Departamento de Telemática

PROYECTO FIN DE CARRERA

SISTEMA DE GESTIÓN DE TRANSPORTES

Autor: Juan Manuel Ardoy de Gracia

Tutor: Mario Muñoz Organero

Leganés, octubre de 2010

Título: SISTEMA DE GESTIÓN DE TRANSPORTES
Autor: Juan Manuel Ardoy de Gracia
Director: Mario Muñoz Organero

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 21 de octubre de 2010 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Agradezco a mi familia, la más directa y la más extensa, amigos y demás personas con las que comparto esta vida, a los que están y dejaron de estar.

También agradezco a mi actual pareja sentimental el saber apearse a un lado cuando este proyecto fin de carrera lo ha requerido.

Gracias por vuestro apoyo y por acompañarme en todo.

Resumen

El presente proyecto trata de proporcionar un proceso de calidad, basado en un equipamiento de nuevas tecnologías, para el funcionamiento formal de los transportes de cualquier empresa u organización que implique ciertos desplazamientos. Su misión es la integración en un sistema de las actividades de transporte o distribución, permitiendo la asignación de rutas a sus transportistas. La empresa que adopte este sistema podrá gozar en estas funciones de un mayor rendimiento que con cualquier otro modus operandi más "manual", y que seguramente se verá traducido en mayores beneficios.

Caben destacar las ventajas fundamentales que aporta el sistema desarrollado: la centralización de toda la información referente a estos desplazamientos en un Sistema de Información (SI) gestionado informáticamente (con acceso vía web), la divulgación de dicha información a sus responsables mediante comunicaciones inalámbricas y el manejo que estos disponen de ella a través de sus dispositivos móviles, mediante los cuales pueden registrar las incidencias y el estado resultante de las rutas que le han sido encomendadas.

Para logar todo esto han intervenido múltiples tecnologías. En concreto, tenemos desde la tecnología de Bases de Datos (BBDD) imprescindible para la manutención del SI (MySQL), pasando por la tecnología web empleada para su gestión (J2EE), hasta otras tecnologías de comunicación inalámbrica vigentes para su distribución (bluetooth y NFC), y concluyendo con la tecnología J2ME para el manejo de la misma en dispositivos móviles; contemplando, por tanto, la interoperatividad entre dispositivos móviles (protocolo NFCIP) con el servidor bluetooth (J2SE y librería BlueCove), y entre el servidor web (GlassFish) y ordenadores personales (a través de su navegador web).

Como se puede observar, todo el desarrollo está sustentado sobre productos de *Sun Microsystems*, a lo que se extienden algunas herramientas auxiliares como el protocolo NFCIP-1 facilitado por *Nokia* o la librería BlueCove aportada por *Google Code*. Así, se ve satisfecho el deseo en este proyecto de llevar a cabo un desarrollo software íntegro en Java con el propósito de que se vislumbre su potencial, su universalidad y versatilidad y de que se dé alcance a toda su variedad de entornos.

Palabras clave: Java, J2EE, J2ME, J2SE, MySQL, GlassFish, servlet, JSP, MIDP, Nokia, NFC, NFCIP, bluetooth, BlueCove, móvil, web, transporte, rutas.

Abstract

The aim of this project is to provide a quality process, based on state-of-the-art technologies, to improve the transport system for any company or organization. The goal is to integrate transport or distribution operations into a system, allowing routes to be assigned to its transport workers. Any company implementing this system will benefit from higher performance as compared to any other manual modus operandi, which will in turn yield higher profit.

The system developed in this project offers some key advantages: centralization of all information on an computationally managed Information System (accessible via web), distribution of that information through wireless communication to the managers and possibility of handling the information on mobile devices carried by the managers, by means of which any incidence can be registered and feedback on the status of the routes assigned can be provided.

Various technologies have been implemented in order to fulfill these objectives. Namely, these range from the Data Base and web technologies indispensable towards the upkeep (MySQL) and management (J2EE) of the Information System, respectively, to other current wireless communication technologies necessary for its distribution (bluetooth and NFC) and, finally, J2ME technology for management tasks on mobile devices; thus enabling interaction between mobile devices (NFCIP protocol) and the bluetooth server (J2SE and BlueCove library), and between the web server (GlassFish) and personal computers (through its web browser).

All aspects of the development have been carried out using *Sun Microsystems* products, which is also the case for some auxiliary tools such as NFCIP-1 protocol supplied by *Nokia*, or the BlueCove library supplied by *Google Code*. Thus, the goal of the project is fulfilled through developing software entirely on Java, in order that its potential, universality and versatility may be more readily envisioned and its wide variety of environments may be reached.

Keywords: Java, J2EE, J2ME, J2SE, MySQL, GlassFish, servlet, JSP, MIDP, Nokia, NFC, NFCIP, bluetooth, BlueCove, mobile, web, transport, routes.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y Objetivos

1.1 Introducción

El sistema PFCJMAG, producto de este proyecto, se puede considerar un sistema de gestión de transportes. Para entender mejor qué queremos decir con esto, vamos a dar unas definiciones iniciales de qué es un sistema en general y también de qué es el transporte:

- **Sistema**. Conjunto de medios interconectados (objetos, seres humanos, informaciones) utilizados en algún proceso dinámico con el fin de alcanzar los objetivos señalados, y afectados por factores internos y externos.
- **Transporte**. Se denomina transporte al traslado de personas o bienes de un lugar a otro. El transporte es una actividad fundamental de la logística que consiste en colocar los productos de importancia en el momento preciso y en el destino deseado. (1)

"*La ausencia de plan, método y orden para actuar e interrelacionarse nos coloca frente al caos. Sistema nos indica orden; lo opuesto a sistema es el caos.*"

Con esta premisa, es más sencillo comprender qué objetivo abarca el actual proyecto. Su misión es la integración en un sistema de las actividades de transporte o distribución, permitiendo la asignación de rutas a sus transportistas. De este modo, el presente proyecto trata de proporcionar un proceso de calidad, basado en un equipamiento de nuevas tecnologías, para el funcionamiento formal de los transportes de cualquier

empresa u organización que implique ciertos desplazamientos. En este ámbito, podemos comprender desde empresas proveedoras de algún producto que ofrezca la distribución entre sus clientes, o proveedoras de algún servicio que precise visitas de mantenimiento e inspecciones, o cualquier empresa que realice actividades comerciales a domicilio; empresas de transporte, tanto de mercancías como de pasajeros, empresas de mensajería y paquetería, empresas de reparto o de reposición de máquinas expendedoras o de fuentes de agua tan comunes en oficinas,…; hasta cualquier tipo de empresa que conlleve el movimiento de cierto personal entre diferentes lugares.

La principal ventaja planteada por este proyecto es la centralización de toda la información referente a estos desplazamientos en un Sistema de Información (SI) gestionado informáticamente (con acceso vía web), en el que las diferentes personas encomendadas para cada recorrido puedan registrar las incidencias y el estado resultantes. Pero realmente, la mayor ventaja que presenta el sistema es la divulgación de dicha información a sus responsables y el manejo que estos disponen de ella a través de dispositivos que han venido a irrumpir en nuestra vida cotidiana como son sus teléfonos móviles. Del mismo modo que es interesante dicha distribución y usabilidad de la información, se atiende al beneficio de su centralización.

Podemos incluir un esquema inicial de todo lo que comprende este proyecto, en el que se pueda apreciar la diferenciación de entornos definida para el desarrollo, y que se toma como pauta en la presente memoria, en los que se emplean las tres ediciones de la plataforma Java específicas para cada situación:

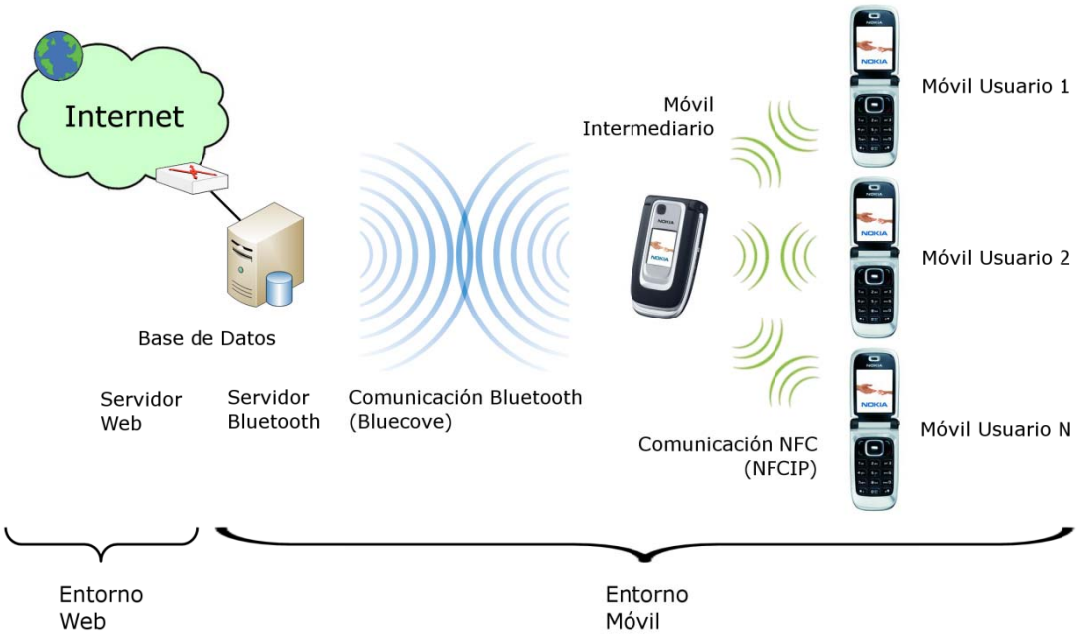


Figura 1 – Esquema de dimensionado del proyecto

Cada edición de la plataforma Java es utilizada por los siguientes componentes:

- **J2EE**. En el servidor web.
- **J2SE**. En el servidor bluetooth, junto con BlueCove.
- **J2ME**. En los dispositivos móviles, tanto el intermediario como los de usuarios.

En definitiva, existen dos grandes partes diferenciadas en el desarrollo del presente proyecto que a lo largo de la memoria describiremos por separado: web y móvil (esta última, como se puede observar, abarca desde el servidor bluetooth hasta los móviles de los usuarios).

1.2 Objetivos

Como mencionábamos al inicio, se pretende alcanzar un proceso normalizado para el transporte que emplee cualquier empresa, la cual podrá gozar en estas funciones de un mayor rendimiento que con cualquier otro modus operandis más "manual", y que seguramente se verá traducido en mayores beneficios. De este modo, el objetivo de este proyecto es el desarrollo de una solución software que facilite la actividad diaria de cualquier empresa vinculada al transporte.

Partiendo de un guión común que fácilmente compartirán estas empresas en su funcionamiento, la presente solución adopta un modelo de información genérico a la vez que abarca múltiples tecnologías para la gestión y distribución de la misma. En concreto, tenemos desde la empleada para la gestión del SI (servidor web), como otras tecnologías de comunicación inalámbrica vigentes para su distribución (bluetooth y NFC); contemplando, por tanto, la interoperatividad entre dispositivos móviles, servidor y ordenadores personales. Todo ello llevado a cabo bajo el sello de *Sun Microsystems* (recientemente comprada por *Oracle Corporation*), disponiendo de entre sus productos del gestor de base de datos MySQL y la plataforma de programación Java (en sus distintas ediciones J2SE, J2EE y J2ME), sobre la que se sustentan y se extienden paquetes o librerías aportadas por *Google Code* y *Nokia*, empleadas para el desarrollo de las comunicaciones inalámbricas (BlueCove y NFCIP, respectivamente).

1.3 Motivación personal

La motivación personal por la realización de este proyecto nace del deseo de llevar a cabo un desarrollo software íntegro en Java en el que se vislumbrara su potencial, haciendo patente la universalidad y versatilidad de este lenguaje mediante el uso de herramientas auxiliares y específicas para comunicaciones inalámbricas en capas de nivel superior a esta plataforma y dando alcance a toda su variedad de entornos. Al mismo tiempo, se ha pretendido aplicar todo este despliegue a una aplicación útil y efectiva en el mundo laboral.

Sí es cierto que, con la inclusión de dos tecnologías inalámbricas para una misma comunicación extremo a extremo, ha predominado la intención del desarrollo de diferentes tecnologías antes que la verdadera simplicidad funcional de la aplicación a nivel profesional.

1.4 Fases del desarrollo

La división de entornos manifiesta en la *Figura 1 – Esquema de dimensionado del proyecto* ha delimitado claramente dos subfases del desarrollo, al igual que hará con el resto de capítulos de la memoria.

De esta forma, tras las nociones básicas del diseño, tanto de la parte común (SI) como de ambos entornos, la primera subfase del desarrollo fue la que concierne a la implementación de la aplicación web, seguida de la segunda referente al entorno móvil, que abarca las aplicaciones móviles y el servidor bluetooth.

No obstante, antes de las etapas de desarrollo y de diseño, tuvo lugar un período de primeras implementaciones de prueba respecto al entorno móvil para una toma de contacto con el funcionamiento de las comunicaciones inalámbricas que en este se emplearían, lo cual podemos considerar propio de la etapa de análisis.

1.5 Medios empleados

1.5.1 Medios Hardware

Para el desarrollo de este proyecto ha resultado indispensable el uso del siguiente material:

- Un ordenador personal de sobremesa o portátil, con funcionalidad bluetooh.
- Dos móviles compatibles con bluetooth y NFC.

De este modo, hemos optado por los siguientes componentes:

- Un ordenador portátil Toshiba Satellite A200 – 1TP, con dispositivo bluetooth integrado; de propiedad del autor previamente a la realización del proyecto.
- Dos móviles Nokia NFC 6131, aportados por la Universidad Carlos III de Madrid en calidad de préstamo.

1.5.2 Medios Software

Debo confesar que el establecimiento del entorno de desarrollo ha suscitado auténticos quebraderos de cabeza en diferentes momentos a lo largo de todo el proyecto. En este punto se tratará de explicar los problemas acaecidos en este sentido, sin caer en un excesivo embrollo como ha resultado en la realidad. En principio, cabe señalar la decisión en el uso del IDE Eclipse (2) para todo el desarrollo, buscando la integración en esta herramienta de ambas partes del proyecto e incluyendo todas las facilidades al alcance mediante *plugins* o complementos útiles de cada una.

Según lo mencionado en el anterior apartado, hasta la finalización de la primera subfase del desarrollo habíamos logrado un entorno de desarrollo bastante estable e integrado. En este mismo, habíamos sido capaces de implementar previamente, como ya se ha dicho, ciertas pruebas de la parte móvil mediante el SDK dispuesto por Nokia para su dispositivo Nokia 6131 NFC. No obstante, a raíz de la iniciativa de mi tutor de proyecto sobre el simulador del Nokia 6212 que permitía probar conexiones NFCIP, la segunda subfase del desarrollo se torna en una variación algo radical del entorno de desarrollo que, sin embargo como ya veremos, no alcanza el propósito buscado.

Así, el entorno o los medios software empleados para el desarrollo quedan determinados por las diferentes subfases del desarrollo, de modo que vamos a repasar esta sección atendiendo a ambas fases. Finalmente, enumeramos cierto software de apoyo, aunque no imprescindible, para el desarrollo.

1.5.2.1 Primera fase del desarrollo

Dada la intención de utilizar la tecnología inalámbrica NFC en el ámbito móvil y la disposición de esta en el terminal Nokia 6131, sobre el que finalmente probaremos físicamente el funcionamiento del proyecto, nos acogimos en la denominada primera fase a la herramienta de desarrollo *Nokia 6131 NFC SDK 1.1*, accesible en *Forum Nokia* (3), por sus características descritas en la sección *2.3.5 SDKs de Nokia*.

Adelantar por el momento que corresponde a una herramienta de desarrollo para dispositivos móviles propietaria de *Nokia*, basada en J2ME y que implementa la especificación *JSR 257 v1.0* (*Contactless Communication API*), además de incluir tanto un simulador de terminal móvil (análogo al ofrecido por WTK, *Wireless ToolKit*, de la mano de *Sun Microsystems*) como de tarjetas externas o *tags* virtuales, para reproducir comunicaciones NFC entre ambos.

Pues bien, dicha herramienta condiciona el entorno de desarrollo en los siguientes términos (4):

- *Microsoft Windows XP* (*Service Pack 2*), en su versión en inglés[1].
- *Java™ 2 SDK version 1.4.2* o *1.5.0*.

