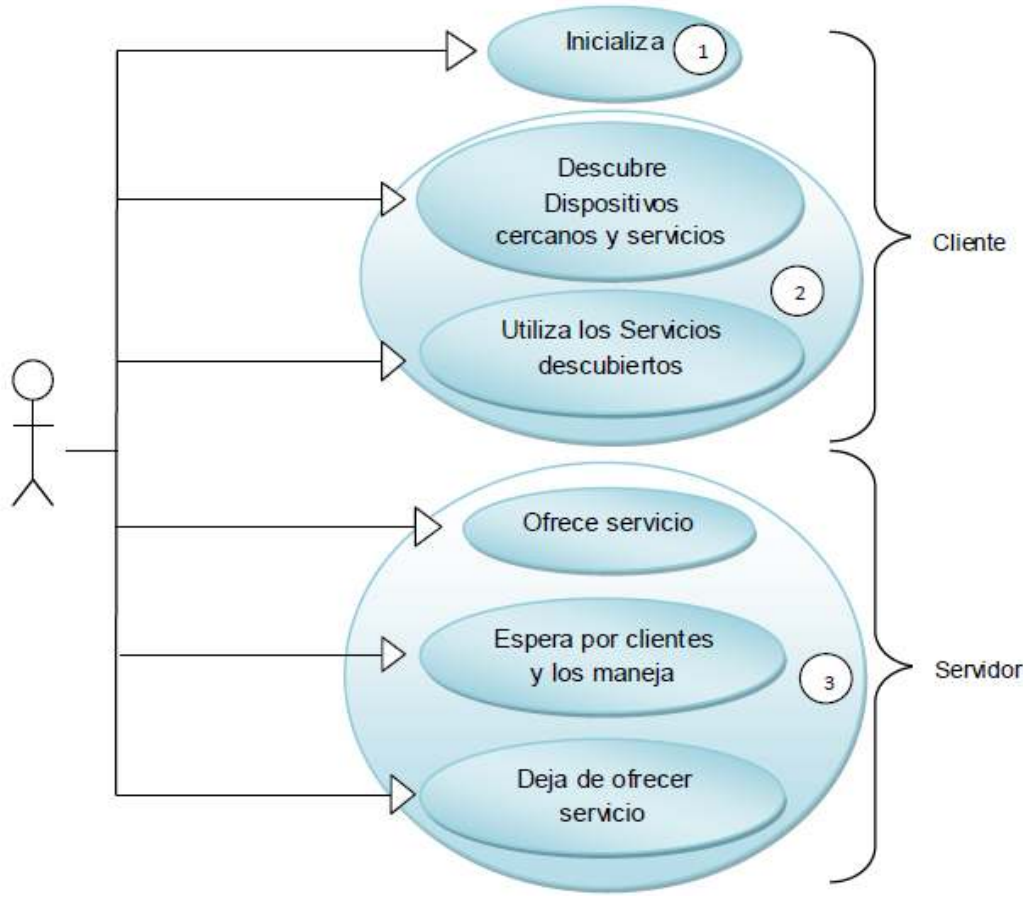

Figura 25. Caso de uso específico aplicación *Bluetooth*
(Adaptación de [29])

4.1.1. Arquitectura orientada a Driver

La arquitectura orientada a driver es la que se utiliza en las soluciones comerciales para los ratones *Bluetooth*. Como se ha descrito en el apartado anterior, el cometido de la aplicación es simular el software de un ratón *Bluetooth*.

La arquitectura orientada a driver pretende emular los eventos que genera un ratón *Bluetooth*, de modo que a través de un driver instalado en el ordenador, este sea capaz de mover el ratón mediante la generación de interrupciones en el sistema operativo.

La singularidad de esta arquitectura reside en el desarrollo de un driver en lugar de un servicio. De modo que el sistema operativo es el encargado de atender los eventos *Bluetooth* y detectar el driver responsable capaz de gestionarlos.

De cara a la aplicación móvil es transparente el que haya un driver o un servicio escuchando. De hecho, lo único que necesita es conocer el UUID hacia el

que se debe mandar la información. Pudiendo utilizar el UUID del servicio RFCOMM que se despliega por defecto en el registro de la pila de *Bluetooth*.

4.1.2. Arquitectura orientada a Servicio

La arquitectura orientada a servicio, como ya se ha mencionado, requiere desarrollar un servicio *Bluetooth* en el ordenador, y desarrollar una aplicación móvil que permita establecer una conexión *Bluetooth* con el servicio desplegado.

El servicio se crea por medio de una aplicación *Java*. Ésta se encarga de registrar el nuevo servicio, y abrir una conexión para recibir peticiones sobre el mismo.

La desventaja de desplegar un servicio en el ordenador es que obliga a tener una conexión abierta permanentemente, al contrario que con un driver que se queda latente hasta que no le llega un evento. Lo que supone un menor consumo de recursos del sistema.

Por otro lado, la ventaja que ofrece el servicio es poder utilizar la plataforma *Java*, que permite abstraerse del sistema operativo para que el que se desarrolla, ofreciendo una solución multiplataforma.

La programación orientada a driver, para sistema operativo *Windows,* limita el uso de las tecnologías disponibles principalmente a *C/C++*, ofreciendo la herramienta WDK (Windows Driver kit) [30] para el desarrollo y testeo de los mismos.

Como conclusión, el hecho de no poder programar un driver en *Java*, es el motivo por el que la arquitectura orientada a servicio es la elegida para el desarrollo de este proyecto.

5 Diseño y Desarrollo

5.1. Introducción

En este capítulo, en primer lugar, se realiza una descripción del diseño heredado de la aplicación *paciente*. A continuación, se explica el diseño del nuevo software implementado, que queda embebido dentro de esta aplicación.

El software desarrollado se va a presenta como una nueva opción, dentro del menú de entrada de la aplicación *paciente*. Dicha opción va a aparecer con el nombre de "*Control Remoto*", y va a ser totalmente independiente del resto de funcionalidades existentes. Por lo que no requerirá interacción alguna con la aplicación *supervisor*.

Por otro lado, cabe destacar que el proyecto deja abierto un aspecto que debe tratarse dentro de la parte del software del ordenador. Éste se trata de la acción del "*click*", ya que con la información proporcionada por la aplicación "C*ontrol Remoto"* solo se es capaz de mover el puntero del ratón.

5.2. Aplicación *paciente*

En esta sección se detalla el diseño de la aplicación *paciente*. Ésta consta de dos paquetes, uno llamado `Mensaje` y otro llamado `Paciente`. La nueva funcionalidad se desarrolla íntegramente dentro del paquete `Paciente`.

Para poder agregar el nuevo desarrollo, es necesario conocer el diseño del paquete `Paciente`. En la Figura 26 se presenta el diagrama de objetos del paquete.

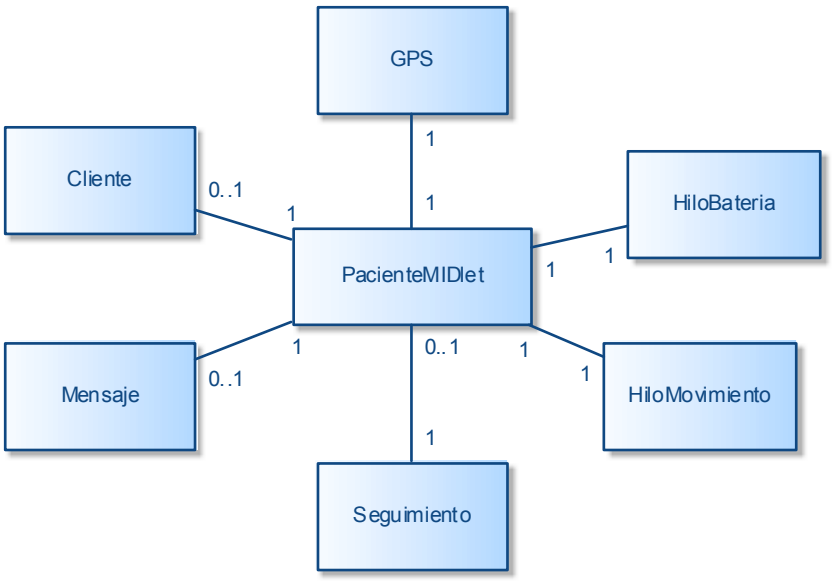

Figura 26. Base de diseño. Diagrama de objetos del paquete Paciente

Como se aprecia en el diagrama anterior, el paquete `Paciente` está compuesto por un *MIDlet* llamado `PacienteMIDlet`, que a su vez tiene asociados los objetos que se detallan a continuación:

- `GPS`. Es el objeto con el que se accede a la información proporcionada por el sensor de localización del teléfono móvil. Dicho objeto, proporciona las coordenadas de localización del paciente.

- `HiloBateria`. Es el objeto que se encarga de monitorizar el estado de la batería del teléfono móvil. Además, gestiona el envío de un mensaje de alerta a los supervisores que tenga registrados, en caso de que se agote la batería.

- `HiloMovimiento`. Es el objeto que se encarga de obtener los datos proporcionados por el acelerómetro. A su vez, ejecuta la función de detección de caídas del paciente. En caso de detectar una caída, gestiona el envío de un mensaje de alerta a los supervisores registrados.

- `Seguimiento`. El objeto se instancia solo en el caso que sea solicitada la opción de seguimiento desde un supervisor. Este objeto gestiona el envío de la posición según el intervalo fijado, y envía un mensaje de alerta en caso de que el paciente salga del perímetro establecido como área de seguridad.

- `Mensaje`. El objeto se instancia en el caso de que sea necesario enviar un mensaje. Este objeto se encarga de rellenar el contenido del mensaje a enviar hacia el supervisor.

- `Cliente`. El objeto se encarga de recorrer la lista de clientes (supervisores) asignados.

5.3. Software Control Remoto

Para el desarrollo del software "*Control Remoto*" se han añadido dos objetos nuevos al diagrama anterior de la Figura 26, resultando un nuevo diagrama de objetos como se puede ver en la Figura 27.

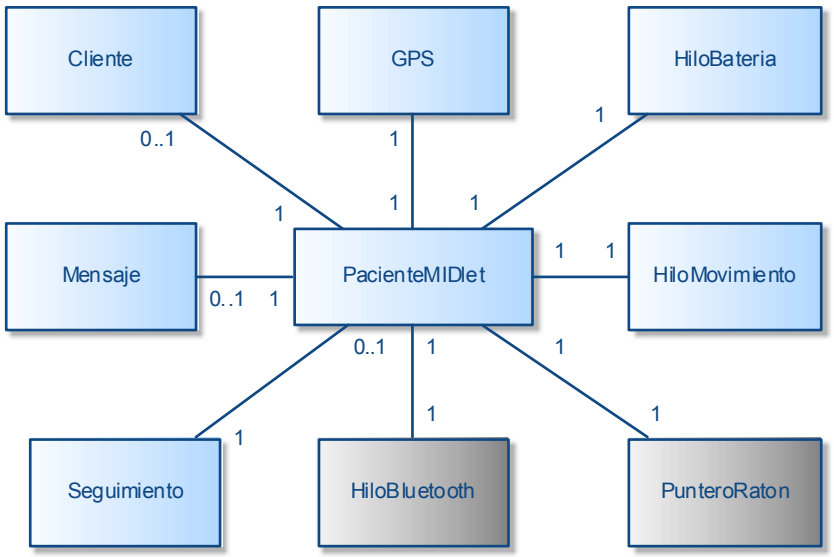

Figura 27. Nuevo diagrama de objetos del paquete Paciente

Los dos nuevos objetos que se han añadido al paquete `Paciente` son instancias de las clases `HiloBluetooth` y `PunteroRatón`, que se van a encargar de realizar las siguientes tareas:

- La clase `HiloBluetooth` se encarga de implementar el cliente *Bluetooth*.

- La clase `PunteroRatón` va a interactuar con las clases `HiloMovimiento` e `HiloBluetooth` para gestionar la información de las coordenadas del acelerómetro.

Además, se van a crear dos métodos nuevos dentro de la clase `HiloMovimiento`, con los que se va obtener periódicamente las coordenadas del acelerómetro.

En las siguientes secciones se describe:

- la implementación realizada del cliente *Bluetooth,*
- y la lógica de la clase `HiloMoviemiento` para la gestión del acelerómetro.

5.3.1. Cliente *Bluetooth*

En esta sección se detalla la implementación del cliente *Bluetooth* realizado. Para ilustrar el diseño desarrollado se han descrito diversos diagramas, que se presentan como apoyo para la explicación del mismo.