El segundo requisito era, en esta primera fase, el más restrictivo limitando en versiones respecto a la plataforma Java. Decidimos utilizar la más moderna de las dos posibles, es decir, *Java 2 SDK versión 1.5.0* o, lo que es lo mismo en la nueva

[1] Respecto al primer requisito, resaltar que el Sistema Operativo sobre el que realizamos el desarrollo es el mismo que el indicado, en su versión *Professional*, pero en español. Y en efecto, tuvimos un pequeño problema dada la opción de instalar también un *plugin* que incorpora *Nokia 6131 NFC SDK 1.1* para Eclipse; y es que, a la hora de simular una aplicación móvil a través de la opción de dicho *plugin*, se reportaba un error de lectura del archivo .jad temporal que se genera en nuestro sistema de ficheros para dicha acción. Finalmente, y tras múltiples consultas en foros de Internet, descubrí el motivo por el que se requería la versión en inglés: la ruta temporal obtenida por dicho *plugin* de la variable de entorno TEMP del Sistema Operativo para alojar temporalmente el archivo que necesitaba contenía un carácter con tilde que no reconocía ("Configuración local"), accediendo de forma errónea a dicha ruta e interrumpiendo la ejecución del simulador. Así, solventamos este problema, y con ello el requisito de la versión inglesa del SO, tan sólo modificando en la variable de entorno el carácter problemático, quitándole la tilde, y generando otra carpeta que correspondiese con esta nueva ruta. Hilo de discusión al respecto en el foro de *Nokia*: http://discussion.forum.nokia.com/forum/showthread.php?111758-¿Posible-BUG-Nokia-SDK-plug-in-para-NFC&p=706042#post706042 [Último acceso: 11/10/2010 18:30h].

nomenclatura que comienza a raíz justo de esta versión, *Java SE JDK 5.0* (en su último *Update 21* al comienzo del proyecto) (5). Esta versión incluye su correspondiente *JRE 5.0*, y es lo que nos determinaba la plataforma a emplear en el otro ámbito del proyecto, la parte web:

- *Java EE 5 SDK Update 8*, con el servidor de aplicaciones *GlassFish Enterprise Server v2.1.1* (6)

Respecto a la integración con el IDE *Eclipse* las especificaciones nos llevaron a emplear la *versión 3.1.2*[2]. Con esta versión, gracias al *plugin WTP* en su *versión 1.0.3* correspondiente, integrábamos en la misma herramienta de desarrollo el ámbito web a la parte móvil sin problemas.

Para ir descubriendo el terreno sobre el que nos desenvolveríamos, en el período de pruebas previo al desarrollo ya mencionado, realizamos implementaciones para transferencias simples *NDEF*, sobre comunicación NFC, entre memoria interna del terminal y tarjetas externas. Una vez probada esta parte, mediante simulación, fuimos conscientes de que la comunicación que requeriríamos para el esquema de nuestro proyecto no sería entre móvil y tarjeta externa, sino entre dos móviles (o entre sus dos tarjetas internas). Este aspecto es el denominado uso P2P (ver punto *2.3.4.4 Usos y Aplicaciones*) implementado mediante el nombre NFCIP en una extensión de la especificación JSR 257 aportada por *Nokia*.

Además, también descubrimos respecto a la comunicación bluetooth que la especificación JSR 82, aportada como paquete opcional por la plataforma J2ME, sólo era específica entre dos dispositivos móviles, cuando en nuestro caso requeríamos entre móvil y ordenador. Así, empleamos BlueCove (ver *2.3.3.5 BlueCove*), que es la implementación del JSR 82 sobre J2SE, lo cual nos permite el funcionamiento de dicha especificación entre ambas ediciones de Java.

Como podemos ver, a raíz de unas premisas restrictivas del entorno de desarrollo marcadas por *Nokia*, decidimos el resto de versiones de dicho entorno en lo que respecta a la parte web. Así, el desarrollo del proyecto seguía en esas versiones delimitadas y sin dar ningún problema al finalizar el desarrollo de la aplicación web.

Con todo esto, pasamos a terminar de enumerar el entorno restante de la primera etapa de desarrollo:

- Librería *BlueCove*, implementación de JSR 82 para J2SE, en su *versión 2.1.0*. (7)
- *MySQL 5.1.45* en su versión gratuita (8) (la última al inicio del proyecto), como gestor de la Base de Datos (BD) del sistema.
- Librería *MySQL Connector / Java 5.1.12* (9) (la última versión al comienzo del proyecto).

[2] En realidad, el requisito es entre la *versión 3.1.1 ó 3.2*, según la documentación correspondiente (4). Además, integramos también al IDE Eclipse tras su instalación, y antes de la instalación e integración del *Nokia 6131 NFC SDK 1.1*, la herramienta *Carbide.j versión 1.5* (41) tal y como menciona dicha documentación.

1.5.2.2 Segunda fase del desarrollo

Con la propuesta de mi tutor de proyecto sobre el simulador del Nokia 6212 que permitía probar conexiones NFCIP, junto con la razonable intención de integrar todo el desarrollo en la misma herramienta, nos disponemos a adecuar la aplicación de Eclipse, y es donde surgen problemas de compatibilidad entre versiones del IDE y los nuevos *plugins*, a diferencia de lo ya consolidado en la primera etapa, dando lugar a una nueva confección del entorno de desarrollo en la segunda etapa no del todo integrado.

Así, cambiamos el planteamiento que en la anterior etapa nos marcaba desde un principio todo el entorno de desarrollo. Ahora, con la herramienta de desarrollo *S40 Nokia 6212 NFC SDK* (10), tenemos otros requisitos y especificaciones (11):

- Respecto a Java: se requiere *JRE 6*, pasando con ello al *JDK 6* (en donde está incluido) (12), y pudiendo emplear ahora tanto *J2EE 5* como *6* (con servidor *GlassFish 2.1.1* ó *3*, respectivamente).
- Respecto al IDE *Eclipse*: Se requiere la *versión 3.3.1*, además del *plugin eclipseME 1.7.6*. Y, para el desarrollo de proyectos J2EE, el *WTP 2.0.1* sería la versión correspondiente.

Con esto, la versión del J2EE que empleo en esta etapa es también la 5 con el servidor *GlassFish 2.1.1*, dada la buena experiencia y el desarrollo adquirido en la primera etapa.

Hasta aquí, nada ha empeorado con respecto a la primera etapa, únicamente tenemos versiones más recientes. El problema llega a la hora de incluir el servidor instalado por el J2EE en el IDE, ya con el WTP incorporado, para facilitar el desarrollo web. En la anterior etapa sí fui capaz de añadir dicho servidor GlassFish en el IDE Elipse, pero en esta ocasión, tras varios intentos manuales con los ficheros oportunos ofrecidos en la web (13) y automáticamente mediante la opción de *descarga de servidores*, no he logrado el resultado esperado.

En principio, el no conseguir incluir el servidor únicamente nos impide arrancarlo, comúnmente o en modo de depuración, pararlo y actualizar su despliegue web desde el propio IDE, pero sí podríamos generar el fichero WAR desde este y alojarlo manualmente en el servidor a través de su consola de administración (en formato web); al igual que en la parte móvil, donde finalmente generaremos paquetes exportables (jar y jad) a dispositivos físicos mediante eclipseME, prescindiendo del simulador.

Aun así, la decisión tomada en esta segunda etapa pasa a ser la de emplear dos IDEs diferentes para cada entorno, vulnerando la premisa de integrar todo el entorno en una sola herramienta y siendo esta herramienta aparentemente la misma pero duplicada. De esta manera, un IDE sería para la parte móvil tal y como el presentado hasta ahora con el *S40 Nokia 6212 NFC SDK*, y sin necesidad del *WTP 2.0.1* para la parte web; ya que esta parte la trabajaría con otro IDE proporcionado por GlassFish, que parece tomar la base de Eclipse pero integrando también las opciones para el desarrollo web, y se llama *glassfish-tools-bundle-for-eclipse 1.2*[3] (14).

[3] Para el correcto funcionamiento de esta herramienta tuve que hacer una variación en el texto de uno de los ficheros que la componen (eclipse.ini) del modo que encontré en la web (únicamente la primera línea de la parte *Configuración de las opciones de memoria*):

Respecto al resto del entorno, enumerado al final de la exposición de la primera fase, se conserva exactamente igual en esta segundo período.

Finalmente, tras el desarrollo de la parte móvil en el entorno descrito para esta fase, resulta frustrante reconocer que la funcionalidad P2P del simulador Nokia 6212, por la que variamos todo el entorno de desarrollo, no conseguimos que se ejecutara correctamente. Por consiguiente, tras tanto quebradero de cabeza en el cambio del entorno de desarrollo, no logramos la mejoría que pretendíamos respecto a la simulación del entorno móvil; es más, lo único que conseguimos fue desintegrar el entorno de desarrollo en lo que respecta a ambas partes, web y móvil.

Aún así, respecto a la plataforma Java básica de ejecución del proyecto, J2SE, que requieren ambas SDKs de *Nokia*, y también la parte web, logramos actualizarla en versión (de la 5 a la 6). Como resultado, hemos podido comprobar el correcto funcionamiento del entorno móvil generando paquetes exportables (jar y jad) de las aplicaciones móviles, gracias al *plugin* eclipseME, que hemos traspasado vía bluetooth y ejecutado en los dispositivos Nokia 6131; sin dar ningún problema en el sentido de que la versión J2SE y el SDK de Nokia empleados finalmente correspondían al *S40 Nokia 6212 NFC SDK*.

1.5.2.3 Otro software de apoyo

En este punto mencionaremos cierto software que nos ha facilitado el diseño y desarrollo:

- *Dezing for Databases v5*, herramienta para el diseño de modelos de datos y *scripting* SQL, empleada al inicio del desarrollo.
- *MySQL WorkBench*, análoga a la anterior pero con mayor funcionalidad específica para MySQL adoptada posteriormente en el desarrollo (dada la expiración del período de prueba de la anterior).
- *SmartDraw 2008*, herramienta gráfica para la realización de diagramas UML y otros.

1.6 Estructura de la memoria

Para facilitar la lectura de la memoria, se incluye a continuación un breve resumen de cada capítulo:

- **Capítulo 1 – Introducción y objetivos**. Como hasta aquí se se ha visto, este capítulo plantea de qué trata el proyecto, el problema que pretende abordar y, en función de ello, los objetivos perseguidos. Luego, es razonable relacionar este capítulo con la etapa de análisis del proyecto, aunque también se incluye un esquema del plantemiento tecnológico adoptado (*Figura 1 – Esquema de*

---

http://wikis.sun.com/display/sunidmdev/Setting+Up+the+IDE+Plugin+for+Eclipse#SettingUptheIDEPluginforEclipse-DownloadEclipse3.3.1%2C%22EclipseIDEforJavaEEDevelopers%22 [Último acceso: 11/10/2010 18:30h].

dimensionado del proyecto). Al mismo tiempo, se han presentado unos apartados referentes ya a las fases del desarrollo llevado a cabo y a los medios empleados.

- **Capítulo 2 – Estado del Arte**. En este capítulo se van a explicar las tecnologías que se han usado para diseñar e implementar el sistema objeto de esta memoria, denominado PFCJMAG.
- **Capítulo 3 – Descripción del Sistema: Alto Nivel**. Con este capítulo comienza la descripción del sistema, resolviendo asuntos y tomando decisiones concernientes al diseño. Por el momento, las únicas figuras que se incluyen ilustran el modelo de datos adoptado, tan crucial para el diseño completo. Para finalizar el capítulo, se ilustra un caso de uso bastante detallado y específico, que puede facilitar la compresión sobre el fin perseguido por el proyecto y las vías a seguir hasta él.
- **Capítulo 4 – Descripción del Sistema: Bajo Nivel**. Este capítulo, con el que se culmina la descripción del sistema, viene a reflejar el resultado de la puesta en práctica del diseño planteado en el anterior capítulo, revelando los impedimentos encontrados para llevarlo a cabo y haciendo una revisión más a fondo de su versión final, con la ayuda conceptual de diagramas UML tanto de paquetes y clases como de comportamiento. También se hace hincapié sobre las particularidades y detalles más destacables surgidos a raíz de la etapa de implementación y desarrollo.
- **Capítulo 5 – Pruebas**. En todo proyecto de ingeniería software es imprescindible un apartado de pruebas en el que se verifique la robustez de las aplicaciones y que todo funcione según lo deseado.
- **Capítulo 6 – Conclusiones y trabajos futuros**. En este último capítulo se da pie a innumerables trabajos futuros para el sistema presentado, que trata de ser una base sólida para otros posibles proyectos de mejora con una definición y unas especificaciones más precisas.
- **Glosario – Acrónimos**. En esta parte de todo documento formal es donde se definen y comentan ciertos términos utilizados en el texto, en este caso siglas y acrónimos, con el fin de ayudar al lector a comprender mejor sus significados.
- **Referencias – Bibliografía**. A lo largo del texto se irán encontrando referencias, con formato numérico y entre paréntesis, que aluden a esta parte final del texto para facilitar una fuente bibliográfica.
- **Anexo A – Implantación en Cliente**. Esta es la guía que deberá seguir cualquier cliente que desee instaurar y poner en marcha el sistema sobre el equipamiento indicado.
- **Anexo B – Operaciones con Base de Datos**. Este anexo muestra el detalle de implementación de todas las transacciones SQL que puede efectuar el sistema.
- **Anexo C – Memoria Económica**. En este capítulo se expone el presupuesto del proyecto. Para ello, se hace un seguimiento de todas las etapas que ha abarcado su realización, un desglose de costes de personal, costes de material y costes totales; con lo que, finalmente, se alcanza un presupuesto.

Capítulo 2

Estado del Arte

Este capítulo hace referencia a una de las primeras etapas en la realización de un proyecto, la cual permite determinar cómo ha sido tratado y cómo se encuentra el asunto en cuestión hasta el momento, y cuáles son las tendencias actuales. Respecto a la utilidad pretendida por el proyecto es de suponer que habrá sido tratada, antes de posibles adopciones tecnológicas, de manera parecida a la que plantearemos, pero mediante el empleo de otros medios más tradicionales como, por ejemplo, fichas o plantillas escritas de puño y letra, de las cuales se podría cuestionar su practicidad. En el momento en el que recurrimos a la tecnología de hoy en día para la aplicación objeto del proyecto es cuando el estado del arte debemos trasladarlo a ésta misma.

Existen hoy día múltiples soluciones más complejas que abarcan el objetivo perseguido por este proyecto, y mucho más. Estas soluciones no escatiman en recursos y se pueden encontrar con el nombre de Sistemas Logísticos (15).

Tras este matiz, procederemos a definir las tecnologías empleadas para el desarrollo y el soporte para implementarlas, dando una explicación más o menos extensa en cada caso y diferenciando la correspondencia a uno u otro de los ambientes citados ya en la introducción: web y móvil. Como base en la elección del soporte de desarrollo se ha decidido, mientras fuera posible, que sea de código abierto, ya que no tienen un coste añadido de utilización y cuentan siempre con una gran comunidad de desarrolladores dispuestos a ayudar, incluso para futuras mejoras.

Así pues, dado que elegimos una plataforma común para ambos ambientes, comenzaremos con el estado del arte de dicha plataforma, además de describir el resto de herramientas utilizadas de forma conjunta en ambos entornos.

2.1 Plataforma base y tecnologías comunes

2.1.1 Java

Java es un lenguaje de programación orientado a objetos desarrollado por *Sun Microsystems* (16), recientemente adquirida por *Oracle Corporation* (17), a principios de los años 90.

Los lenguajes estructurados se basan en estructuras de control de bloques de código y subrutinas independientes que soportan recursividad y variables locales. La programación orientada a objetos (POO) toma las mejores ideas de la programación estructurada y las combina con nuevos conceptos de organización, permitiendo descomponer un programa en grupos relacionados. Cada subgrupo pasa a ser un objeto autocontenido con sus propias instrucciones y datos.

Por otro lado, la existencia de distintos tipos de procesadores y ordenadores llevó a la conclusión de que era muy importante conseguir un software que no dependiera del tipo de procesador utilizado. Se planteó la necesidad de conseguir código capaz de ejecutarse en cualquier tipo de máquina. Una vez compilado no debería ser necesaria ninguna modificación por el hecho de cambiar de procesador o de ejecutarlo en otra máquina. La clave consistió en desarrollar un código neutro el cual estuviera preparado para ser ejecutado sobre una máquina virtual denominada *Java Virtual Machine* (JVM).

De esta guisa, Java fue diseñado con el propósito de crear un lenguaje que pudiera funcionar en redes computacionales heterogéneas (redes formadas por más de un tipo de computadora) y que fuera independiente de la plataforma en la que se fuera a ejecutar. Java funciona de la siguiente manera: el compilador de Java deja el programa en un pseudocódigo (ficheros con extensión .class, no es código máquina) y luego el intérprete de Java ejecuta el programa (la ya mencionada *Java Virtual Machine* o JVM). Por eso Java es multiplataforma: existe un intérprete para cada máquina diferente.