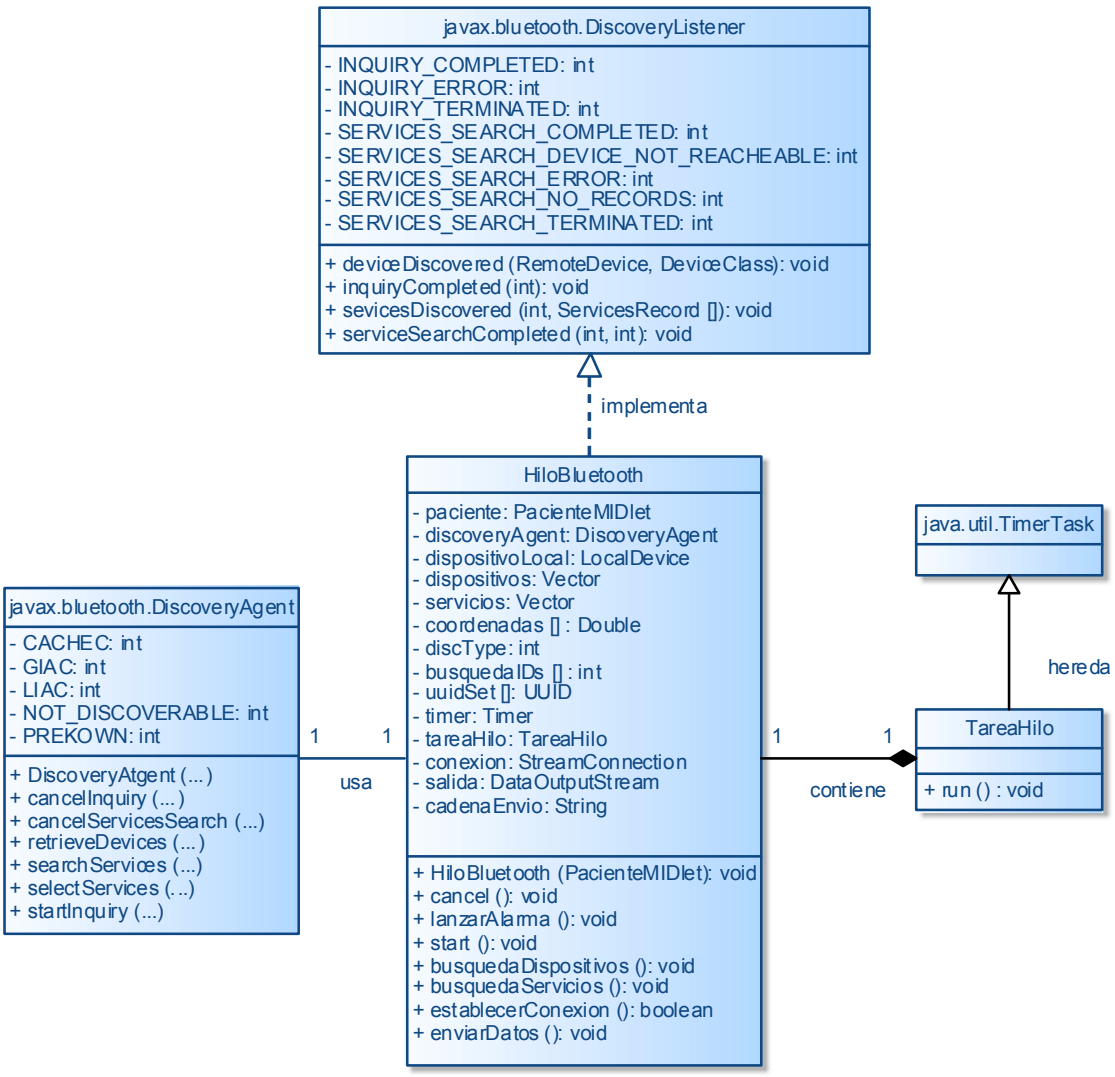

Figura 28. Diagrama de clases HiloBluetooth

Todo cliente *Bluetooth* debe realizar los procesos de búsqueda de dispositivos y servicios. Para ello la clase `HiloBluetooth` implementa la interfaz `DiscoveryListener` (ver Figura 28). Dicha interfaz contiene los métodos para realizar ambos procesos.

En primer lugar, para iniciar y cancelar el proceso de descubrimiento de dispositivos se usa la clase `DiscoveryAgent`, que invoca al método `startInquiry()`. Dentro de éste se invoca a los métodos `deviceDiscovered()` y `inquiryCompleted()` de la interfaz. Finalizado el proceso de descubrimiento se obtienen los posibles estados resultantes, que se muestran en el diagrama de la Figura 29.

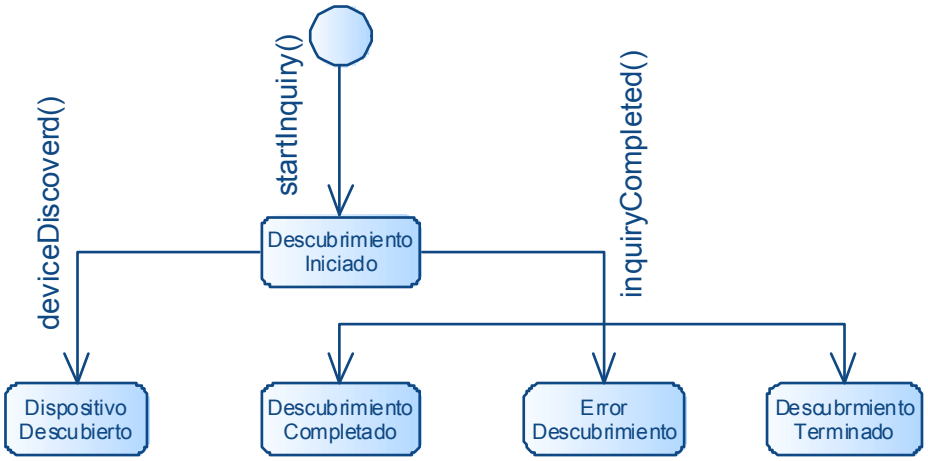

Figura 29. Diagrama de estados descubrimiento de dispositivos
(Adaptación de [29])

A continuación, se inicia el proceso de búsqueda de servicios a través del método `searchServices()` de la clase `DiscoveryAgent`. Este método, se encarga de invocar a los métodos `servicesDiscovered()` y `servicesSearchCompleted()` de la interfaz. Como resultado del proceso de búsqueda se obtienen los posibles estados resultantes, mostrados en el diagrama de la Figura 30.

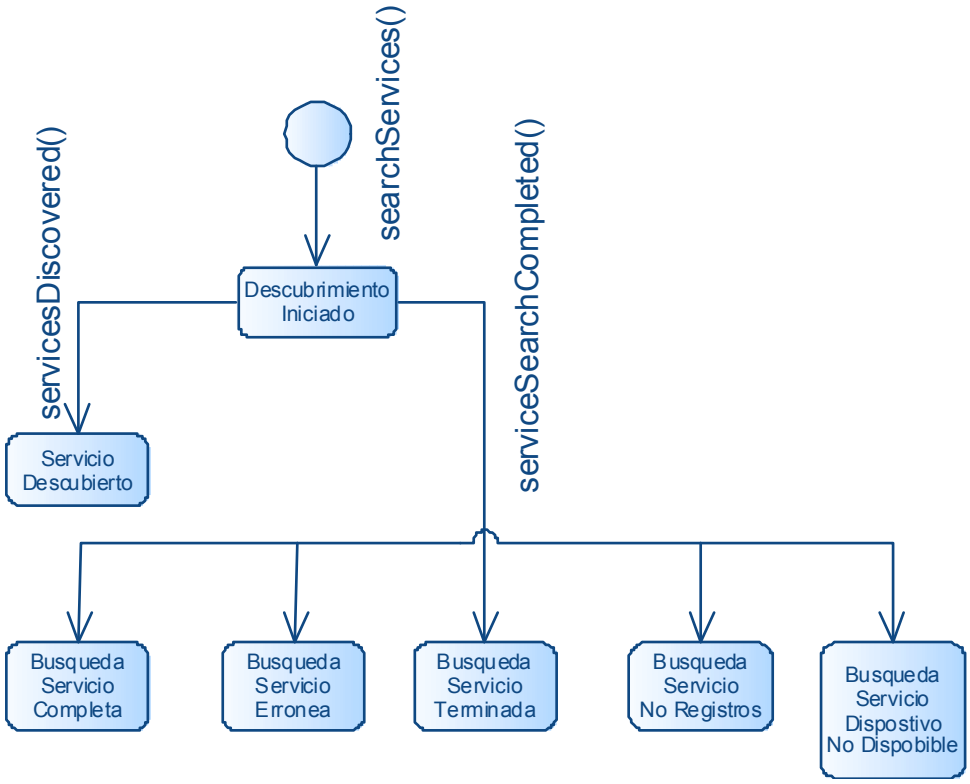

Figura 30. Diagrama de estados descubrimiento de servicios
(Adaptación de [29])

Una vez explicados los procesos de descubrimiento y búsqueda anteriores, en la Figura 30 se presenta el diagrama de actividad que engloba ambos procesos.

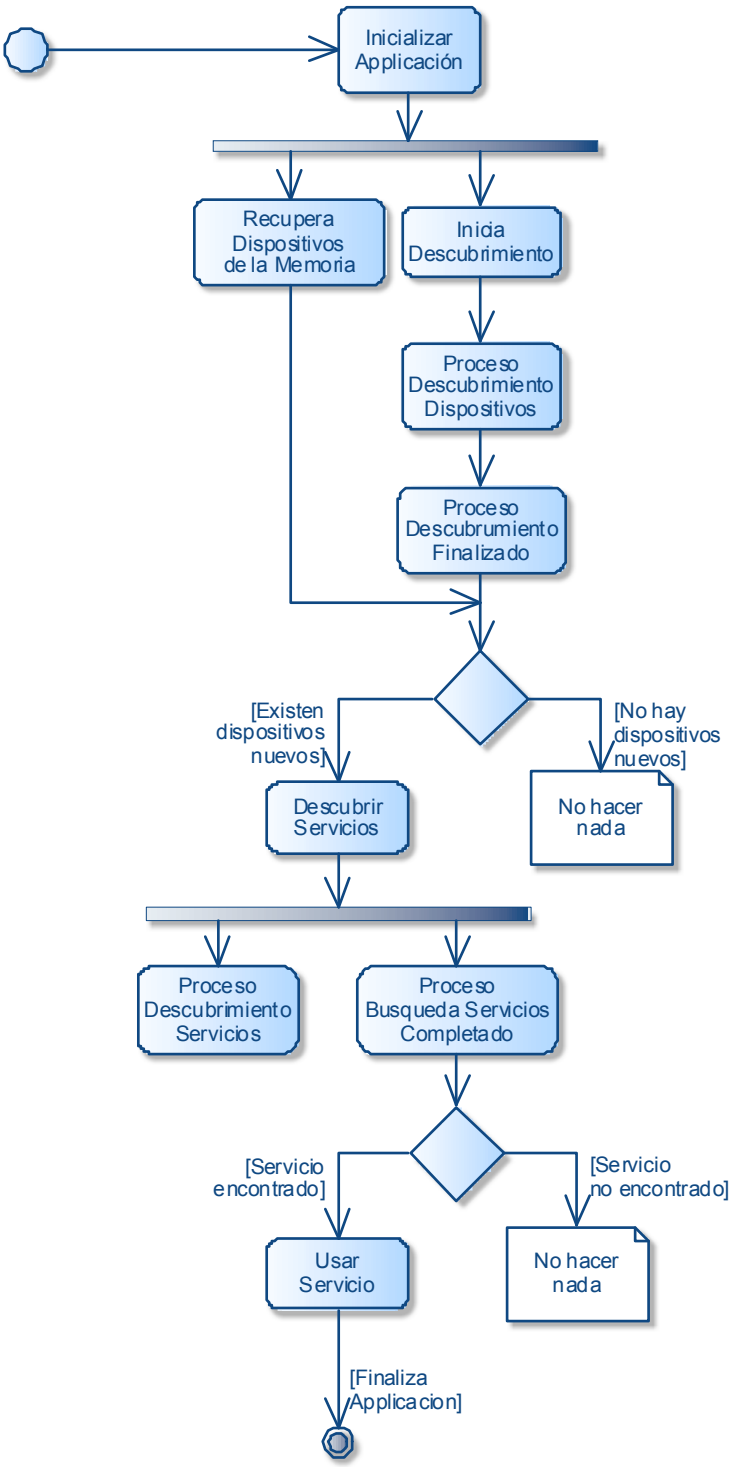

Figura 31. Diagrama de actividad interfaz Discovery Listener

A continuación se procede a explicar de forma más detallada el establecimiento de la conexión del cliente Bluetooth, mediante el flujo de código principal de la clase `HiloBluetooth` (para más información, véase Anexo B). El punto de arranque se establece con la llamada al método `start()`, donde se suceden la siguiente secuencia de acciones:

- Se instancia el objeto `dispositivoLocal`, de la clase `LocalDevice`, mediante el método `getLocalDevice()`.

```
dispositivoLocal = LocalDevice.getLocalDevice();
```

- Mediante el método `setDiscoverable()`, del objeto `dispositivoLocal`, se permite que el dispositivo sea visible para el resto, donde se le pasa como parámetro `DiscoveryAgent.GIAC`, que posibilita que la conectividad sea ilimitada.

```
dispositivoLocal.setDiscoverable(DiscoveryAgent.GIAC);
```

- Se instancia el objeto `discoveryAgent`, de la clase `DiscoveryAgent`, mediante el método `getDiscoveryAgent()`. A través de este objeto se realizará la búsqueda de dispositivos y servicios *Bluetooth*.

```
discoveryAgent = dispositivoLocal.getDiscoveryAgent();
```

- Se instancia la primera entrada del *array* `uuidSet`, de la clase `UUID`, con el identificador del servicio desplegado en el servidor.

```
uuidSet = new UUID[1];
uuidSet[0] = new UUID(0xABCD);
```

- Se llama al método `busquedaDispositivos()`, en el que se realizan las siguientes acciones:

    - Se llama al método `startInquiry()` del objeto `discoveryAgent`, pasándole como parámetro la instancia de la clase `HiloBluetooth` (`this`), ya que ésta implementa la interfaz `DiscoveryListener`.

      ```
      discoveryAgent.startInquiry(DiscoveryAgent.GIAC, this)
      ```

    - A través de este método se inicia la búsqueda de dispositivos, mediante métodos `deviceDiscovered()` y `inquiryCompleted()`, de la interfaz `DiscoveryListener`.

    - Finalizada la búsqueda, se devuelve el flujo de control al método `start()`.

- Si el resultado de la llamada al método `busquedaDispositivos()` ha sido satisfactorio, se realiza la llama al método `busquedaServicios()`, en el que se suceden las siguientes acciones:

    - Se realiza una iteración, donde se obtiene cada objeto dispositivo, almacenado en el *array* de vectores, llamado `dispositivos[]`, rellenado en el proceso de búsqueda de anterior. En cada iteración se instancia el objeto `dispositivo_remoto`, de la clase `RemoteDevice`, con cada uno de los dispositivos encontrados.

      ```
      RemoteDevice dispositivo_remoto = (RemoteDevice)
      dispositivos.elementAt(i);
      ```

    - A su vez, dentro de cada iteración se llama al método `searchServices()` del objeto `discoveryAgent`. A este método se le pasan tres parámetros, el identificador del servicio, el identificador del dispositivo remoto y la instancia de la clase `HiloBluetooth`. Del mismo modo que en la búsqueda de dispositivos con el método `startInquiry()`, con este método se inicia la búsqueda de servicios, mediante los métodos `servicesDiscovered()` y `serviceSearchCompleted()`, de la interfaz `DiscoveryListener`.

      ```
      discoveryAgent.searchServices (null, uuidSet,
      dispositivo_remoto, this);
      ```

    - Finalizada la búsqueda, se devuelve el flujo de control al método `start()`.

- Si el resultado de la llamada al método `busquedaServicios()` ha sido satisfactorio, llegado a este punto, solo queda establecer la conexión con el servidor. Para ello, se realizan las siguientes acciones

    - Se obtiene la *url* del servicio, almacenada previamente en el proceso de búsqueda de servicios, dentro del *array* de vectores llamado `servicios[]`.