Sun describe el lenguaje Java como "simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico". En esta línea, las características de diseño del lenguaje se encuentran del siguiente modo:

- **Familiar**. Como la mayoría de los programadores están acostumbrados a programar en C o en C++, la sintaxis de Java es muy similar al de estos. Java no sería un lenguaje totalmente nuevo, así que no le sería tan complicado recalar en los programadores escépticos.
- **Simple**. Los diseñadores de Java trataron de mantener las facilidades básicas del lenguaje en un mínimo y proporcionar un gran número de extras con las librerías

de clases. Elimina la complejidad de los lenguajes como C y da paso al contexto de los lenguajes modernos orientados a objetos. La filosofía de programación orientada a objetos es diferente a la programación convencional.

- **Orientado a Objetos**. Para que un lenguaje pueda considerarse orientado a objetos debe soportar como mínimo las características de:
  - Encapsulación.
  - Herencia.
  - Polimorfismo.
  - Enlace dinámico.
- **Robusto**. Uno de los problemas más comunes en los lenguajes de programación es la posibilidad de escribir programas que pueden bloquear el sistema. Algunas veces este bloqueo puede ser inmediato, pero en otras ocasiones llega a aparecer inesperadamente porque, por ejemplo, la aplicación accede a zonas de memoria que no estaban siendo ocupadas por otros programas hasta ese momento. Un ejemplo claro de lenguaje no robusto es C. Al escribir código en C o C++ el programador debe hacerse cargo de la gestión de memoria de una forma explícita, solicitando la asignación de bloques a punteros y liberándolos cuando ya no son necesarios.

  El sistema de Java maneja la memoria de la computadora por ti. Los punteros, la aritmética de punteros y las funciones de asignación y liberación de memoria no existen. En lugar de los punteros, se emplean referencias a objetos, los cuales son identificadores simbólicos. El gestor de memoria de Java lleva una contabilidad de las referencias a los objetos. Cuando ya no existe una referencia a un objeto, éste se convierte en candidato para la recogida de basura (*garbage collection*). No te tienes que preocupar por punteros o memoria que no se esté utilizando, etc. Java realiza todo esto sin necesidad de que uno se lo indique.
- **Seguro**. Se pretendía construir un lenguaje de programación que fuese seguro, esto es, que no pudiera acceder a los recursos del sistema de manera incontrolada. Por este motivo añadido, se eliminó la posibilidad de manipular la memoria mediante el uso de punteros y la capacidad de transformar números en direcciones de memoria (tal y como se hace en C), evitando así todo acceso ilegal a la memoria. Esto se asegura mediante el compilador Java que efectúa una verificación sistemática. El sistema de Java tiene además ciertas políticas que evitan que se puedan codificar virus con este lenguaje. Existen muchas restricciones que limitan lo que se puede y no se puede hacer con los recursos críticos de una computadora.
- **Portable**. El principal objetivo de los diseñadores de Java, y dado el gran crecimiento de las redes en los últimos años, fue el de desarrollar un lenguaje cuyas aplicaciones una vez compiladas pudiesen ser inmediatamente ejecutables en cualquier máquina y sobre cualquier sistema operativo. Por ejemplo, un programa desarrollado en Java en una estación de trabajo Sun, que emplea el sistema operativo Solaris, debería poderse llevar a un PC que utilice el sistema operativo Windows NT. Tal y como el código compilado de Java (conocido como *bytecode*) es interpretado, un programa compilado de Java puede ser utilizado por cualquier computadora que tenga implementado el intérprete de Java.
- **Independiente de la arquitectura**. Al compilar un programa en Java, el código resultante es un tipo de código binario conocido como *bytecode*. Este código es interpretado por diferentes computadoras de igual manera, solamente hay que implementar un intérprete para cada plataforma. De esa manera Java logra ser un lenguaje que no depende de una arquitectura computacional definida.

- **De alto rendimiento (Multithreaded)**. Una de las características del lenguaje es que soporta la concurrencia a través de *threads* (hilos). Un lenguaje que soporta múltiples *threads* es un lenguaje que puede ejecutar diferentes líneas de código al mismo tiempo. En ocasiones, puede interesarnos dividir una aplicación en varios flujos de control independientes, cada uno de los cuales lleva a cabo sus funciones de manera concurrente. Cuando los distintos flujos de control comparten un mismo espacio lógico de direcciones, se denominan *threads*.
- **Interpretado**. Java corre en máquina virtual, por lo tanto es interpretado.
- **Dinámico**. Java no requiere que compiles todas las clases de un programa para que este funcione. Si realizas una modificación a una clase, Java se encarga de realizar un *Dynamic Bynding* o un *Dynamic Loading* para encontrar las clases.

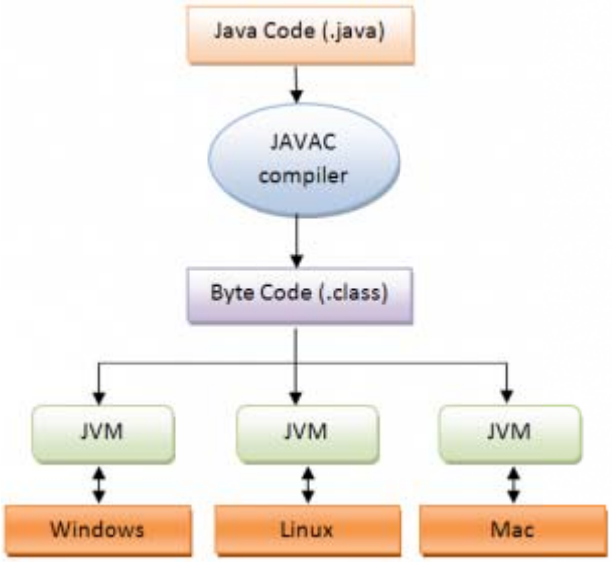


Figura 2 – Ejecución en Java

Java ha desarrollado tres ediciones para diferentes entornos: J2SE, J2EE y J2ME (ediciones Estándar, *Enterprise* y Micro, respectivamente). Más adelante, veremos en detalle cada una de ellas.

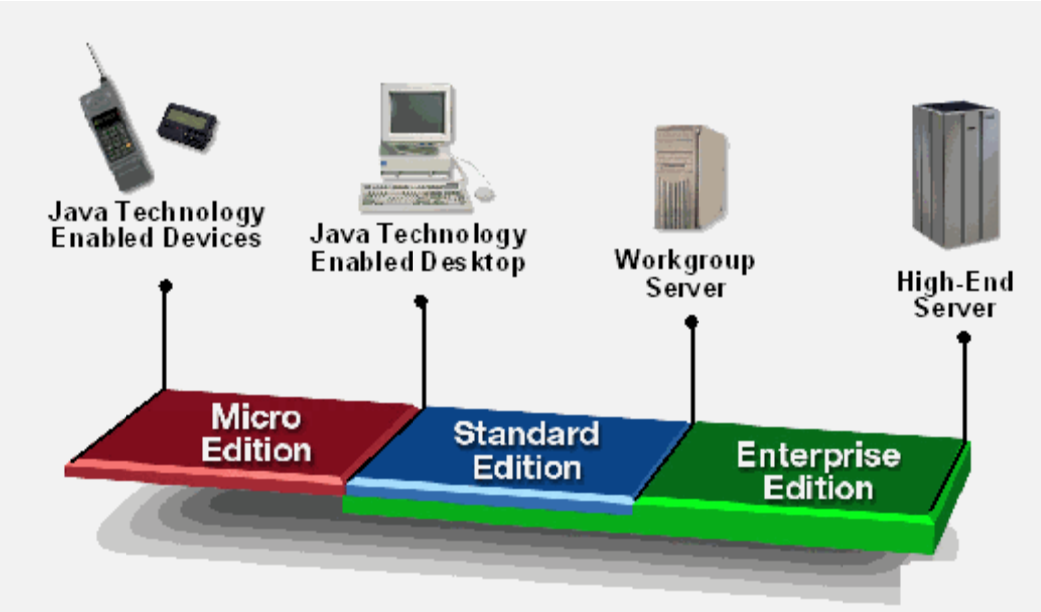


Figura 3 – Ediciones de Java

2.1.2 MySQL

MySQL es un sistema de gestión de BBDD relacionales, multihilo y multiusuario que utiliza el estándar SQL como lenguaje de acceso. Con multihilo se quiere decir que es capaz de atender varias peticiones al mismo tiempo, y el soporte multiusuario nos da la posibilidad de crear varias cuentas en el sistema con diferentes permisos.

MySQL es una idea originaria de la empresa de software libre *MySQL AB*, establecida inicialmente en Suecia en 1995 y cuyos fundadores son David Axmark, Allan Larsson, y Michael "Monty" Widenius. El objetivo que persigue esta empresa consiste en que MySQL cumpla el estándar SQL, pero sin sacrificar velocidad, fiabilidad o usabilidad.

MySQL AB – desde enero de 2008 una subsidiaria de *Sun Microsystems* y ésta a su vez de *Oracle Corporation* desde abril de 2009 – desarrolla MySQL como software libre en un esquema de licencia dual. Por un lado, se ofrece bajo la Licencia Pública General de GNU (GNU GPL) para cualquier uso compatible con esta licencia, pero para aquellas empresas que quieran incorporarlo en productos privativos deben adquirir una licencia específica que les permita este uso.

MySQL es propietario y está patrocinado por la empresa privada citada que posee el copyright de la mayor parte del código. Esto es lo que posibilita el esquema de licenciamiento anteriormente mencionado. Además de la venta de licencias privativas, la compañía ofrece soporte y servicios. Para sus operaciones contratan trabajadores alrededor del mundo que colaboran vía Internet.

MySQL es software de código abierto. Código abierto significa que es posible para cualquier persona usarlo y modificarlo: cualquier persona puede obtener el código fuente de MySQL y usarlo sin pagar, cualquier interesado puede estudiar el código fuente y ajustarlo a sus necesidades; todo según la definición de su licencia GNU GPL para

diferentes situaciones. La licencia GNU GPL de MySQL obliga a que la distribución de cualquier producto derivado (aplicación) se haga bajo esa misma licencia. Si un desarrollador desea incorporar MySQL en su producto pero desea distribuirlo bajo otra licencia que no sea la GNU GPL, puede adquirir una licencia comercial de MySQL que le permitirá hacer justamente eso.

Está desarrollado en su mayor parte en ANSI C (mezcla de C y C++). El Software de la base de datos de MySQL es un sistema cliente-servidor que consiste en un servidor SQL que soporta diferentes *backend*, varios programas de cliente y librerías, aplicaciones administrativas y un amplio rango de APIs. Además, el servidor MySQL puede emplearse como una librería que se puede integrar en la aplicación para hacerla más pequeña, más rápida y un producto autónomo fácil de manejar.

Respecto a SQL, Lenguaje de Consulta Estructurado, fue comercializado por primera vez en 1981 por IBM, el cual fue presentado a ANSI (Instituto Nacional Estadounidense de Estándares) y desde entonces ha sido considerado como un estándar para las bases de datos relacionales. Desde 1986, el estándar SQL ha aparecido en diferentes versiones como por ejemplo: SQL:92, SQL:99, SQL:2003.

MySQL es un sistema de administración relacional de bases de datos. Una base de datos es una colección estructurada de tablas que contienen datos. Estos pueden ser desde una simple lista de compras, a una galería de pinturas o el vasto volumen de información en una red corporativa. El término relacional hace mención a que archiva datos en tablas separadas, en vez de colocar todos los datos en un gran archivo; esto permite velocidad y flexibilidad. Las tablas están conectadas por relaciones definidas que hacen posible combinar datos de diferentes tablas para una consulta. Para agregar, procesar y acceder a datos guardados en un computador se necesita un administrador o gestor como el servidor MySQL. Dado que los ordenadores son muy buenos manejando grandes cantidades de información, los administradores de bases de datos juegan un papel central en computación, como aplicaciones independientes o como parte de otras aplicaciones.

MySQL es muy utilizado en aplicaciones web. MySQL es muy rápido en la lectura, pero puede provocar problemas de integridad en entornos de alta concurrencia en la modificación. En aplicaciones web hay baja concurrencia en la modificación de datos y, en cambio, el entorno es intenso en lectura de datos, lo que hace a MySQL ideal para este tipo de aplicaciones.

MySQL es el gestor de bases de datos de código abierto más popular mundialmente, por lo que se ha convertido en uno de los pilares de las aplicaciones web construidas con LAMP (Linux, Apache, MySQL, PHP/Perl/Python). MySQL puede ejecutarse en más de 20 plataformas incluyendo Linux, Windows, OS/X, HP-UX, AIX, Netware, etc.

2.1.3 JDBC

JDBC (*Java DataBase Connectivity*) es una especificación de un conjunto de clases y métodos de operación que permiten a cualquier programa Java acceder a sistemas de bases de datos de forma homogénea. La aplicación de Java debe tener acceso a un controlador (*driver*) JDBC adecuado. Este controlador es el que implementa la funcionalidad de todas las clases de acceso a datos y proporciona la comunicación entre

el API JDBC y la base de datos real. De manera muy simple, al usar JDBC se pueden hacer tres cosas:

- Establecer una conexión a una fuente de datos (por ejemplo, una base de datos).
- Mandar consultas y sentencias a la fuente de datos.
- Procesar los resultados.

Toda la conectividad a bases de datos de Java se basa en sentencias SQL, por lo que se hace imprescindible un conocimiento adecuado de SQL para realizar cualquier clase de operación con bases de datos.

2.1.4 Eclipse

Eclipse es una plataforma de software de código abierto para desarrollar en Java, creada originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos, capacidades y servicios complementarios.

La definición que da el proyecto Eclipse acerca de su software es: "una especie de herramienta universal, un IDE abierto y extensible para todo y nada en particular". Esta plataforma es considerada un Entorno de Desarrollo Integrado (IDE) ya que con ella es posible escribir código Java, compilarlo y ejecutarlo sin tener que cambiar de aplicación. Eclipse dispone de un editor de texto con resaltado de sintaxis, la compilación es en tiempo real, tiene pruebas unitarias con JUnit, control de versiones con CVS, integración con Ant, asistentes (*wizards*) para creación de proyectos, clases, tests, etc. y refactorización (técnica de ingeniería de software para reestructurar código fuente, alterando su estructura interna sin cambiar su comportamiento externo).

Eclipse emplea módulos (*plugins*) para proporcionar toda su funcionalidad al frente de la plataforma, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software. Como ya se ha indicado, hemos utilizado los *plugins* que nos han convenido para cada entorno de desarrollo; sin embargo, no ahondaremos sobre el estado del arte de estos mismos.

En cuanto al desarrollo, Eclipse provee al programador marcos de trabajo o perspectivas muy atractivas para el desarrollo de aplicaciones gráficas, definición y manipulación de modelos de software, aplicaciones web, etc.

2.2 Entorno Web

El desarrollo de aplicaciones web posee determinadas características que lo hacen diferente del desarrollo de aplicaciones o software tradicional.

La ingeniería de la web es multidisciplinar y aglutina contribuciones de diferentes áreas: arquitectura de la información, ingeniería de hipermedia e hipertexto, ingeniería de requisitos, diseño de interfaz de usuario, usabilidad, diseño gráfico y de presentación, diseño y análisis de sistemas, ingeniería de software, ingeniería de datos, indexado y recuperación de información, testeo, modelado y simulación, despliegue de aplicaciones, operación de sistemas y gestión de proyectos.

La ingeniería de la web no es un clon o subconjunto de la ingeniería de software aunque ambas incluyan desarrollo de software y programación, pues, a pesar de que la ingeniería de la web utiliza principios de ingeniería de software, incluye nuevos enfoques, metodologías, herramientas, técnicas, guías y patrones para cubrir los requisitos únicos de las aplicaciones web.

Las alternativas para seleccionar una tecnología para el desarrollo de aplicaciones web son variadas; soluciones simples tales como los lenguajes Perl, PHP, etc. pueden solucionar el problema satisfactoriamente pero implican un costo adicional, y en el transcurrir del tiempo se verán afectadas (como ejemplo Perl tiene la lógica de presentación y la lógica de negocio en un solo archivo).

Por lo expuesto, surgen diferentes tecnologías que intentan solucionar y hacer este tipo de aplicaciones más flexibles, dotándolas de características distribuidas que claramente definen una separación de los elementos.

Una solución puede ser la que presentamos en el desarrollo y la fuente de este trabajo, Java, que es un lenguaje de programación concurrente, orientado a objetos, con capacidades cliente-servidor. En Java se puede abrir una conexión a una página web u otra aplicación de Internet, y leer o escribir datos, en gran medida como un programador de C o C++ lee o escribe en el terminal local.

En lo relativo a servidores de aplicaciones, el panorama actual está claramente centrado en el uso de servidores de aplicaciones en torno a Java EE.