      ```
      String URL = sr.getConnectionURL
      (ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);
      ```

    - Se obtiene la conexión con el servicio a través del objeto llamado conexión, de la clase `StreamConnection`, mediante el método `open()` de la clase `Connector`.

```
conexion = (StreamConnection) Connector.open(URL);
```

- Finalmente, se obtiene el flujo de salida de datos del servicio, a través del método `openDataOutpuStream()`, del objeto conexión instanciado previamente.

```
salida = conexion.openDataOutputStream();
```

- Una vez encontrado el servidor *Bluetooth*, y establecido conexión con el servicio desplegado, se instancia la tarea de ejecución periódica, donde se llama al método `enviaDatos()`. Este método se encarga de escribir los mensajes, en el flujo de salida de datos de la conexión.

```
salida.writeUTF(cadenaEnvio);
```

5.3.2. Gestión del acelerómetro

En esta sección se define la forma en la que se obtiene la información proporcionada por el acelerómetro y como ésta es tratada para ser enviada por el canal *Bluetooth* establecido. Este proceso se define dentro de la clase `HiloMovimiento` (para más detalle, véase Anexo B).

Como se ha mencionado ya en capítulos anteriores, *Java ME* proporciona un *API* específico (JSR-256) que interacciona con la interfaz de los sensores. A continuación en la Figura 32 se muestra el diagrama de clases del *API*.

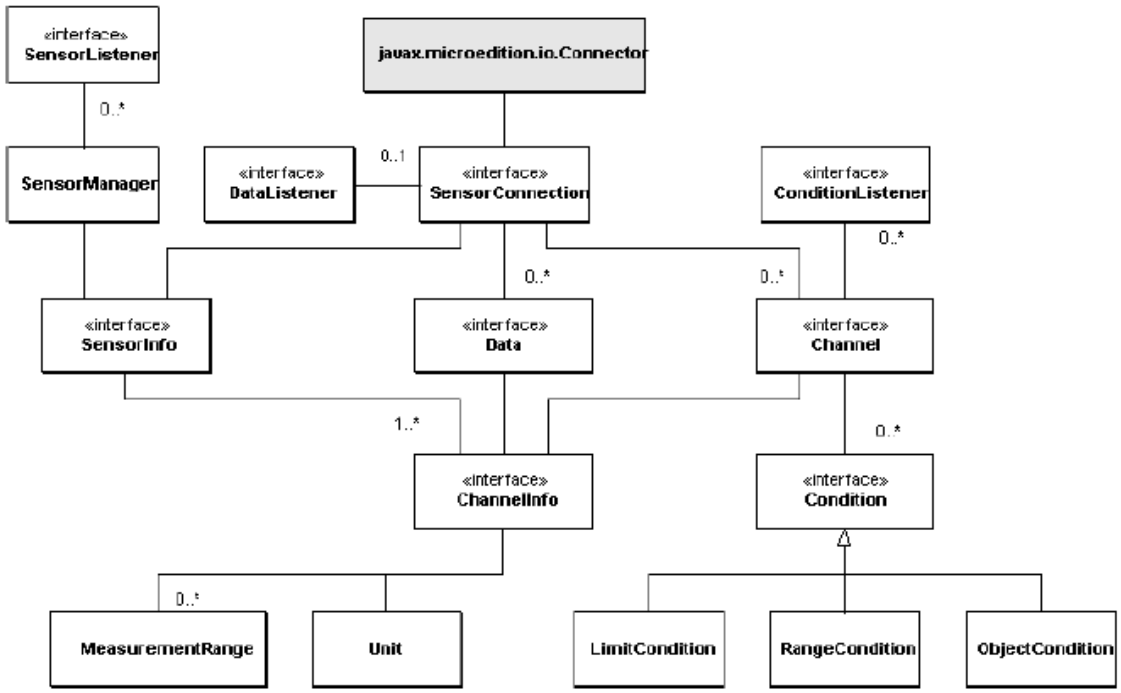

Figura 32. Diagrama de clases *API* JSR-256
[26]

La clase `HiloMovimiento` es la encargada de usar este *API* dentro del paquete `Paciente` (ver Figuras 26 y 27). Ésta implementa las interfaces del *API* `SensorConnection` y `SensorInfo`, con las que se establece la conexión con el sensor y se obtiene la información proporcionada por éste.

Mediante el método `findSensors`() de la interfaz `SensorInfo` se obtiene la información del sensor de aceleración. A través de la clase `Connector`, realizando un casting de la interfaz `SensorConnection` se obtiene la conexión con el sensor, como se muestra en la siguiente línea extraída del código:

```
connection = (SensorConnection) Connector.open(sensorAcc.getUrl());
```

Establecida la conexión con el sensor, se obtienen los datos de éste a través de la interfaz `Data` del siguiente modo:

```
Data[] data;
data = connection.getData(BUFFER_SIZE);
```

Finalmente, se realiza la llamada al nuevo método desarrollado `setPlainSensorData (data)`, ya dentro de éste se obtiene las coordenadas de cada eje, y se procesa los datos obtenidos a través del nuevo método `unscaleValue`() para obtener el valor real, en caso de que se le aplique una escala

al valor obtenido. Por último dentro del mismo método, se setean las coordenadas obtenidas dentro del objeto `posicionRaton` (instancia de la clase `PunteroRaton`, ver Figura 27) mediante el método `set_coordenadas()`, como se muestra en la línea del código siguiente:

```
posicionRaton.set_coordenadas(val);
```

Enlazando con la sección anterior, donde se definía el cliente *Bluetooth*, en el método `enviaDatos()` de la clase `HiloBluetooth` se obtiene el valor de las coordenadas como se muestra en la siguiente línea del código:

```
posicionRaton.get_coordenadas();
```

Según el funcionamiento descrito en esta y la anterior sección, el sistema de obtención de datos del acelerómetro y la conexión con el servicio *Bluetooth* funcionan de forma asíncrona. La aplicación *paciente* cuando arranca inicializa el objeto `HiloMoviemiento`, mientras que el `HiloBluetooth` no es instanciado hasta que el usuario selecciona el modo "*Control Remoto*" (según se indica en manual de usuario de la aplicación, véase Anexo A).

5.4. Conclusiones

Como ha sido mencionado en la introducción de este capítulo, la solución global del sistema delega la acción del "*click"* del ratón al servicio desarrollado en el ordenador. Como primera solución se ha decidido utilizar un pulsador externo (como los presentados en el capítulo del estado de la cuestión), no obstante existen otras posibles soluciones. Una de ellas sería realizar la acción a través de la detección de un periodo de un periodo prefijado en la configuración del servicio, en el que el puntero permanece inmóvil.

En cuanto al diseño del software, se ha seguido el modelo impuesto por la especificación JSR - 82*, de Java ME*. Esta tecnología ha quedado obsoleta hoy en día, de modo que sería interesante de cara al futuro migrar el sistema a una plataforma de diseño más actual.

6 Conclusiones y Trabajos Futuros

6.1. Introducción

En este capítulo se recogen las conclusiones expuestas a lo largo de toda la memoria, y, a continuación se presentan posibles trabajos futuros derivados de este proyecto.

6.2. Conclusiones

La tecnología escogida para el desarrollo de la nueva funcionalidad "*Control Remoto"* suscita una seria crítica para la implementación de la misma.

De los proyectos presentados en el estado de la cuestión sobre aplicaciones móviles, se concluye que en la fecha de presentación de los mismos (la mayoría entre 2007 y 2009), la tecnología con mayor proyección y con más adeptos, para el desarrollo de aplicaciones móviles, era *Java ME*.

La justificación mayoritaria expuesta del porqué elegir *Java ME* en lugar de otras tecnologías, se sustentaba en que *Java* ofrece una solución multiplataforma gracias al uso de su *máquina virtual*. De modo que, para la plataforma *Java ME* se implementaron una *máquina virtual* y una *API* específicas para el desarrollo de aplicaciones móviles, lo que posibilitó el poder codificar aplicaciones para cualquier plataforma hardware.