El estado del arte del lenguaje Java, para aplicaciones tanto del lado cliente como del servidor, ha alcanzado un nivel de madurez que ya nadie pone en duda, y es lo que ha llevado a muchas organizaciones a adoptar la plataforma J2EE como base de su estrategia de actualización tecnológica para entornos distribuidos, adopción del e-business, etc.

Complementando Java surgen muchas aplicaciones tales como los servlets que se ejecutan en el servidor y que no presentan ningún tipo de interfaz gráfica, puesto que están totalmente controlados por un servicio de red como pudiera ser un servidor web. Hasta ahora CGI era el único medio de proporcionar interacción entre el cliente y el servidor. Un ejemplo muy común de uso de CGI son los típicos formularios que el usuario ha de rellenar con sus datos personales, que posteriormente pasan a formar parte de una base de datos.

Con la aparición de J2EE, surgió todo un nuevo catálogo de patrones de diseño. Desde que J2EE es una arquitectura por sí misma que involucra otras arquitecturas, incluyendo servlets, *JavaServer Pages* (JSPs), *Enterprise JavaBeans* y más, merece su propio conjunto de patrones específicos para diferentes aplicaciones empresariales.

De acuerdo con el libro *Core J2EE patterns Best Practices and Design Strategies* (18), existen cinco capas en la arquitectura J2EE: cliente, presentación, negocios, integración y recurso. El libro expone 15 patrones J2EE que están divididos en tres de las anteriores capas: presentación, negocios e integración; en lo que no profundizaremos ahora mismo.

En nuestro caso, emplearemos el servidor GlassFish, proporcionado por *Sun* en la misma instalación de J2EE, que mediante JDBC se conecta a la base de datos MySQL. Con dicho motor de base de datos estará organizada toda la información del sistema PFCJMAG, tanto en lo referente a usuarios como a toda la gestión de las rutas.

2.2.1 J2EE

Java Platform Enterprise Edition (J2EE) es una plataforma de programación en la que se define el estándar para el desarrollo de aplicaciones empresariales multicapa diseñado por la empresa *Sun Microsystems*.

J2EE simplifica estas aplicaciones empresariales basándolas en componentes de software modulares y estandarizados ejecutándose sobre un servidor de aplicaciones, proveyendo un completo conjunto de servicios a estos componentes y manejando muchas de las funciones de la aplicación de forma automática, sin necesidad de una programación compleja.

Un componente es una unidad de software independiente que forma parte de una aplicación J2EE, con sus clases y ficheros relacionados, comunicándose con otros componentes. En la especificación hay definidos tres tipos:

- **Componentes de cliente**: aplicaciones instaladas en el cliente y los applets.
- **Componentes de red**: *Java Servlets* y *Java Pages* (JSPs).
- **Componentes de negocio**: *Enterprise Java Beans*.

Estos componentes se escriben en lenguaje Java y se compilan de la misma forma que una aplicación cualquiera, pero con la diferencia de que se ensamblan en una aplicación J2EE, se verifica si están bien formados y cumplen la especificación y se despliegan cuando se ejecuta el servidor.

Java EE incluye varias especificaciones de API, tales como JDBC, RMI, e-mail, JMS, servicios web, XML, etc. y define cómo coordinarlos. Java EE también configura algunas especificaciones únicas para componentes Java EE. Estas incluyen *Enterprise JavaBeans*, servlets, portlets (siguiendo la especificación de *Portlets Java*), JSPs y varias tecnologías de servicios web. Esto permite al desarrollador crear una aplicación de empresa portable entre plataformas y escalable, a la vez que integrable con tecnologías anteriores. Otros beneficios añadidos son, por ejemplo, que el servidor de aplicaciones puede manejar transacciones, la seguridad, escalabilidad, concurrencia y gestión de los componentes desplegados; dando como resultado que los desarrolladores puedan centrarse más en la lógica de negocio de los componentes en lugar de en tareas de mantenimiento de bajo nivel.

Las APIs de Java EE incluyen varias tecnologías que extienden la funcionalidad de las APIs base de Java SE. Haremos mención únicamente a dos de estos paquetes a los que da alcance nuestro desarrollo.

- ***javax.naming***. Los paquetes *javax.naming*, *javax.naming.directory*, *javax.naming.event*, *javax.naming.ldap* y *javax.naming.spi* definen la API de JNDI. La Interfaz de Nombrado y Directorio Java es una interfaz de programación de aplicaciones para servicios de directorio. Esto permite a los clientes descubrir y buscar objetos y nombres a través de un apelativo. Adicionalmente, especifica un SPI que permite que las implementaciones del servicio de directorio sean conectadas al *framework*. Las implementaciones pueden hacer uso de un servidor, un fichero, o una base de datos.

- ***java.sql***. Los paquetes *java.sql* y *javax.sql* definen la API de JDBC que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema de operación donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL.

Por último, la versión de Java EE que usamos en este proyecto (versión 5) ha intentado facilitar el desarrollo de sus componentes haciendo uso de la sintaxis del lenguaje, las cuales han sido incluidas en J2SE 5.0. Java 2.1 apoya este objetivo definiendo las anotaciones de inyecciones de dependencias en manejadores de etiquetas JSP y escuchadores de contexto.

Otro punto mejorado de Java EE 5 ha sido la especificación de sus tecnologías web, llamadas *JavaServer Pages* (JSP), *JavaServer Faces* (JSF) y *JavaServer Pages Standard Tag Library* (JSTL).

2.2.2 Servlet – JSP

La web ha tenido una larga trayectoria desde sus comienzos en 1984 hasta la actualidad, la cual comentamos seguidamente a grandes rasgos.

La web fue creada para intercambiar documentos estáticos aunque finalmente entraron en marcha contenidos dinámicos, todo gracias a la *Common Gateway Interface* (CGI). CGI era un estándar que permitía a los servidores web interactuar con aplicaciones externas, por lo cual las páginas ya no tenían por qué ser estáticas. Aunque las aplicaciones CGI fueron muy buenas, contaban con una serie de limitaciones, lo que dio lugar a la creación de los servlets de Java.

Los servlets de Java fueron una revolución ya que constituían una gran mejora frente a la GCI. Los servlets proporcionaban un método independiente de la plataforma basado en componentes para crear aplicaciones web. Además eran más eficaces que los CGI ya que permitían crear un solo proceso pesado y que múltiples peticiones fueran procesadas en la misma instancia de servlet. Su gran problema es que necesitaban una gran interactuación con el navegador, ya que por ejemplo había que mandar una gran cantidad de peticiones de impresión (*println*) al navegador.

Finalmente se comenzaron a desarrollar aplicaciones web basadas en *JavaServer Pages* (JSP), ya que los servlets no tenían incorporado HTML lo cual se comprobó que provocaba un problema de responsabilidad. Las JSP son documentos de textos que combinan etiquetas estáticas HTML y XML, además de *scriptles*.

Tras todos estos sucesos y avances de la web, los desarrolladores se dieron cuenta de que debían combinar las JSP y los servlets para desplegar aplicaciones web. De esta forma aparecieron dos modelos:

- **Modelo 1**. Las páginas JSP gestionaban todo el procesamiento de una petición y eran responsables de mostrar el resultado al cliente.
- **Modelo 2 (MVC)**. La petición del cliente la interceptaba un servlet controlador que gestionaba el procesamiento inicial de la petición y determinaba que página JSP mostrar a continuación.

Retomaremos esta cuestión más adelante, tras una descripción más detallada de los elementos que intervienen.

2.2.2.1 Servlet

Un servlet es un componente que forma parte de una aplicación web y se guarda y ejecuta en un servidor J2EE. Su función principal es interaccionar con el cliente recibiendo peticiones y generando respuestas a partir de ellas.

La tecnología servlet de Java provee desarrollos web con un mecanismo simple y consistente para extender la funcionalidad de un servidor web y un acceso a un sistema de datos. Un servlet puede ser casi considerado como una applet que corre sobre un servidor. Por ello, los servlets de Java hacen posibles muchas aplicaciones web actuales.

A diferencia de un JSP, como veremos más adelante, un servlet es un componente escrito puramente en Java, en términos Java esto equivale a una clase y como cualquier otra clase Java ésta se encuentra compuesta por diversos métodos o funciones.

La tecnología servlet proporciona las mismas ventajas del lenguaje Java en cuanto a portabilidad ("*write once, run anywhere*") y seguridad, ya que un servlet es una clase de Java igual que cualquier otra y, por tanto, tiene en tal aspecto todas las características del lenguaje.

Además, los servlets se benefician de la gran capacidad de Java para ejecutar métodos en ordenadores remotos, para conectar con bases de datos, para la seguridad en la información, etc. Se podría decir que las clases estándar de Java entregan ya resueltos muchos problemas que con otros lenguajes tiene que resolver el programador.

Las características de los servlets son:

- Son independientes del servidor utilizado y de su sistema operativo, lo que quiere decir que, al estar escritos en Java, el servidor puede estar escrito en cualquier lenguaje de programación, obteniéndose exactamente el mismo resultado que si lo estuviera en Java.

- Los servlets pueden llamar a otros servlets e incluso a métodos concretos de otros servlets. De esta forma se puede distribuir de forma más eficiente el trabajo a realizar. Por ejemplo, se podría tener un servlet encargado de la interacción con los clientes y que llamara a otro servlet para que a su vez se encargara de la comunicación con una base de datos. De igual forma, los servlets permiten redireccionar peticiones de servicios a otros servlets (en la misma máquina o en una máquina remota).
- Los servlets pueden obtener fácilmente información acerca del cliente (la permitida por el protocolo HTTP), tal como su dirección IP, el puerto que se utiliza en la llamada, el método utilizado (GET o POST), etc.
- Permiten además la utilización de cookies y sesiones, de forma que se puede guardar información específica acerca de un usuario determinado, personalizando de esta forma la interacción cliente-servidor. Una clara aplicación es mantener la sesión con un cliente.
- Los servlets pueden actuar como enlace entre el cliente y una o varias bases de datos en arquitecturas cliente-servidor de tres capas (si la base de datos está en un servidor distinto).
- Asimismo, pueden realizar tareas de proxy para un applet. Debido a las restricciones de seguridad, un applet no puede acceder directamente por ejemplo a un servidor de datos localizado en cualquier máquina remota, pero el servlet sí puede hacerlo en su lugar.
- Al igual que los programas CGI, los servlets permiten la generación dinámica de código HTML dentro de una propia página HTML. Así, pueden emplearse servlets para la creación de contadores, banners, etc.

Los tres métodos que debe implementar un servlet son:

- *service* (obligatorio). Este método es la parte medular de todo servlet ya que dentro de él se incluyen las tareas principales de ejecución.
- *init* (opcional). Es un método ejecutado antes del método *service*, su labor principal es adquirir o inicializar algún recurso que será empleado por *service*, estos recursos típicamente son conexiones hacia BBDD.
- *destroy* (opcional). Ejecutado una vez que ha terminado el método *service*, su labor es liberar los recursos utilizados o adquiridos en el proceso de ejecución, los cuales generalmente son aquellos reservados por *init*.

2.2.2.2 JSP

Un JSP (*Java Server Page*) es uno de los componentes más básicos empleados para aplicaciones de servidor en Java. Su composición consta de dos grandes partes: HTML y lenguaje Java.

Mediante HTML se especifica el contenido estático de despliegue y es mediante fragmentos del lenguaje Java que se genera contenido dinámico, en efecto, cumpliendo la definición de aplicación de servidor.

La tecnología de las páginas JSP ayuda a crear contenido web dinámico de una forma rápida y sencilla como respuesta a una petición de un cliente web. JSP permite un desarrollo rápido de aplicaciones basadas en web independientemente de servidores y plataformas. JSP se encuentra en la plataforma Java EE 5.

Las páginas JSP se dividen en dos subcategorías, que son usadas para proveer una plataforma independiente que extiende las capacidades de un servidor web:

- *Tag Libraries*.
- *JavaServer Pages Standard Tag Library*.

Las principales características que se añadieron a esta definición fueron:

- Soporte para la comprobación de expresiones regulares, que son evaluadas por un gestor de etiquetas cuando es necesario (compilación del archivo), mientras que son evaluadas dinámicamente cada vez que una página es ejecutada. En la parte de descripción del sistema a bajo nivel retomaremos esta cuestión para los JSPs implementados (sección *4.2.1 Casos excepciones en el diseño*).
- Soporte para expresiones *Ivalue*, que aparecen en la parte izquierda de la operación de asignación. Cuando se usa un *Ivalue*, una Expresión de Lenguaje (EL) representa la referencia a una estructura de datos, como por ejemplo podría ser una propiedad de un JavaBean, que ha sido asignado a alguna salida del usuario.

La ejecución y transformación de elementos Java es trabajo de un *Servlet Engine*. La secuencia de ejecución para un JSP es la siguiente (partiendo de la requisición de página):

1. Compilar JSP a servlet.
2. Cargar/Compilar el JSP (servlet).
3. Inicializar el JSP (servlet).
4. Ejecutar el JSP (servlet).
5. Limpiar el JSP (servlet).

El primer paso puede ser sorprendente, pero efectivamente todo JSP es convertido a un servlet; la razón de esto es que el surgimiento de JSP se debió a la necesidad de facilitar un formato programático sencillo para personal no familiarizado con Java, esto es, el uso de JSPs permite eliminar visualmente funciones de arranque, ejecución y finalización, la inclusión de éstas se lleva a cabo en la conversión de JSP a servlet.

Una vez convertido el JSP a servlet, este servlet (JSP) debe ser compilado para continuar con el proceso, esta compilación también es tarea del *Servlet Engine*. Compilado el servlet (JSP) se inicializa, ejecuta y limpia, estos tres pasos son llevados a cabo por las funciones: *_jspx_init()*, *_jspService()* y *_jspx_Destroy()*, respectivamente. Estas mismas funciones son parte central de cualquier servlet, como ya se ha visto.

Debido a que un JSP ofrece una abstracción por encima de un servlet, y éste último es un objeto (clase) Java, también se tiene acceso a diversos objetos dentro de un JSP sin la necesidad de realizar declaraciones complejas. Pasamos a citarlos brevemente, en forma de tabla.

Objeto	Uso	Clase Base
request	Representa el objeto de solicitud dentro de un JSP/servlet.	HttpServletRequest
response	Representa el objeto de respuesta dentro de un JSP/servlet.	HttpServletResponse
pageContext	Representa el contexto del JSP/servlet.	PageContext
session	Representa la sesión del usuario en un JSP/servlet.	HttpSession
application	Representa el objeto de aplicación (contexto) para un JSP/servlet.	ServletContext
out	Representa el objeto de escritura (para enviar a pantalla) en un JSP/servlet.	JspWriter
config	Representa el objeto de configuración para un JSP/servlet.	ServletConfig
page	Representa el objeto del JSP/servlet en sí.	Object
exception	Representa el objeto de errores para un JSP/servlet.	Throwable

Tabla 1 – Objetos declarados implícitamente en un JSP (19)

En definitiva, la principal ventaja de JSP frente a otros lenguajes es que permite integrarse con clases Java, lo que permite separar en niveles o capas las aplicaciones web, almacenando en clases Java las partes que consumen más recursos (así como las que requieren más seguridad) y dejando la parte encargada de formatear el documento HTML en el archivo JSP. La idea fundamental detrás de este criterio es el de separar la lógica del negocio de la presentación de la información.

2.2.3 JavaBeans[4]

Un JavaBean es una manera de modularizar el uso de datos en una aplicación con JSPs o servlets a través de una clase. Su característica primordial es el uso de los métodos *get* y *set*, los cuales permiten el acceso a sus valores. El diseño de un JavaBean es relativamente sencillo, ya que no posee código extenso.

Los detalles de un JavaBean común pueden ser los siguientes:

- La clase implementa la interface *java.io.Serializable*, una característica primordial de todo JavaBean.
- Posteriormente se definen los campos (*fields*) utilizados dentro del JavaBean en un contexto privado (*private*).
- Se definen dos constructores Java, uno que asigna los parámetros a la instancia con valores de entrada y el constructor sin datos de entrada.

[4] Un JavaBean no es lo mismo que un *Enterprise JavaBean*. Como vemos en este punto, el diseño mediante el uso de JavaBeans permite encapsular diversos valores en una clase Java y que su contenido sea manipulable fácilmente en JSPs o servlets. Lo anterior es muy diferente al uso de *Enterprise JavaBeans*, los cuales son complementarios a JSPs o servlets; el uso de *Enterprise JavaBeans* se da en ambientes que requieren controles transaccionales y de seguridad más avanzados que lo que ofrecen JSPs o servlets.

- A través de los diversos métodos *get/set* es posible modificar los valores iniciales definidos en el JavaBean.