Sin embargo, hoy en día, *Java ME* no ha cubierto las expectativas creadas en sus inicios, quedándose cada vez más en desuso. El porqué, se debe a la aparición del nuevo sistema operativo *Android*. Éste, ha irrumpido de forma devastadora dentro de los sistemas operativos (de software libre) orientados a teléfonos móviles, siendo el más extendido dentro de los actuales *smartphones*.

Aun conociendo el estado de la cuestión referente a la tecnología *Java ME*, la imposición de la elección tecnológica viene marcada por la base de diseño del proyecto *Localiza*, motivo por el cual ha quedado fuera del ámbito de este proyecto la evaluación de otras posibles tecnologías más actuales para el desarrollo del mismo.

De este modo el sistema desarrollado queda limitado a un rango de dispositivos móviles en los que poder cargar el sistema, dispositivos que se van a quedar fuera de mercado tarde o temprano.

6.3. Trabajos Futuros

La crítica expuesta en el apartado anterior abre la posibilidad a un futuro proyecto, que consistiría en migrar el desarrollo completo del sistema a otra plataforma. De modo que la propuesta resultante sería: “Migración del sistema *Localiza* a la plataforma de desarrollo de aplicaciones móviles para *Android”*.

Así mismo, como proyectos futuros, de cara a la complementación del sistema *Localiza*, se plantean futuros trabajos relacionados con el ámbito de la e-salud. En este ámbito existen una gran cantidad proyectos ya implementados, de los cuales se podría se podría aprovechar mucho del trabajo ya realizado. Un proyecto de carácter general podría ser: “Integración de un sistema de monitorización de sensores para la medición de bioseñales dentro de la plataforma *Localiza*”, de este título a su vez se podrían derivar múltiples trabajos en función de lo ambicioso que se pretenda ser.

7 Bibliografía

[1] Luis Miguel Caballero Gonzalez, “Herramienta de Control de dispositivo móvil mediante un ordenador”, trabajo fin de máster, Escuela Universitaria de Ingeniería Técnica de Telecomunicación, Universidad Politécnica de Madrid, Junio, 2012.

[2] Grupo ACCEDO, “Accesibilidad en Windows XP”, Octubre 2009.

[3] Joaquim Fonoll y Antonio Sacco, “Accesibilidad de los sistemas operativos Windows y Linux”, Departamento Educación Generalitat de Cataluña,España y Universidad Abierta Interamericana de Buenos Aires, Argentina.

[4] Cifras INE. Boletín informativo del Instituto Nacional de Estadística, “Panorámica de la discapacidad en España. Encuesta de Discapacidad, Autonomía persona y situaciones de Dependencia. 2008”, Octubre, 2009.

[5] Laura Pedram Parra, “Desarrollo de servicios de valor añadido para una herramienta de localización de personas basada en telefonía móvil”, trabajo fin de máster, Escuela Universitaria de Ingeniería Técnica de Telecomunicación Universidad Politécnica de Madrid, Octubre, 2011.

[6] WiViK® on-screen keyboard (virtual keyboard) software, http://www.wivik.com/ (Consultado el 3 de junio de 2012)

[7] Tecnologías adaptativas aplicadas a discapacidades motrices, https://sites.google.com/site/tecnologiasadaptativas/ (Consultado el 3 de junio de 2012)

[8] http://shop.nuance.es/store/nuanceeu/es_ES/pd/productID.223074400/offerID.14815854509/pgm.79488100/Currency.EUR (Consultado el 3 de junio de 2012).

[9] http://www.bj-adaptaciones.com/catalogo/ratones?gclid=CJuWq_--srACFUxlfAodIiWdXg (Consultado el 3 de junio de 2012).

[10] “State Reference Centre for Personal Autonomy and Technical Aid”, http://www.ceapat.es/ceapat_06/servicios/unidad_demostracion/discapacidad_fisica/index.htm (Consultado el 3 de junio de 2012).

[11] Bluetooth SIG (Special Interest Group), https://www.bluetooth.org/Building/overview.htm (Consultado el 22 de febrero de 2012).

[12] Wi-Fi Alliance, http://www.wi-fi.org/discover-and-learn (Consultado el 15 de mayo de 2012).

[13] Leandro Flórez Aristazábal, "Bluelight. Sistema para el control de la intensidad de luz de lámparas incandescentes por medio de Bluetooth", proyecto fin de carrera, Universidad Antonio José Camacho, Facultad de Ingenierías, Santiago de Cali, Julio, 2009.

[14] D.David Barbolla Asenjo, "Desarrollo de una aplicación de comunicación Bluetooth entre una PDA y sensores", proyecto fin de carrera, Universidad Carlos III de Madrid, Julio, 2006.

[15] Juan Pablo Sevilla Martín y Pablo García Sanchez, "Ulfsark. Comunicación entre dispositivos Bluetooth", proyecto fin de carrera, Universidad de Granada, 2006.

[16] Antonio Ángel Botella Galindo, "Pasarela Bluetooth /GPRS para dispositivos móviles", Universidad de Málaga, Junio, 2007.

[17] Sergio Laguna García, "Gestor de contraseñas en un dispositivo móvil accesible por Bluetooth", proyecto fin de carrera, Universidad Autónoma de Barcelona, Septiembre, 2008.

[18] Miguel Sanchez Opazo, "Avisador médico basado en tecnología Bluetooth", proyecto fin de carrera, Universidad Politécnica de Cataluña, Julio, 2009.

[19] Javier A. Cuenca Retaña, "Sistema de Posicionamiento en Interiores mediante Balizas Bluetooth", proyecto fin de carrera, Universidad Pontificia de Comillas (ICAI), Madrid, 2010.

[20] http://www.android.com/developers/ (Consultado el 3 de junio de 2012).

[21] Núcleo de Arquitectura Bluetooth, https://www.bluetooth.org/Building/HowTechnologyWorks/Architecture/Overview.htm (Consultado el 22 de febrero de 2012).

[22] Arquitectura de Protocolos Bluetooth, http://www.seguridadmobile.com/bluetooth/especificacion-bluetooth/arquitectura-de-protocolo/index.html (Consultado el 2 de marzo de 2012.)

[23] Perfiles Bluetooth, https://www.bluetooth.org/Building/HowTechnologyWorks/ProfilesAndProtocols/Overview.htm (Consultado el 2 de marzo de 2012).

[24] Sergio Gálvez y Lucas Ortega Díaz, “Java a tope: J2ME (Java 2 Micro Edition)”, Universidad de Málaga, 2003.

[25] C. Enrique Ortiz, “Managing the MIDlet Life-Cycle with a Finite State Machine”, Agosto, 2004, http://developers.sun.com/mobility/midp/articles/fsm/ (Consultado el 2 de Febrero).

[26] JSR 256 Expert Group, “Mobile Sensor API: JSR 256”, Version1.0, Released: 2006-02-20.

[27] Alberto Gimeno Brieba, “JSR-82: Bluetooth desde Java”, 2004.

[28] Pedro Manuel Borches Juzgado, “Java 2 Micro Edition: Soporte Bluetooth”, Universidad Carlos III de Madrid, Marzo, 2004.

[29] “Using the Java APIs for Bluetooth Wireless Technology, Part 1 - API Overview”, http://developers.sun.com/mobility/apis/articles/bluetoothintro/ (Consultado el 2 de de marzo de 2012).