A través de JavaBeans es posible encapsular conceptos de diferentes tipos en el proceso, permitiendo la manipulación de valores a través de una sola clase, de esta manera facilitando el manejo de información y, a su vez, el desarrollo de software. A simple vista, el uso de JavaBeans parece superficial, pero conforme empieza a crecer el tamaño de una aplicación su uso se hace más evidente.

2.2.4 MVC

La estructura MVC (*Model-View-Controller*) es un paradigma utilizado en diversos desarrollos de software. A través de este *framework* se logra una división de las diferentes partes que conforman una aplicación, siendo su principal razón de ser la manutención del código fuente.

Conforme incrementan las necesidades de cualquier aplicación, la modificación del código existente se hace inminente y, si no existe una clara división de uso, el código no sólo se torna indescifrable sino, en ocasiones, impredecible debido a la mezcla de funcionalidades que pueden surgir.

A través de MVC se realiza la siguiente división:

- **Modelo**. Es el responsable de la lógica de negocio: concentra las funcionalidades relacionadas con el modelo de datos, esto es, el acceso y manipulación de depósitos informativos como BBDD y archivos.
- **Vista**. Es la responsable de presentar el contenido: se basa en el aspecto visual o gráfico que será empleado por la aplicación en cuestión.
- **Controlador**. Empleado como un mediador entre el medio gráfico, Vista, y el Modelo, coordina las acciones que son llevadas a cabo entre ambos, es decir, se encarga de controlar el flujo y estado de la entrada de datos por parte del cliente.

La evolución de los desarrollos de aplicaciones web tiene una trayectoria popular por su rapidez y complejidad, lo cual podemos resumir en los siguientes puntos:

- **Modelo 1**. Son las aplicaciones web más primitivas. Se identifican con este modelo las clásicas aplicaciones web CGI, basadas en la ejecución de procesos externos al servidor web, cuya salida por pantalla era el HTML que el navegador recibía en respuesta a su petición. Presentación, negocio y acceso a datos se confundían en un mismo script Perl.
- **Modelo 1.5**. Aplicado a la tecnología Java, se da con la aparición de las páginas ASP de Microsoft, y posteriormente con los JSPs y servlets. En este modelo, las responsabilidades de presentación (navegabilidad, visualización, etc) recaen en las páginas dinámicas generadas en el servidor, mientras que los componentes incrustados en las mismas (JavaBeans, ActiveX, etc.) son los responsables del modelo de negocio y acceso a datos.
- **Modelo 2**. Como evolución del modelo 1.5, con la incorporación del patrón MVC a este tipo de aplicaciones, se define lo que se conoce como Modelo 2 de la

arquitectura web. Se aprecia la incorporación de un elemento controlador de la navegación de la aplicación.

- **Modelo-Vista-Controlador (MVC)**, es el patrón de diseño que ha llegado a dominar el espacio de programación de aplicaciones web. Ahora es muy común usar los términos Modelo 2 y MVC indistintamente.

El uso de MVC para JSPs y servlets en ambientes web ha empezado a generar gran interés, debido a que una vez diseñada una aplicación para ambiente web es raro que ésta permanezca sin cambios. El uso de MVC permite realizar diseños con JSPs o servlets que logran verdaderas soluciones a escala.

Tomemos el simple caso de un proceso de registro que después de varios meses requiere ser modificado para solicitar datos no contemplados en el diseño inicial, o bien, la modificación de un proceso de trabajo (*WorkFlow*) que requiere modificar el orden de solicitud de datos; cualquiera de los casos anteriores requiere forzosamente modificación del código de JSPs o servlets.

El MVC funciona de la siguiente manera: el flujo de la aplicación está dirigido por un controlador central. El controlador delega solicitudes (en nuestro caso, solicitudes HTTP) a un manejador apropiado. Los manejadores están unidos a un modelo, y cada manejador actúa como un adaptador entre la solicitud y el modelo. El modelo representa o encapsula un estado o lógica de negocio de la aplicación. Luego el control es devuelto a través del controlador hacia la vista apropiada. El reenvío puede determinarse consultando los conjuntos de mapeos, cargados desde un fichero de configuración en XML. Esto proporciona un acoplamiento cercano entre la vista y el modelo, que hace que las aplicaciones web sean más fáciles de crear y de mantener.

Observese la siguiente ilustración que demuestra los posibles resultados al respecto de una aplicación con JSPs y servlets:

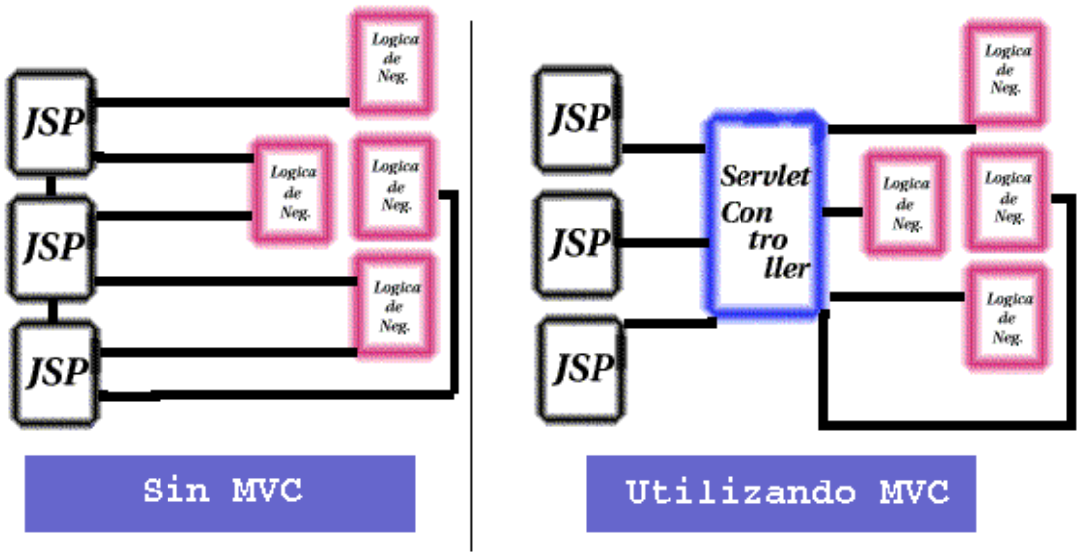


Figura 4 – Adaptación a MVC de una aplicación web (19)

La sección izquierda demuestra un posible diseño que pudo ser empleado al diseñar la aplicación antes mencionada, las principales deficiencias son las siguientes:

- La secuencia de JSPs que conforma la parte visual de la aplicación está enlazada directamente entre sí, esto es, el *WorkFlow* (Flujo de Trabajo) esta implementado en cada JSP y cualquier cambio en esta secuencia implica modificar el código fuente de diversos JSPs.
- La lógica de negocios, que correspondería al Modelo de la aplicación, también está enlazada directamente con los JSPs, esto tiene como consecuencia que el diseño de la lógica de negocios (a través de servlets) está fuertemente basado en el diseño de los JSPs frontales, esto dificulta que sean reutilizados diversos elementos de lógica de negocios (servlets) en distintos JSPs.

Utilizando MVC se obtienen los siguientes resultados a los problemas anteriores:

- La secuencia de JSPs se enlaza a un Controlador en forma de servlet, este enlace permite alterar el flujo de trabajo a través de una modificación sencilla del servlet controlador; los JSPs no contienen código para el flujo de trabajo.
- La lógica de negocios que correspondería al Modelo es obtenida a través del servlet controlador, esto permite que la lógica de negocios (servlets) permanezca aislada de cualquier tipo de despliegue gráfico (JSPs); en efecto, facilitando la reutilización de componentes de negocios (servlets) entre diversos JSPs.

Finalmente, mencionar que hoy en día existen diversas implementaciones para utilizar un *framework MVC* en ambientes de JSPs/servlets. Los *frameworks MVC* han pasado a ser una parte importante de cualquier proyecto web. Actualmente, la evolución de la propia web y de los estándares ha sido el caldo de cultivo de la aparición de un gran número de *frameworks* para facilitar y estandarizar la programación web. La mayoría de ellos son de código abierto, lo que permite su utilización en todos los proyectos independientemente de la infraestructura de despliegue.

El más conocido de todos los *frameworks MVC* es Struts. La historia de Struts es larga y su presencia continua durante muchos años hace que sea el *framework* de referencia.

2.2.5 GlassFish

GlassFish es un servidor de aplicaciones desarrollado por *Sun Microsystems* que implementa las tecnologías definidas en la plataforma Java EE y permite ejecutar aplicaciones que siguen esta especificación. Es gratuito y de código libre, se distribuye bajo una licencia dual a través de la CDDL y la GNU GPL. La versión comercial se denomina *Sun GlassFish Enterprise Server*.

GlassFish está basado en el código fuente donado por *Sun* y *Oracle Corporation*. GlassFish tiene como base al servidor *Sun Java System Application Server* de *Sun Microsystems*, un derivado de *Apache Tomcat* y que usa un componente adicional para escalabilidad y velocidad.

La versión elegida para la implementación ha sido la v2.1.1, incluida en la instalación de *Java EE 5 SDK Update 8*, con capacidades de cluster y nuevas características de interconexión entre servicios web (ver *1.5 Medios empleados*).

2.2.6 WAR

¿Cuáles y cuántos JSPs y servlets pertenecen a una aplicación? La respuesta es: todos aquellos que se encuentren dentro del mismo WAR (*Web-ARchive*). Un WAR es la manera de agrupar y distribuir tanto JSPs como servlets.

Hasta este punto, se han mencionado los detalles respecto a servlets y JSPs, pero aún falta mencionar la forma en que ambos pasan de ser un programa independiente a interactuar unos con otros. Para facilitar la distribución e interacción entre JSPs o servlets estos son agrupados en una estructura denominada WAR descrita a continuación:

- **/*.html, *.jsp y *.css**. Este directorio base contiene los elementos típicamente empleados para un sitio web: documentos HTML, CSS (*Cascading Style Sheets*), JavaScript e imágenes; además, en este directorio residen los JSPs a utilizarse en el WAR. Aquí no residen los servlets, estos deben ser colocados en otra parte del WAR ya que son clases Java puras.
- **/WEB-INF/web.xml**. Este archivo contiene elementos de configuración del WAR como: página de inicio, ubicación o mapeo de servlets, parámetros para componentes adicionales tales como Struts y otros elementos como manejo de errores.
- **/WEB-INF/classes/**. Este directorio contiene las clases Java utilizadas dentro del WAR, es dentro de este directorio donde generalmente residen los servlets diseñados para el WAR.
- **/WEB-INF/lib/**. Este directorio contiene los archivos JAR que serán utilizados por la aplicación, estos generalmente corresponden a las clases utilizadas para conectarse a BBDD o a aquellas utilizadas como librerías.

Este tipo de estructura logra interoperabilidad para las diversas aplicaciones Java, ya que la mayoría de servidores de aplicaciones emplean esta misma estructura para ejecutar componentes.

Finalmente cabe mencionar que la creación de WARs no es obligatoria, sino sólo una manera de transferir y modularizar aplicaciones web.

2.2.7 Conjunto de Conexiones (*Connection Pool*)

El realizar conexiones de cualquier tipo en un sistema de cómputo es un proceso que implica el uso de diversos recursos, por esta razón es recomendable reutilizar cualquier tipo de conexión una vez establecida. La reutilización de estas conexiones establecidas o latentes se lleva acabo colocándolas en un grupo (*pool*) para que cualquier programa o recurso del sistema pueda adquirirla, sin incurrir en las penalidades de generar la conexión desde una etapa inicial.

El mantener conexiones hacia BBDD en una aplicación Java generalmente es de suma importancia, ya que se realizan búsquedas y actualizaciones constantemente; ante este constante uso de conexiones, se opta por emplear algún tipo de *Pool* hacia BBDD.

Hoy en día, existen un gran número de *Pools* disponibles para ambientes JSPs/servlets, algunas de estas implementaciones se encuentran disponibles dentro del mismo driver para la BD, otras se encuentran disponibles dentro de las librerías del *Servlet Engine* o del servidor de aplicaciones, y otras como un producto distribuido por un tercero ajeno al driver, *Servlet Engine* o servidor de aplicaciones.

Generalmente los *Pools* distribuidos por terceros incluyen una serie de métodos adicionales en su estructura interna, los cuales permiten generar conexiones al arranque, asignar un número máximo de conexiones al *Pool* y otra serie de funcionalidades; sin embargo, para utilizar este tipo de *Pools* es necesario conocer los diversos métodos propietarios, lo cual implica documentación especifica del producto.

Ante este tipo de variaciones en los diversos *Pools* disponibles en el mercado, se optará por utilizar una implementación lo más flexible y avanzada posible, y un *Pool* que cumple con estas características es aquel incluido en el propio controlador JDBC de MySQL, el cual permite la reutilización de recursos y puede ser empleado en conjunción de diversas BBDD.

Al emplear el *Pool* mencionado se podrá tener acceso a diversas funciones avanzadas y mecanismos que permiten mayor escalabilidad, además de que su presencia en el software correspondiente garantiza que éste sea mejorado constantemente.

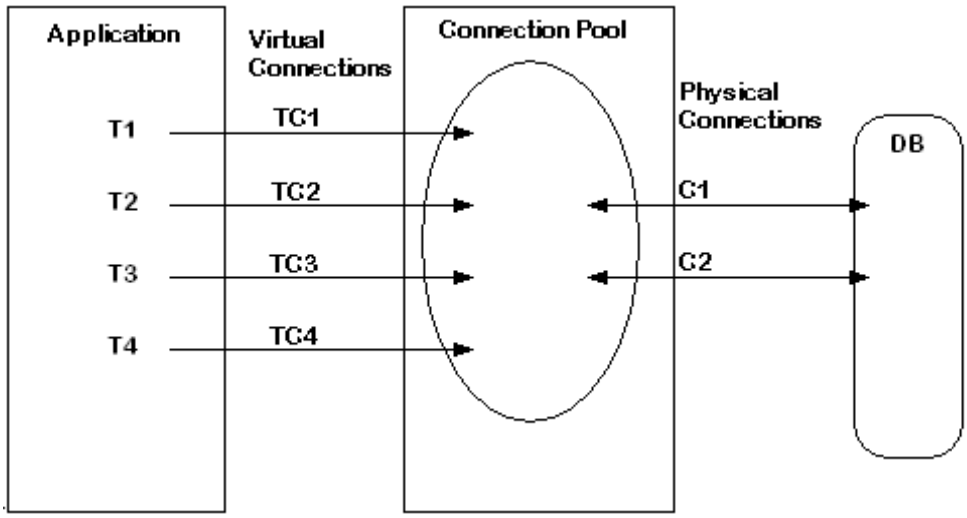


Figura 5 – Diagrama de funcionamiento de un Conjunto de Conexiones

2.2.8 HTML y JavaScript

2.2.8.1 HTML

HTML es el lenguaje de marcado predominante para la construcción de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes. HTML se escribe en forma de etiquetas, rodeadas por corchetes angulares (<,>). HTML también puede describir, hasta un cierto punto, la apariencia de un documento y puede incluir un script (por ejemplo

JavaScript), el cual puede afectar el comportamiento de navegadores web y otros procesadores de HTML.

El diseño en HTML, aparte de cumplir con las especificaciones propias del lenguaje, debe respetar unos criterios de accesibilidad web, siguiendo unas pautas, o las normativas y leyes vigentes en los países donde se regule dicho concepto. Se encuentra disponible y desarrollado por el W3C a través de las Pautas de Accesibilidad al Contenido Web 1.0 WCAG (20).

2.2.8.2 JavaScript

JavaScript es un lenguaje de programación interpretado, es decir, que no requiere compilación, utilizado principalmente en páginas web, con una sintaxis semejante a la del lenguaje Java y el lenguaje C.

Al igual que Java, JavaScript es un lenguaje orientado a objetos propiamente dicho, ya que dispone de herencia, si bien ésta se realiza siguiendo el paradigma de programación basada en prototipos, ya que las nuevas clases se generan clonando las clases base (prototipos) y extendiendo su funcionalidad.

JavaScript es un lenguaje que se integra directamente en páginas HTML. Tiene como características principales las siguientes:

- Es interpretado (que no compilado) por el cliente, es decir, directamente del programa fuente se pasa a la ejecución de dicho programa, con lo que, al contrario que los lenguajes compilados, no se genera ni código objeto ni ejecutable para cada máquina en el que se quiera ejecutar dicho programa.
- Está basado en objetos, pero a diferencia de Java, no emplea clases como tal, ni herencia, ni otras técnicas específicas de la POO.
- Su código se integra en las páginas HTML, incluido en las propias páginas.
- No es necesario declarar los tipos de variables que van a utilizarse, ya que JavaScript realiza una conversión automática de tipos.
- Las referencias a objetos se comprueban en tiempo de ejecución. Esto es consecuencia de que JavaScript no es un lenguaje compilado.
- No puede escribir automáticamente al disco duro. Por eso JavaScript se considera un lenguaje seguro para el entorno.

Todos los navegadores modernos interpretan el código JavaScript integrado dentro de las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del DOM (*Document Object Model*).

Tradicionalmente, se venía utilizando en páginas web HTML para realizar tareas y operaciones únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor. JavaScript se ejecuta en el agente de usuario al mismo tiempo que las sentencias van descargándose junto con el código HTML.

2.3 Entorno Móvil

En primer lugar, se presentará brevemente la plataforma estándar de Java, J2SE, puesto que con las descripciones dadas en un inicio del lenguaje Java y posteriormente de J2EE no es preciso matizar nada más. Además, se presentarán a continuación algunos aspectos sobre la plataforma J2ME, clave esencial para el desarrollo de esta parte, contrastados con las otras dos ediciones.

Seguidamente pasaremos a las tecnologías inalámbricas: una muy conocida hoy en día como es bluetooth y otra menos extendida como es NFC. Sobre ambas podremos conocer cómo funcionan, algunas características tales como la banda de frecuencia sobre la que operan o la tasa de bits que alcanzan en las comunicaciones, sus usos o campos de aplicación, etc.

Por último, daremos paso a las SDKs de Nokia que han intervenido en el proyecto.

2.3.1 J2SE

Esta edición de Java es la más utilizada y es la que recoge en cierta manera la iniciativa del lenguaje Java. Contiene un conjunto de ejecución y de APIs para crear aplicaciones no destinadas a ser centralizadas y ni de accesos concurrentes.

Dada estas características, ha sido necesaria para la implementación de la aplicación para el servidor bluetooth.

2.3.2 J2ME

En este apartado vamos a ver la edición de la plataforma Java que *Sun Microsystems* ha diseñado específicamente para dispositivos móviles y embebidos, *Java 2 Micro Edition* (J2ME). Aunque en esta introducción veamos una visión general de J2ME y su entorno, nos centraremos en el estudio de las herramientas y bibliotecas que ofrece J2ME para la programación de dispositivos específicos como teléfonos móviles y PDAs.

Java comenzó su andadura como lenguaje de programación a mediados de la década de los noventa del siglo pasado. Originalmente fue concebido como un lenguaje de programación que permitía escribir un programa una vez y poder ejecutarlo en multitud de ordenadores, con diferentes plataformas sin tener que compilarlo de nuevo. Esa es una gran ventaja y una característica muy deseable en el entorno de los pequeños dispositivos, por lo que se ha exportado esa filosofía a estos aparatos. Así, mediante J2ME se podrán escribir aplicaciones para una gran variedad de dispositivos diferentes.

Por supuesto, esta nueva edición de Java no es la misma que se utiliza para desarrollar aplicaciones distribuidas en Internet, por ejemplo, sino que es una versión reducida que se adapta claramente a las características físicas de los pequeños dispositivos.

La primera versión salió allá por 1999, la cual sólo contenía una única máquina virtual y un único API, hecho que puso de manifiesto la insuficiencia de esta solución para la gran variedad de dispositivos diferentes. De esta forma, en el año 2000, nació la primera versión de una configuración, es decir, el *Connected Limited Device Configuration* (CLDC 1.0). Una configuración ofrece el API básico para programar dispositivos, aunque no aporta todas las clases necesarias para desarrollar una aplicación completa. Por tanto, la primera configuración no tenía las herramientas necesarias para permitir a los desarrolladores escribir programas para estos dispositivos. En julio de 2000 nació la primera implementación de un perfil, concretamente el llamado *Mobile Information Device Profile* (MIDP 1.0), aunque no estaba destinado a PDAs sino a teléfonos móviles y a paginadores.

Más tarde saldría la versión 2.0 de MIDP, haciendo que J2ME se consolidase más aún dentro de la comunidad de desarrolladores de dispositivos móviles y expandiéndose a una gran velocidad hasta nuestros días.

2.3.2.1 Diferencias con J2EE y J2SE

Algunas diferencias que ofrece J2ME con respecto a J2EE y J2SE, directamente derivadas de las condiciones en las que se va a hacer uso de esta edición, son las siguientes:

- **Tipos de datos**. J2ME no incluye los tipos *float* y *double*, ya que la mayoría de los dispositivos CLDC no tienen unidad de coma flotante debido fundamentalmente a que es una operación muy costosa.
- **Preverificación**. La verificación del código en J2ME se hace fuera del dispositivo, con objeto de reducir la carga de la máquina.
- Inclusión de los **ficheros "descriptor" y "manifiesto"** al empaquetar ficheros J2ME, conteniendo información sobre las aplicaciones que incluyen.
- **Nueva biblioteca gráfica** adaptada a los dispositivos con memorias de poco tamaño y pantallas también pequeñas.
- No existe un método *main* como entrada para la ejecución de la función. Éste se sustituye por el método ***startApp***.
- La **recolección de basura** se hace de manera **manual** y no automática como en el J2EE. Esta decisión se toma así para reducir la utilización de la memoria.

2.3.2.2 Arquitectura

Como ya se ha visto en la introducción, dos son los pilares básicos en J2ME, la configuración y el perfil.

2.3.2.2.1. Configuraciones

J2ME presenta dos configuraciones: CLDC y CDC. La primera se dedica a dispositivos con estrictas limitaciones de memoria, capacidad de cálculo, consumo y conectividad de red. Por otro lado, CDC se encarga de dispositivos con más potencia.

Parte de CLDC es un subconjunto de CDC, por lo que la portabilidad de aplicaciones se puede conseguir cuando nos movemos de un entorno más restringido a otro más amplio o rico. De la misma manera, y siguiendo en el hilo de la portabilidad, una

aplicación en J2ME podrá ejecutarse en J2SE normalmente, salvo que se utilicen las bibliotecas específicas de J2ME.

❖ **CLDC**

Veamos más detalladamente algunas características de CLDC, ya que es la configuración que se ha usado en este proyecto. Comencemos por las propiedades mínimas requeridas a un dispositivo para poder desarrollar con esta configuración:

- De 160 a 512 KB de memoria disponible para el entorno de Java.
- Un procesador de 16 o 32 bits.
- Consumo de energía bajo.
- Permiten algún tipo de conectividad a una red.

Al tener como objetivo dispositivos con prestaciones reducidas, CLDC elimina una gran cantidad de características que sí aparecen en J2EE y J2SE, tanto en el propio lenguaje Java como en la máquina virtual, como por ejemplo:

- Interfaz nativo de Java (*Java Native Interface* - JINI) (Máquina virtual).
- Cargadores de clases definidas por el usuario (Máquina virtual).
- Grupos de hilos e hilos demonios (Máquina virtual).
- Finalización (lenguaje Java).
- Referencias débiles (Máquina virtual).
- Reflexión (Máquina virtual).
- Tipos de datos de punto flotante (lenguaje Java).
- Algunos aspectos de seguridad y APIs (Máquina virtual).
- Verificación de ficheros de clases (Máquina virtual).
- Posee algunas limitaciones en las gestiones de errores (lenguaje Java).

Pero ¿qué razones se consideraron para eliminar esas prestaciones?

Por supuesto, una de ellas se basa en cuestiones de ahorro de memoria, ya que el tamaño general del API queda reducido. Aunque otras también han sido quitadas por cuestiones de aligerar el procesador, como es el caso de las operaciones en punto flotante o la verificación de las clases. Concretamente, esta operación, que identifica y rechaza ficheros de clases inválidas, se realizaba en la máquina virtual en la edición J2SE, pero en ese caso aumenta su tamaño. Por esta razón, en J2ME la verificación se hace en dos partes: la primera, la preverificación, en un ordenador distinto; la segunda sí se lleva a cabo en el propio dispositivo, pero en este caso es mucho más simple y rápida. Más adelante estudiaremos de manera más detallada el proceso de verificación.

Con respecto a la seguridad, al no definir CLDC completamente el sistema de seguridad de Java, deben eliminarse prestaciones que sí figuran en el J2SE y que harían muy vulnerables las aplicaciones. Por ejemplo, sin este modelo de seguridad, un cargador de clases definido por el usuario podría alterar la forma en que el camino de las clases fuera recorrido, pudiendo una aplicación sustituir trozos de las bibliotecas del núcleo de Java y ganar acceso al dispositivo de tal forma que pudiera dañarlo.

También se considera la simple conveniencia como criterio: algunas clases pueden ser desarrolladas por los programadores basadas en otras que sí se mantienen. Igual

ocurre con algunos métodos de clases. Por otro lado, otras clases se eliminan por que no tienen razón de ser.

En cuanto a la máquina virtual de CLDC, KVM, requiere entre 40 y 80 Kbytes dependiendo de las opciones de compilación y el tipo de dispositivo para el que se compile. Esto implica que se podrán ejecutar aplicaciones con un total de 128 Kbytes. Aparte de esto, se necesitan otros 32 Kbytes para memoria dinámica de la aplicación a ejecutar. KVM está implementada en C y está diseñada para ser tan completa y rápida como sea posible. De hecho, puede ejecutarse de un 30% a un 80% de la velocidad de la JVM.

Volviendo a la verificación de clases, la máquina virtual de Java estándar efectúa un proceso en tiempo de ejecución que se denomina verificación de clases, el cual se lleva a cabo antes de cargar ninguna clase en memoria. El objetivo es asegurar la integridad de los ficheros donde se almacena una clase Java y que el código en ella no intente acceder a memoria fuera de su espacio de nombres, eliminando la posibilidad de que pueda sustituir alguno de los paquetes del núcleo de Java lo que pondría en juego la seguridad del sistema. Esta etapa juega un papel muy importante en el modelo de seguridad de Java. Para que nos hagamos una idea, J2SE verifica, entre otros, estos puntos:

- Inicialización de todas las variables locales antes de su uso.
- El constructor de un objeto debe ser llamado justo seguidamente de la creación del mismo, y antes de que se use.
- Cada constructor tiene que comenzar con una llamada al constructor de su superclase.
- Las variables locales y miembros estáticos deben contener referencias a objetos que sean asignables legalmente.

Si nos trasladamos a CLDC, este proceso será muy costoso en términos de uso de recursos, ya que requiere mucha memoria, procesador y espacio para código binario. Es por esto por lo que los diseñadores de KVM decidieron hacer la verificación de clases de manera diferente a como se hace con JVM. Así, antes de que la clase se llegue a emplear en el dispositivo, ésta es modificada externamente por una utilidad "preverificadora". La idea es añadir al fichero clase generado por *javac* (compilador) nuevo código que identifique la clase como válida (pasa a ser una clase verificada). Seguidamente, se transfiere al dispositivo y la KVM sólo tiene que comprobar si esta información está o no presente, y si contiene o no la información correcta. En cualquiera de los dos casos negativos, el proceso de carga se interrumpe y se lanza una excepción. Esta comprobación se puede hacer justo cuando se carga la clase o como parte del proceso de instalación de la aplicación. En cualquier caso es un proceso más rápido que la preverificación y requiere menos memoria.

2.3.2.2.2. Perfiles

Los perfiles suministran un interfaz de usuario, métodos de entrada y mecanismos de persistencia, así como un entorno de desarrollo completo para un conjunto específico de dispositivos, soportando sus características concretas. Los perfiles son necesarios porque las configuraciones no presentan ninguna de estas prestaciones. Así, J2ME oferta al programador el concepto de perfil, el cual define una plataforma común para un grupo determinado de dispositivos, los cuales comparten características y funciones. Un dispositivo puede cubrir más de un perfil.

Los perfiles se asientan sobre una configuración dada y utilizan los servicios que ofrece, aunque es la parte de la arquitectura J2ME que más cerca se encuentra del aparato físico. Para el caso de CLDC, su único perfil es MIDP, el cual cubre el mercado de dispositivos móviles, que comparten características como memoria limitada, pantalla pequeña, conexión a algún tipo de red inalámbrica y mediante banda ancha limitada y alimentados normalmente por baterías.

- **MIDP – MIDlets**

Las aplicaciones MIDP se denominan MIDlets, las cuales pueden utilizar tanto las facilidades aportadas por MIDP como las APIs que MIDP hereda de CLDC, pero nunca acceden directamente al sistema operativo subyacente, por lo que no serían portables. Un MIDlet consiste en una clase Java, como mínimo, derivada de la clase abstracta *MIDlet*, y que se ejecuta en un entorno de ejecución dentro de la máquina virtual, la cual provee un ciclo de vida bien definido controlado mediante métodos que cada MIDlet debe implementar. Un grupo de MIDlets que están relacionados se suelen agrupar en un MIDlet suite. Todos estos MIDlet se empaquetan, instalan, desinstalan y borran como una única entidad y comparten recursos tanto en tiempo de ejecución como estáticos.

Comenzando con los requerimientos de memoria, MIDP necesita 128 KB de RAM disponible para la implementación correspondiente. A esta cantidad debemos sumarle la que necesita CLDC y como mínimo 32 Kbytes para almacenar la pila de la aplicación, tamaño que obliga al programador a tener bastante cuidado a la hora de diseñar las aplicaciones. Además, los dispositivos MIDP cuentan con 8 KB como mínimo de memoria no volátil que se utiliza como almacenamiento persistente, que no se borra tras apagar el aparato (salvando el problema del cambio de batería).

Sobre las pantallas de los dispositivos, la especificación MIDP indica que ésta requiere 96 píxeles de ancho por 54 de alto y que debe soportar al menos dos colores.

Con respecto a los tipos de entradas del dispositivo, el rango es muy amplio: desde los que tienen un teclado alfanumérico completo, hasta aquellos que permiten escribir en ciertas áreas de la pantalla, pasando por los teclados de los teléfonos móviles. La especificación mínima requiere un teclado que permita marcar los números del 0 al 9, junto con el equivalente a las teclas del cursor y un botón de selección.

MIDP no asume que los dispositivos estén permanentemente conectados a una red, ni siquiera que soporten TCP/IP, pero sí que tengan algún tipo de acceso a una red. En este sentido, la especificación sí establece que soporte HTTP 1.1, bien mediante una pila de protocolos o una pasarela WAP.

También los sistemas operativos de los dispositivos tienen restricciones. Por ejemplo, deben ofrecer un entorno de ejecución protegido donde la máquina virtual pueda correr, o algún tipo de apoyo para el acceso a una red, como puede ser el caso de un API para programar *sockets*, sobre el cual el protocolo HTTP se pueda implementar. Es el sistema operativo el que ofrecerá acceso al teclado y al posible dispositivo puntero, entregando los correspondientes eventos que surjan. Además, será el encargado de abstraer al MIDP la pantalla, ya que será visto por éste como una matriz de píxeles, y de ofrecer un interfaz para el acceso al almacenamiento persistente.

Un aspecto muy importante a tener en cuenta es el de la seguridad. CLDC y MIDP no incluyen ningún tipo de prueba en las llamadas al API de las que incluye J2SE, lo que puede suponer un peligro, porque el MIDlet no está restringiendo en ningún sentido. Es por esto por lo que el usuario debería ser bastante cuidadoso a la hora de instalar nuevas aplicaciones.

Los MIDlets necesitan empaquetarse antes de que sean trasferidos a un dispositivo para su instalación: tanto la subclase MIDlet correspondiente, como las clases que requiera y el resto de ficheros necesarios constituirán un único fichero JAR, incluyendo también el conocido como manifiesto del JAR que contiene información de empaquetado que indica qué se almacena en el fichero. Adicionalmente se emplea otro segundo fichero conocido como *Java Application Descriptor* (JAD). El manifiesto del JAR almacena el dispositivo, el nombre y la versión del MIDlet suite en el JAR correspondiente, así como qué ficheros de clase se corresponden con cada MIDlet, información útil para la instalación. El segundo es un fichero de texto que contiene una lista de atributos junto con su valor correspondiente. Algunos atributos están también contenidos en el manifiesto del JAR; esto es debido a que éste puede ser grande y su transferencia lenta, debido a la baja velocidad del acceso a la red que suelen tener estos dispositivos, y en vez de descargar el JAR completo se descarga el fichero JAD, el cual es mucho más pequeño y rápido de transferir, se muestra por la pantalla del dispositivo y se decide si se instala o no.