[30] Windows Driver Kit (WDK), http://msdn.microsoft.com/en-us/library/windows/hardware/gg487428.aspx (Consultado el 13 de marzo de 2012).

Anexo A: Manual de Usuario

A.1 Introducción

En este capítulo se detallan los pasos a realizar para arrancar el modo *"Control Remoto"* de la aplicación *paciente* del sistema *Localiza*. También se presentan los errores que puede lanzar la aplicación en caso de no lograr arrancar el modo correctamente.

Para más información acerca del funcionamiento de la aplicación paciente, se recomienda consultar en el capítulo de "Manual de Usuario" de la memoria del proyecto "*Desarrollo de servicios de valor añadido para una herramienta de localización de personas basada en telefonía móvil*" [5].

A.2 Activación/desactivación Control Remoto

Para el iniciar el modo "*Control Remoto*" el usuario debe realizar las siguientes acciones:

- Una vez arrancada la aplicación, seleccionar en el menú de entrada la opción que aparece con el nombre "Control Remoto" (ver Figura 33).
- A continuación, seleccionar la opción activar y pulsar el botón aceptar (ver Figura 34).
- Una vez pulsado el botón activar, la aplicación tardará unos segundos en arrancar. Una vez establecida la comunicación, aparecerá el siguiente mensaje por pantalla "Control Remoto ha sido activado con éxito" (ver Figura 35).

  En caso de querer parar el modo Control Remoto el usuario deberá:

- Seleccionar en el menú de entrada la opción que aparece con el nombre "Control Remoto" (ver Figura 33).
- A continuación, seleccionar la opción desactivar y pulsar el botón aceptar (ver Figura 34).
- Una vez pulsado el botón iniciar, aparece el siguiente mensaje por pantalla "Control Remoto ha sido desactivado" (ver Figura 36)

Figura 33. Menú principal aplicación *paciente*

Figura 34. Pantalla de inicio modo Control Remoto

Figura 35. Aviso confirmación control remoto activado

Figura 36. Aviso confirmación control remoto desactivado

A.3 Manejo de errores modo Control Remoto

En este apartado se detallan los posibles errores que puede lanzar la aplicación en caso de o bien no encontrar ningún dispositivo *Bluetooth*, o bien no encontrar el servicio deseado.

En caso de no poder establecer la comunicación con el servicio remoto del ordenador, la aplicación lanza una alarma imprimiendo por pantalla el mensaje "Imposible iniciar Control Remoto" (ver Figura 37). Para conocer con más detalle el motivo de dicha alarma el usuario puede consultar los mensajes de error escritos seleccionando la opción del menú principal "Mensaje de aplicación".

Figura 37. Mensaje de error modo Control Remoto

En la tabla III se describen los posibles mensajes que escribe la aplicación en caso de detectar un error al intentar acceder al modo "*Control Remoto*", también se detalla la causa y el modo de actuación ante dichos errores.

Mensaje de Error	Causa	Acción
"Error en el descubrimiento de dispositivos..."	Error interno aplicación.	Reiniciar aplicación.
"Ningún dispositivo encontrado"	Dispositivo *Bluetooth* remoto no iniciado.	Arrancar dispositivo *Bluetooth* del ordenador.
"No se pueden buscar servicios para: "	Error interno aplicación.	Reiniciar aplicación.
"No se ha encontrado ningún servicio"	Servicio remoto no iniciado.	Iniciar servicio remoto en el ordenador.
"Imposible abrir conexión"	Error conexión *Bluetooth.*	Reiniciar aplicación y el servicio remoto del ordenador.

Tabla III Errores de aplicación para el modo *Control Remoto*

Anexo B: Javadoc

B.1 Introducción

El *Javadoc* presentado a continuación, representa solo a las nuevas clases implementadas `HiloBluetooth`, `TareaHilo` y `PunteroRaton` y a la clase `HiloMoviemiento`. Para más detalle acerca de la aplicación paciente, se recomienda consultar el Anexo C de la memoria del proyecto "*Desarrollo de servicios de valor añadido para una herramienta de localización de personas basada en telefonía móvil*" [5]

B.2 Clase *HiloBluetooth*

Overview Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Paciente

Class HiloBluetooth

```
java.lang.Object
  └ Paciente.HiloBluetooth
```

All Implemented Interfaces:

javax.bluetooth.DiscoveryListener

```
public class HiloBluetooth
extends java.lang.Object
implements javax.bluetooth.DiscoveryListener
```

Clase que se encarga de implementar la interfaz bluetooth DiscoveryListener para el envio de datos de las coordenadas x,y,z del acelerómetro.

Author:

Alberto Egido Ros

Nested Class Summary

class	**HiloBluetooth.TareaHilo** Clase que ejecuta la tarea periódica para el envío de datos

Field Summary

Fields inherited from interface javax.bluetooth.DiscoveryListener

`INQUIRY_COMPLETED, INQUIRY_ERROR, INQUIRY_TERMINATED, SERVICE_SEARCH_COMPLETED, SERVICE_SEARCH_DEVICE_NOT_REACHABLE, SERVICE_SEARCH_ERROR, SERVICE_SEARCH_NO_RECORDS, SERVICE_SEARCH_TERMINATED`

Constructor Summary

HiloBluetooth(PacienteMIDlet paciente)
Constructor de la clase

Method Summary

boolean	**busquedaDispositivos**() Método que inicializa la busqueda de los dispositivos bluetooth
boolean	**busquedaServicios**() Método que inicializa la busqueda de los servicios bluetooth
void	**cancel**() Método que cierra la conexión bluetooth y mata el hilo de ejecución
void	**deviceDiscovered**(javax.bluetooth.RemoteDevice dispositivoRemoto, javax.bluetooth.DeviceClass dc) Método invocado por el sistema cuando se encuentra un dispositivo guarda el dispositivo encontrado
void	**enviaDatos**() Método que envía los datos por la conexión bluetooth
boolean	**establecerConexion**() Método que se encarga de establecer la conexión bluetooth con el servidor
void	**inquiryCompleted**(int discType) Método invocado por el sistema cuando el descubrimiento de dispositivos ha finalizado.

void	**lanzarAlarma**() Clase que lanza una alarma en caso de no poder arrancar el servicio control remoto
void	**servicesDiscovered**(int transID, javax.bluetooth.ServiceRecord[] servRecord) Método de la interfaz DiscoveryListener que realiza la busqueda de los servicios.
void	**serviceSearchCompleted**(int transID, int respCode) Método de la interfaz DiscoveryListener que finaliza y completa la busqueda de los servicios y libera el hilo
void	**start**() Método que inicializa y arranca el hilo para enviar los datos por bluetooth

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

HiloBluetooth

public HiloBluetooth(PacienteMIDlet paciente)

Constructor de la clase

Parameters:

`paciente` - MIDlet padre

Method Detail

cancel

public void cancel()

Método que cierra la conexión bluetooth y mata el hilo de ejecución

---

lanzarAlarma

public void lanzarAlarma()

Clase que lanza una alarma en caso de no poder arrancar el servicio control remoto

---

start

public void start()

Método que inicializa y arranca el hilo para enviar los datos por bluetooth

---

busquedaDispositivos

public boolean busquedaDispositivos()

Método que inicializa la busqueda de los dispositivos bluetooth

Returns:

true si se encuentra algún dispositivo *Bluetooth*

---

busquedaServicios

public boolean busquedaServicios()

Método que inicializa la busqueda de los servicios *Bluetooth*

Returns:

true si se encuentra el servicio

---

deviceDiscovered

public void deviceDiscovered(javax.bluetooth.RemoteDevice dispositivoRemoto,
javax.bluetooth.DeviceClass dc)

Método invocado por el sistema cuando se encuentra un dispositivo guarda el dispositivo encontrado

Specified by:

`deviceDiscovered` in interface
`javax.bluetooth.DiscoveryListener`

Parameters:

`dispositivoRemoto` - objeto RemoteDevice almacena el dispositivo encontrado

`dc` - objeto DeviceClass

---

inquiryCompleted

public void inquiryCompleted(int discType)

Método invocado por el sistema cuando el descubrimiento de dispositivos ha finalizado.

Recuerda el discType para su posterior evaluación.

Specified by:

`inquiryCompleted` in interface
`javax.bluetooth.DiscoveryListener`

Parameters:

`discType` - identificador del código del resultado de la busqueda

---

servicesDiscovered

public void servicesDiscovered(int transID,
javax.bluetooth.ServiceRecord[] servRecord)

Método de la interfaz DiscoveryListener que realiza la busqueda de los servicios.

Specified by:

`servicesDiscovered` in interface
`javax.bluetooth.DiscoveryListener`

Parameters:

`transID` - identificador de transacción

`servRecord` - array de servicios encontrados

---

serviceSearchCompleted

public void **serviceSearchCompleted**(int transID,
int respCode)

Método de la interfaz DiscoveryListener que finaliza y completa la busqueda de los servicios y libera el hilo

Specified by:

`serviceSearchCompleted` in interface
`javax.bluetooth.DiscoveryListener`

Parameters:

`transID` - identificador de transacción

`respCode` – identificador de respuesta

---

establecerConexion

public boolean establecerConexion()

Método que se encarga de establecer la conexión *Bluetooth* con el servidor

Returns:

true si se logra establecer la conexión

---

enviaDatos

public void enviaDatos()

Método que envía los datos por la conexión *Bluetooth*

B.3 Clase *TareaHilo*

Paciente

Class HiloBluetooth.TareaHilo

```
java.lang.Object
  └java.util.TimerTask
      └Paciente.HiloBluetooth.TareaHilo
```

All Implemented Interfaces:

java.lang.Runnable

Enclosing class:

HiloBluetooth

```
public class HiloBluetooth.TareaHilo
extends java.util.TimerTask
```

Clase que ejecuta la tarea periódica para el envío de datos

Constructor Summary

HiloBluetooth.TareaHilo()

Method Summary

void	**run**() Método que ejecuta el método que escribe los datos en el buffer de salida del canal *Bluetooth*

Methods inherited from class java.util.TimerTask

cancel, scheduledExecutionTime

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

HiloBluetooth.TareaHilo

public HiloBluetooth.TareaHilo()

Method Detail

run

public void **run**()

Método que ejecuta el método que escribe los datos en el buffer de salida del canal *Bluetooth*

Specified by:

`run` in interface `java.lang.Runnable`

Specified by:

`run` in class `java.util.TimerTask`

B.4 Clase *HiloMovimiento*

Paciente

Class HiloMovimiento

```
java.lang.Object
  └Paciente.HiloMovimiento
```

All Implemented Interfaces:

java.lang.Runnable

```
public class HiloMovimiento
extends java.lang.Object
implements java.lang.Runnable
```

Clase que se encarga de leer el valor del acelerómetro del terminal y de comprobar Si se ha producido una caída o si lleva mucho tiempo quieto

Author:

Laura Pedram Parra, Alberto Egido Ros

Constructor Summary

HiloMovimiento(PacienteMIDlet paciente) Constructor de la clase

Method Summary	
void	**cancel**() Metodo que cierra la conexión con el sensor
boolean	**detectarCaida**() Método que detecta si se ha producido una caída
int	**determinarPosicion**(double x, double y, double z)

	Método que devuelve la poscición en la que se encuentra el terminal móvil
`double`	**`getMaxAceleracion`**`(javax.microedition.sensor.Data[] data)` Método que devuelve la aceleración total máxima de los datos de los 3 ejes
`java.lang.String[]`	**`getSensorsInfoChannelsNames`**`(javax.microedition.sensor.SensorInfo sensorsInfo)` Método que detecta los nombres de canales validos para los sensores
`void`	**`getSensorsInfos`**`()` Método que recoge los sensores de tipo "acceleration" disponibles en el terminal
`void`	**`run`**`()` Método que ejecuta el hilo del tratamiento de los datos recogidos por el acelerómetro
`void`	**`setValorCoordenadas`**`(javax.microedition.sensor.Data[] data)` Método que setea las coordenadas de la clase PunteroRaton

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

HiloMovimiento

public HiloMovimiento(PacienteMIDlet paciente)

Constructor de la clase

Parameters:

`paciente` - MIDlet padre

Method Detail

getSensorsInfos

public final void getSensorsInfos()

Método que recoge los sensores de tipo "acceleration" disponibles en el terminal

---

getSensorsInfoChannelsNames

public java.lang.String[]
getSensorsInfoChannelsNames(javax.microedition.sensor.SensorInfo sensorsInfo)

Método que detecta los nombres de canales validos para los sensores

Parameters:

`sensorsInfo` - informacion del sensor para la detección del nombre de los canales

Returns:

array de string con el nombre de los canales

---

cancel

public void cancel()

Método que cierra la conexión con el sensor

---

run

public void run()

Método que ejecuta el hilo del tratamiento de los datos recogidos por el acelerómetro

Specified by:

`run` in interface `java.lang.Runnable`

---

detectarCaida

public boolean detectarCaida()

Método que detecta si se ha producido una caída

Returns:

true si se trata de una caída

---

getMaxAceleracion

public double getMaxAceleracion(javax.microedition.sensor.Data[] data)

Método que devuelve la aceleración total máxima de los datos de los 3 ejes

Parameters:

`data` - array de datos de los 3 ejes

Returns:

aceleración máxima

---

determinarPosicion

public int determinarPosicion(double x,
double y,
double z)

Método que devuelve la posición en la que se encuentra el terminal móvil

Parameters:

`x` - valor del eje X

`y` - valor del eje Y

`z` - valor del eje Z

Returns:

int número de la posición

---

setValorCoordenadas

public void setValorCoordenadas(javax.microedition.sensor.Data[] data)

Método que setea las coordenadas de la clase PunteroRaton

Parameters:

`data` - array de datos de los 3 ejes

B.5 Clase *PunteroRaton*

Overview Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Paciente

Class PunteroRaton

```
java.lang.Object
  └ Paciente.PunteroRaton
```

```
public class PunteroRaton
extends java.lang.Object
```

Clase que almacena las coordenadas del acelerómetro

Author:

Alberto Egido Ros

Constructor Summary

PunteroRaton()

Method Summary

java.lang.Double[]	get coordenadas()

	Método para obtener el valor de las coordenadas
void	**set_coordenadas**(java.lang.Double[] coordenadas) Método para setear el valor de las coordenadas

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

PunteroRaton

public PunteroRaton()

Method Detail

get_coordenadas

public java.lang.Double[] get_coordenadas()

Método para obtener el valor de las coordenadas

Returns:

array con el valor de coordenadas x,y,z

---

set_coordenadas

public void set_coordenadas(java.lang.Double[] coordenadas)

Método para setear el valor de las coordenadas

Parameters:

`coordenadas` - array con el valor de las coordenadas x,y,z