2.3.2.3 Paquetes opcionales en J2ME

- ***Information Module Profile*** **(IMP) – JSR 195**. Proporciona un entorno de aplicación para dispositivos embebidos que no tiene grandes capacidades gráficas o con recursos limitados de alguna otra manera: paneles de emergencia, parquímetros, sistemas de alarma domésticos y similares.
- ***Wireless Messaging API*** **(WMA) – JSR 120, JSR 205**. Proporciona acceso a recursos de comunicación sin cable como la mensajería SMS.
- ***Mobile Media API*** **(MMAPI) – JSR 135**. Extiende la funcionalidad de la plataforma J2ME incorporando soporte de audio, vídeo y otros tipos de datos multimedia basados en tiempo a dispositivos de recursos limitados.
- ***Location API*** **– JSR 179**. Permite la localización de dispositivos móviles para dispositivos con recursos limitados. El API se ha diseñado para generar información sobre la localización geográfica actual del terminal para las aplicaciones Java. El API cubre la obtención de la localización geográfica presente y la orientación del terminal y acceder a una base de datos de mapas almacenados en el terminal.
- ***SIP API*** **– JSR 180**. Se utiliza para establecer y gestionar sesiones IP multimedia. El mismo mecanismo se puede utilizar para proporcionar mensajería instantánea, presencia y servicios de juego.
- ***Security and Trust Services API for J2ME*** **– JSR 177** (21). Amplía las características de seguridad para la plataforma J2ME añadiendo APIs de cifrado, servicio de firma digital y gestión de credenciales de usuario.
- ***Mobile 3D Graphics*** **– JSR 184**. Permite generar gráficos tridimensionales a frecuencias de imagen interactivas en dispositivos móviles de recursos restringidos.
- ***J2ME Web Services APIs*** **(WSA) – JSR 172**. Amplía la plataforma de servicios web para incluir J2ME. Estas APIs permiten que los dispositivos J2ME puedan

ser clientes de servicios web mediante un modelo de programación consistente con la plataforma estándar de servicios web.

- ***Bluetooth API* – JSR 82**. Proporciona un estándar para la creación de aplicaciones bluetooth, de forma que las aplicaciones desarrolladas con el paquete opcional puedan ejecutarse utilizando esta tecnología.
- ***J2ME RMI Optional Package* – JSR 66**. Permite a dispositivos de consumo y aplicaciones embebidas interactuar como y con aplicaciones distribuidas.
- ***JDBC API* – JSR 169**. Define un subconjunto del API JDBC 3.0 para procesar datos de repositorios, habitualmente BBDD relacionales, mediante SQL y para manipular datos tabulares como si fueran JavaBeans.
- ***Contactless API* – JSR 257**. Utilizado para comunicaciones por proximidad o basadas en contacto. Permite tanto el descubrimiento de tarjetas, como el intercambio de datos con éstas, ya sean NDEF, RFID o lectores externos.

2.3.3 Bluetooth

Norma que define un estándar global de comunicación inalámbrica de corto alcance que posibilita la transmisión de voz y datos entre dispositivos móviles (como teléfonos y ordenadores portátiles) y dispositivos de escritorio (como los ordenadores de sobremesa), mediante un enlace de radiofrecuencia seguro y globalmente libre.

Bluetooth surge principalmente buscando tres objetivos:

- Facilitar las comunicaciones entre equipos móviles y fijos.
- Eliminar cables y conectores entre éstos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre nuestros equipos personales.

Esta tecnología de comunicación comprende hardware, software y requerimientos de interoperatibilidad. Para el establecimiento de la norma se creó en 1998 un grupo de interés especial (*Special Interest Group*) formado por las empresas *Ericsson*, *IBM*, *Intel*, *Nokia* y *Toshiba*.

2.3.3.1 Orígen e historia

El nombre de bluetooth procede del rey vikingo Harald II apodado Blatan (o Bluetooth, como era conocido por los ingleses), el cual, durante la segunda mitad del siglo X, reinó en Noruega y Dinamarca y recibió tal apodo debido a una rara enfermedad que daba color azul a su dentadura.

Fue uno de los más poderosos reyes de Europa y se caracterizó ser un buen comunicador y por unificar las tribus danesas, noruegas y suecas. De esta misma manera la tecnología bluetooth intenta unificar las comunicaciones entre diferentes dispositivos como ordenadores y móviles.

A pesar de que la primera versión o especificación de bluetooth no fue publicada hasta 1999, ya desde 1994 *Ericsson* trabajaba en el estudio de un interfaz de radiofrecuencia, de bajo coste y bajo consumo, que permitiera la interconexión entre dispositivos móviles y otros accesorios sin la necesidad de cables. Mediante este estudio,

Ericsson fue despertando el interés en otras empresas del sector de las telecomunicaciones y se dieron cuenta que para que esta tecnología tuviera éxito era necesario que se asegurara la interoperabilidad entre dispositivos de diferentes fabricantes. Es así como surge en 1998 el *Special Interest Group* (SIG), formado por cinco promotores que fueron: *Ericsson*, *Nokia*, *IBM*, *Toshiba* e *Intel*. Más tarde se unirían a este grupo otras empresas como *Microsoft*, *Compaq*, *Motorola*, *Dell*, *3com*, *Axis Comunication*, *Xircom*, *Lucent*.

Como muestra del gran éxito que fue alcanzando bluetooth, el grupo SIG llegó a estar formado por unas 1500 empresas adheridas en el 2000 y siete años más tarde, en 2007, ya eran 9000 las empresas asociadas.

2.3.3.2 Principales características y especificaciones

La tecnología bluetooth opera en la banda de frecuencia 2,4 GHz (mas en concreto entre 2,4 y 2,48 Ghz), banda que comparte con otras tecnologías de radiofrecuencia (RF). Esta banda se caracteriza por no necesitar licencia para poder operar libremente en ella. Además, se utiliza la técnica de FHSS[5] (*Frecuency Hoping Spread Spectrum*) con un máximo de 1600 saltos/seg. Los saltos de frecuencia se dan entre un total de 79 frecuencias con intervalos de 1 Mhz; esto permite dar seguridad y robustez.

El rango máximo de transmisión usando bluetooth es de unos 10 metros, pudiéndose ampliar hasta los 100 metros aumentando la potencia de transmisión o utilizando repetidores. Además, se alcanza una tasa de entre 720 kbps y 1 Mbps.

Clase	Potencia máxima permitida (mW)	Potencia máxima permitida (dBm)	Rango (aproximado)
Clase 1	100 mW	20 dBm	~100 metros
Clase 2	2.5 mW	4 dBm	~25 metros
Clase 3	1 mW	0 dBm	~1 metro

Tabla 2 – Rangos de transmisión y potencias de bluetooth (22)

Con respecto a la seguridad, bluetooth permite hacer uso de técnicas de autentificación y encriptación (de 64 bits), para controlar la conexión y evitar que dispositivos ajenos a la comunicación puedan acceder a los datos o realizar su modificación. Así se definen diferentes modos de seguridad en el perfil de acceso: transmisión sin seguridad, con seguridad a nivel de servicio o con seguridad a nivel de enlace (que resultaría la más potente).

Asimismo, bluetooth permite una fácil integración con redes TCP/IP.

[5] Técnica de modulación en espectro ensanchado en el que la señal se emite sobre una serie de radiofrecuencias aparentemente aleatorias, saltando de frecuencia en frecuencia sincrónicamente con el transmisor. Los receptores no autorizados escucharán una señal ininteligible.

2.3.3.3 Arquitectura y funcionamiento

El núcleo del sistema bluetooth consiste en un transmisor de radio, una banda base y una pila de protocolos. El sistema permite la conexión entre dispositivos y el intercambio de distintos tipos de datos entre ellos.

2.3.3.3.1. Radio

❖ **Bandas de frecuencia y organización de canales**

Como ya se ha visto, el sistema bluetooth funciona en la banda ISM de 2,4 GHz. Esta banda de frecuencias abarca 2400 – 2483,5 MHz. Los 79 canales RF se organizan por números, de 0 a 78, con un espacio de 1 MHz entre ellos, empezando por 2402 GHz.

Alcance reglamentario	Canales RF
2,400 – 2,4835 GHz	f = +k MHz, k=0,…,78

Tabla 3 – Canales RF de bluetooth

Para satisfacer la reglamentación relativa a las transmisiones fuera de banda de cada país, se utiliza una banda de guarda en los extremos inferior y superior de la gama de frecuencias.

Banda de guarda inferior	Banda de guarda superior
2 MHz	3,5 MHz

Tabla 4 – Bandas de guarda en bluetooth

❖ **Modos de modulación**

Se definen dos modos de modulación. Un modo obligatorio, llamado modo de transferencia básica, que usa una modulación de frecuencia binaria para reducir al mínimo la complejidad del transmisor y receptor. Y un modo opcional, llamado de transferencia de datos mejorada, que usa modulación PSK y cuenta con dos variantes: π/4-DQPSK y 8DPSK.

Para la transmisión bidireccional se emplea una técnica de dúplex por división de tiempo (TDD) en ambos modos. Esta especificación define los requisitos de una radio bluetooth, tanto para el modo de transferencia básica como de transferencia mejorada de datos.

❖ **Características del transmisor**

Los dispositivos de clase 1 disponen de control de potencia. El control de potencia se usa para limitar la potencia transmitida por encima de +4 dBm. Por debajo de +4 dBm, la capacidad para controlar la potencia es opcional, y puede emplearse para controlar el consumo energético y el nivel global de interferencias.

Los dispositivos con capacidad para controlar la potencia usan comandos LMP para optimizar la potencia de salida en un enlace físico (consultar protocolo de gestión de enlace en el punto *2.3.3.3.3 Pila de Protocolos*).

- **Emisiones espurias**. Las emisiones espurias dentro de la banda se medirán con un transmisor de salto de frecuencia que transmitirá en un canal de radio y recibirá en

otro; esto implica que el sintetizador puede cambiar de canal de radio entre recepción y transmisión, pero siempre vuelve al mismo canal de transmisión. Este documento no hace referencia a las emisiones espurias fuera de la banda ISM, ya que el cumplimiento de la normativa en el país de destino del dispositivo es responsabilidad del fabricante.

- **Tolerancia de la radiofrecuencia**. La precisión de la frecuencia central debe ser de ±75 KHz con respecto a la frecuencia central.
- **Transferencia de datos mejorada (EDR)**. Un aspecto clave del modo de transferencia de datos mejorada es que se cambia la secuencia de modulación dentro del paquete. El código de acceso y la cabecera del paquete se transmiten dentro de la secuencia de modulación GFSK de 1 Mbps de la velocidad base, en tanto que la secuencia de sincronización, la carga útil y la secuencia de cola se transmite usando la secuencia de modulación de transferencia de datos mejorada.
- **Características de modulación EDR**. Durante la transmisión del código de acceso y la cabecera del paquete, se usa la secuencia de modulación GFSK de transferencia básica. Durante la transmisión de la secuencia de sincronización, la carga útil y la secuencia de cola se usa un tipo de modulación PSK con una velocidad de transmisión de 2 Mbps u, opcionalmente, 3 Mbps.

❖ **Caracterísitcas del receptor**

- **Nivel de sensibilidad**. El nivel de sensibilidad real se define como el nivel de entrada para el cual se satisface un porcentaje de error de bit (BER) del 0,1%. Para cualquier transmisor bluetooth, la sensibilidad del receptor será de –70 dBm o inferior.
- **Rendimiento con interferencias**. La interferencia co-canal y las interferencias en los canales adyacentes en 1 y 2 MHz se miden con la señal útil 10 dB sobre el nivel de sensibilidad de referencia En el resto de canales de radiofrecuencia, la señal útil debe estar 3 dB sobre el nivel de sensibilidad de referencia.
- **Bloqueo fuera de banda**. La supresión (o rechazo) fuera de banda se medirá con la señal útil 3 dB por encima del nivel de sensibilidad de referencia. La señal de la interferencia será de onda continua. El BER será ≤ 0,1%.
- **Características de intermodulación**. La sensibilidad de referencia, BER = 0,1%, se satisfará bajo las siguientes condiciones:
  - La señal útil tendrá una frecuencia f0 con un nivel de potencia de 6 dB por encima del nivel de sensibilidad de referencia.
  - Una señal de onda senoidal estática tendrá una frecuencia f1 con un nivel de potencia de –39 dBm.
  - Una señal con modulación bluetooth tendrá una frecuencia f2 con un nivel de potencia de –39 dBm.
- **Transferencia de datos mejorada (EDR) – nivel de sensibilidad EDR real**. El nivel de sensibilidad real se definirá como el nivel de entrada para el cual se satisface un porcentaje de error de bit (BER) del 0,01%. El nivel de sensibilidad real de un receptor de transferencia de datos mejorada bluetooth π/4-DQPSK y 8DPSK será de –70 dBm o superior. El receptor alcanzará el nivel de sensibilidad de –70 dBm con cualquier transmisor bluetooth que cumpla los requisitos de la especificación del transmisor de transferencia de datos mejorada.

2.3.3.3.2. Banda base

La banda base bluetooth es la parte del sistema bluetooth que especifica o introduce los procedimientos de acceso de medios y capa física entre dispositivos bluetooth. Dos o más dispositivos que comparten el mismo canal físico forman una piconet. Un dispositivo bluetooth actúa como maestro de la piconet, y los demás dispositivos actúan como esclavos. Una piconet puede constar de hasta siete dispositivos activos. Además de estos dispositivos activos, la piconet puede contar con muchos más esclavos en estado de espera.

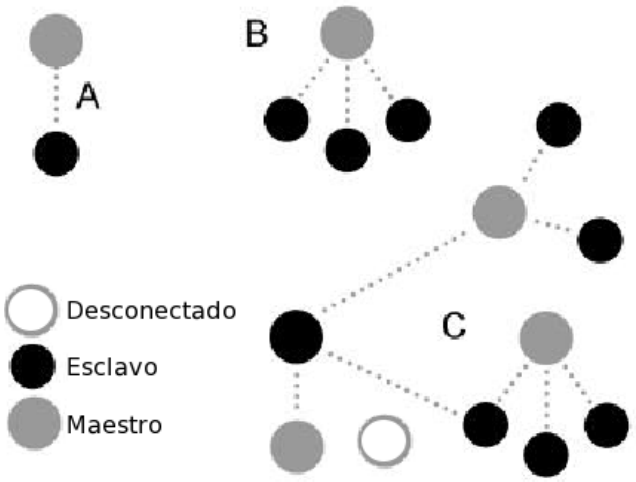


Figura 6 – Piconets con un único esclavo (A), varios esclavos (B) y funcionamiento disperso (C)

❖ **Paquetes**

Los datos se transmiten por radio en paquetes. El paquete general del modo de transferencia básica consta de tres entidades: el código de acceso, la cabecera, y la carga útil. La tasa de transferencia bruta es de 1 Mbps en el modo de transferencia básica.

Figura 7 – Formato estándar de paquetes del modo de transferencia básica

El paquete de transferencia de datos mejorada general consta de seis entidades: el código de acceso, la cabecera, el periodo de guarda, la secuencia de sincronización, la carga útil de la transferencia de datos mejorada, y la cola o tráiler.

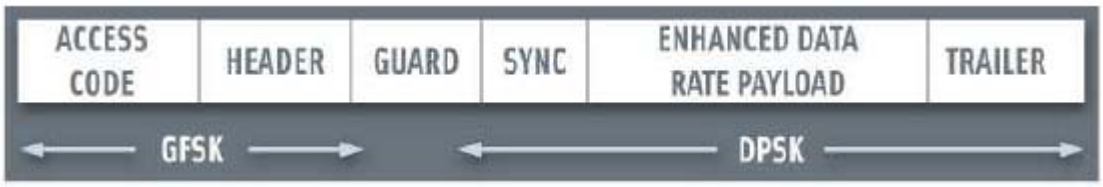


Figura 8 – Formato estándar de paquetes para la transferencia de datos mejorado

El código de acceso y el encabezamiento usan la misma secuencia de modulación que los paquetes del modo de transferencia básico, en tanto que la secuencia de sincronización, la carga útil de la transferencia de datos mejorada y la cola usan la secuencia de modulación de la transferencia de datos mejorada.

La tasa de transferencia de datos mejorada cuenta con una modalidad de modulación primaria que proporciona una velocidad de transmisión bruta de 2 Mbps, y una modalidad de modulación secundaria que proporciona una velocidad de transmisión bruta de 3 Mbps.

❖ **Reloj bluetooth**

Todos los dispositivos bluetooth cuentan con un reloj nativo que debe derivarse de un reloj del sistema de libre funcionamiento. Para la sincronización con otros dispositivos se utilizan compensaciones (*offsets*) que, cuando se suman al reloj nativo, proporcionan relojes bluetooth temporales sincronizados entre sí.

❖ **Dirección de dispositivos bluetooth**

A cada dispositivo bluetooth se le asigna una dirección de dispositivo bluetooth única de 48 bits (BD_ADDR) proporcionada por la autoridad reguladora de IEEE.

❖ **Códigos de acceso**

En el sistema bluetooth, todas las transmisiones que se realizan a través del canal físico se inician con un código de acceso. Se definen tres códigos diferentes:

- Código de acceso del dispositivo (DAC).
- Código de acceso del canal (CAC).
- Código de acceso de consulta (IAC).

❖ **Canales físicos**

Los canales físicos se definen mediante una secuencia de saltos pseudo-aleatorios en los canales RF, la sincronización de los paquetes (ranuras) y un código de acceso. La secuencia de saltos se determina a partir de la dirección del dispositivo bluetooth y de la secuencia de saltos seleccionada. La fase de la secuencia de saltos se determina mediante el reloj bluetooth. Todos los canales físicos se subdividen en ranuras de tiempo cuya longitud depende del canal físico.

- **Canal físico básico de la piconet**. La definición del canal físico básico de la piconet corresponde al maestro de la piconet. El maestro controla el tráfico del canal físico de la piconet mediante una secuencia de consultas.

  Por definición, el dispositivo que inicia una conexión mediante una búsqueda de terminales hace las veces de maestro. Una vez establecida la piconet, es posible intercambiar las funciones de maestro y esclavo.

  El canal físico básico de la piconet se divide en ranuras de tiempo de 625 μs de duración cada una.
- **Canal físico adaptado de la piconet**. Los canales físicos adaptados de la piconet pueden usarse para dispositivos conectados con la función de salto adaptable de frecuencia (AFH) activada. Hay dos diferencias entre los canales físicos básicos y adaptados de la piconet. La primera es el mismo mecanismo del canal que hace que la frecuencia esclava sea la misma que la de la transmisión maestra precedente. La segunda diferencia es que el canal físico adaptado de la piconet

puede no basarse en la totalidad de las 79 frecuencias del canal físico básico de la piconet.

- **Canal físico de detección de paginación**. Si bien las funciones de maestro y esclavo no se definen antes de la conexión, se denomina maestro al dispositivo de paginación (que se convierte en maestro en el estado de CONEXIÓN) y esclavo al dispositivo que realiza la detección de la paginación (que actúa como esclavo en el estado CONEXIÓN).

  El canal físico de detección de paginación sigue una secuencia de saltos más lenta que el canal físico básico de la piconet, y adopta una secuencia corta de saltos pseudo-aleatorios entre los canales RF.
- **Canal físico de detección de búsqueda**. Si bien las funciones de maestro y esclavo no se definen antes de la conexión, se denomina maestro al dispositivo de búsqueda y esclavo al dispositivo que realiza la detección de la búsqueda.

  El canal físico de detección de búsqueda usa una secuencia de saltos más lenta que el canal físico de la piconet, y adopta una secuencia corta de saltos pseudo-aleatorios entre los canales RF.

❖ **Enlaces físicos**

Un enlace físico representa una conexión de banda base entre dispositivos. Este tipo de enlace se asocia a tan sólo un canal físico. Los enlaces físicos tienen propiedades comunes que se aplican a todas las comunicaciones lógicas del enlace físico. Las propiedades comunes de los enlaces físicos son:

- Control de energía.
- Supervisión de enlaces.
- Cifrado.
- Cambio de velocidad de transmisión dictado por la calidad del canal.
- Control de paquetes en múltiples ranuras.
- Comunicaciones lógicas.

Pueden establecerse diferentes tipos de comunicaciones lógicas entre el dispositivo maestro y los dispositivos esclavos. Se han definido cinco comunicaciones lógicas:

- Comunicación lógica por conexión síncrona (SCO).
- Comunicación lógica por conexión síncrona ampliada (eSCO).
- Comunicación lógica por conexión asíncrona (ACL).
- Comunicación lógica por difusión del dispositivo esclavo activo (ASB).
- Comunicación lógica por difusión del dispositivo esclavo en espera (PSB).

❖ **Enlaces lógicos**

Se definen cinco enlaces lógicos:

- Control del enlace (LC).
- Control ACL (ACL-C).
- Isócrono o asíncrono del usuario (ACL-U).
- Síncrono del usuario (SCO-S).
- Síncrono ampliado del usuario (eSCO-S).

Los enlaces lógicos de control LC y ACL-C se usan al nivel del control de enlaces y del gestor de enlaces, respectivamente. El enlace lógico ACL-U se utiliza para transmitir información síncrona o asíncrona del usuario. Los enlaces lógicos SCO-S y eSCO-S se usan para transmitir información síncrona del usuario. El enlace lógico LC se transmite en la cabecera del paquete, todos los demás enlaces lógicos se transmiten en la carga útil del paquete. Los enlaces lógicos ACL-C y ACL-U están indicados en el campo de la ID del enlace lógico (LLID) en la cabecera de la carga útil.

Los enlaces lógicos SCO-S y eSCO-S son transmitidos exclusivamente por las comunicaciones lógicas síncronas. El enlace ACL-U se transmite normalmente por la comunicación lógica ACL; sin embargo, también puede ser transmitido por los datos del paquete DV de la comunicación lógica SCO. El enlace ACL-C puede transmitirse por cualquiera de las dos comunicaciones lógicas SCO o ACL.

2.3.3.3.3. Pila de protocolos

La especificación de bluetooth establece que haya una unificación entre dispositivos de diferentes fabricantes y que así las diferentes aplicaciones puedan operar correctamente. Para ello es necesario que todos los dispositivos que vayan a participar en las comunicaciones ejecuten la misma pila de protocolos.

La pila de protocolos es la que se muestra en la siguiente figura, la cual podemos ver que esta formada por protocolos específicos de bluetooth, como por ejemplo son *Link Manager* (LM) y *Logical Link Control Adaption Protocol* (L2CAP), así como por otros no específicos como pueden ser TCP, UDP, IP, OBEX, etc.

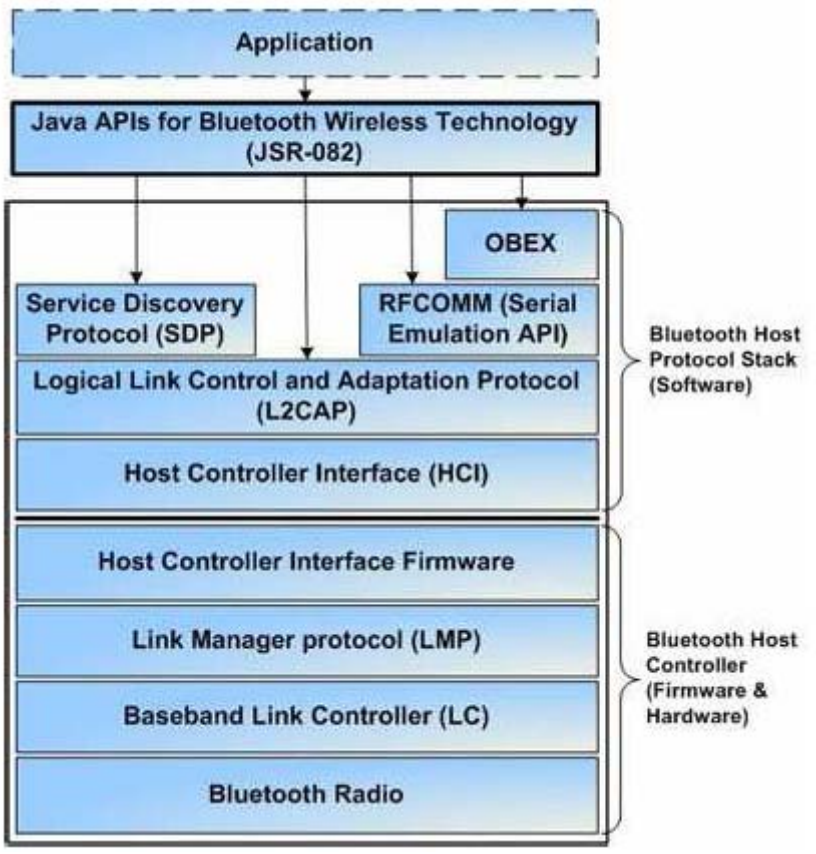


Figura 9 – Pila de protocolos de bluetooth

Además de estos protocolos, se define el HCI (*Host Controller Interface*), encargado de proporcionar al controlador de banda base y al *Link Manager* una interfaz de comandos que nos permite acceder a los registros de control y al estado del hardware.

❖ **Link Manager (LM)**

Es el protocolo encargado de gestionar el establecimiento de la conexión entre los dispositivos, la autentificación y la configuración del enlace. El LM se encarga de localizar y comunicarse con otros gestores por medio del protocolo LMP (*Link Manager Protocol*), que en resumidas cuentas consiste en el envío de PDUs (*Protocol Data Units*) entre los dispositivos.

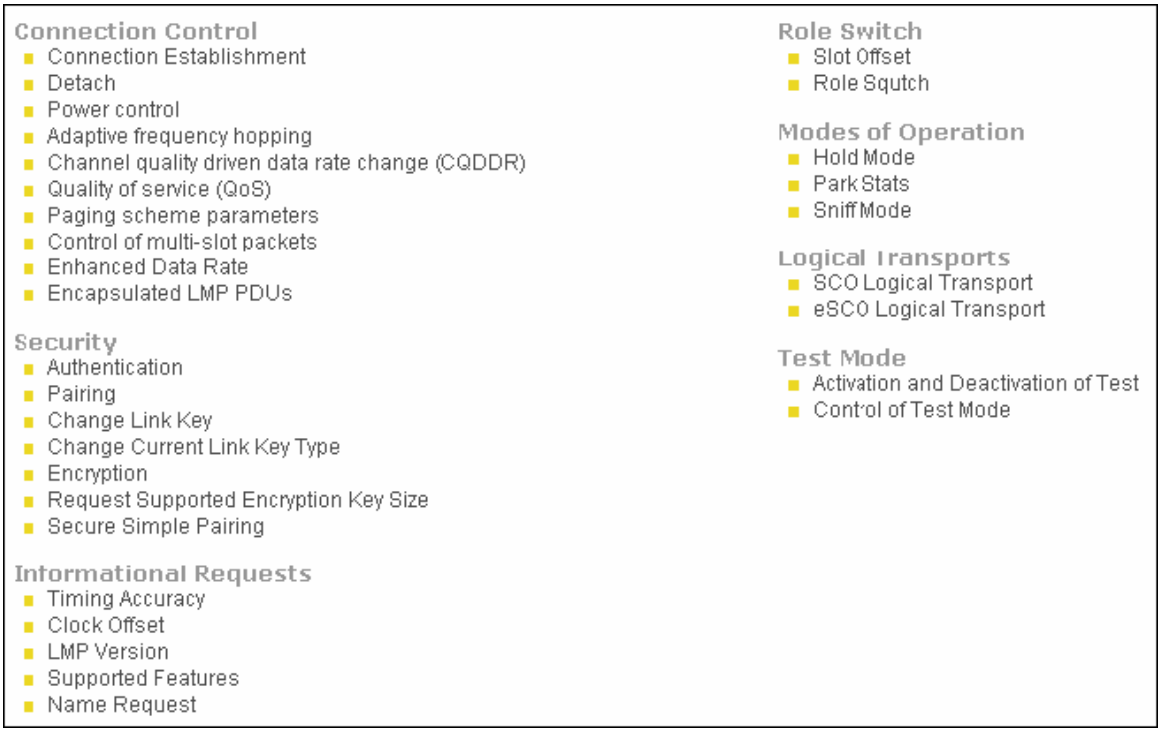


Figura 10 – PDUs definidos en el protocolo LMP

Una vez establecida la conexión, se procede a anunciar los servicios soportados:

- Transmisión y recepción de datos.
- Petición de nombre o ID (longitud máxima 16 caracteres).
- Petición de las direcciones de enlace.
- Establecimiento de la conexión.
- Autentificación.
- Negociación del modo de enlace y establecimiento, por ejemplo, modo datos o modo voz y datos. Esto puede cambiarse durante la conexión.

❖ **Host Controller Interface (HCI)**

HCI proporciona una interfaz de comandos entre el controlador de la banda base y el gestor de enlaces y acceso a los parámetros de configuración. Esta interfaz proporciona un método uniforme de acceso a todas las funciones de la banda base bluetooth.

La capa HCI de la máquina intercambia comandos y datos con el firmware del HCI presente en el dispositivo bluetooth. El driver de la capa de transporte de la controladora de la máquina (es decir, el driver del bus físico) proporciona ambas capas de HCI la posibilidad de intercambiar información entre ellas.

Una de las tareas más importantes de HCI que se deben realizar es el descubrimiento automático de otros dispositivos bluetooth que se encuentren dentro del radio de cobertura. Esta operación se denomina en inglés *inquiry* (consulta). Es importante tener en cuenta que un dispositivo remoto sólo contesta a la consulta si se encuentra configurado en modo visible (*discoverable mode*).

El sistema bluetooth proporciona una conexión punto a punto (con sólo dos unidades bluetooth involucradas) o también una conexión punto – multipunto. En el último caso, la conexión se comparte entre varios dispositivos bluetooth.

- **Logical Link Control and Adaptation Protocol (L2CAP).**

El protocolo L2CAP proporciona servicios de datos tanto orientados a conexión como no orientados a conexión a los protocolos de las capas superiores, junto con facilidades de multiplexación, segmentación y reensamblaje. L2CAP permite que los protocolos de capas superiores puedan transmitir y recibir paquetes de datos L2CAP de hasta 64 kilobytes de longitud.

L2CAP se basa en el concepto de canales. Un canal es una conexión lógica que se sitúa sobre la conexión de banda base. Cada canal se asocia a un único protocolo. Cada paquete L2CAP que se recibe en un canal se redirige al protocolo superior correspondiente. Varios canales pueden operar sobre la misma conexión de banda base, pero un canal no puede tener asociados más de un protocolo de alto nivel.

- **RFCOMM.**

El protocolo RFCOMM proporciona emulación de puertos serie a través del protocolo L2CAP. Es un protocolo de transporte sencillo, con soporte para hasta 9 puertos serie RS-232 y permite hasta 60 conexiones simultáneas (canales RFCOMM) entre dos dispositivos.

Para los propósitos de RFCOMM, un camino de comunicación involucra siempre a dos aplicaciones que se ejecutan en dos dispositivos distintos (los extremos de la comunicación). Entre ellos existe un segmento que los comunica. RFCOMM pretende cubrir aquellas aplicaciones que utilizan los puertos serie de las máquinas donde se ejecutan. El segmento de comunicación es un enlace bluetooth desde un dispositivo al otro (conexión directa).

RFCOMM trata únicamente con la conexión de dispositivos directamente, y también con conexiones entre el dispositivo y el módem para realizar conexiones de red. RFCOMM puede soportar otras configuraciones, tales como módulos que se comunican vía bluetooth por un lado y que proporcionan una interfaz de red cableada por el otro.

❖ **OBEX (Object Exchange).**

Es un protocolo de comunicaciones que facilita el intercambio de objetos binarios entre dispositivos. Es mantenido por la *Infrared Data Association*[6] (IrDA) pero ha sido adoptada también por el SIG de bluetooh y por la *Open Mobile Alliance*[7] (OMA). Una de las primeras aplicaciones populares de OBEX tuvo lugar en la PDA Palm III. Esta PDA y sus múltiples sucesoras utilizaron OBEX para intercambiar tarjetas de negocio, datos e incluso aplicaciones.

OBEX es similar en diseño y funcionalidad a HTTP, protocolo en el que el cliente utiliza un transporte fiable para conectarse a un servidor y así recibir o proporcionar objetos. No obstante, OBEX difiere en algunos puntos importantes:

- **Transporte**. HTTP funciona normalmente sobre un puerto TCP/IP. OBEX, en cambio, es comúnmente implementado sobre una pila IrLAP/IrLMP/Tiny TP de un dispositivo de infrarrojos; mientras que funcionando con bluetooth, se suele implementar sobre una pila en Banda Base/Link Manager/L2CAP/RFCOMM. En cualquier caso, ofrece otras posibilidades.
- **Transmisiones binarias**. Para intercambiar información sobre una petición o un objeto, HTTP utiliza texto legible por el ser humano, mientras que OBEX utiliza tripletes binarios llamados cabeceras. Éstos, resultan más simples de elaborar para dispositivos con características limitadas.
- **Soporte para realizar sesiones**. Las transacciones HTTP carecen inherentemente de estado. Generalmente, un cliente HTTP establece una conexión, efectúa una sola petición, recibe respuesta y cierra la conexión. En OBEX, una sola conexión de transporte podría utilizarse para efectuar varias operaciones relacionadas entre sí. De hecho, las últimas novedades de la especificación OBEX permiten almacenar la información del estado de una conexión intacta incluso si la conexión finalizó inesperadamente.

❖ **Protocolo de Descubrimiento de Servicios (SDP).**

Este protocolo permite a las aplicaciones cliente descubrir la existencia de diversos servicios proporcionados por los servidores de aplicaciones, así como el conjunto de los atributos y propiedades de éstos. Estos atributos de servicio incluyen el tipo o clase de servicio ofrecido y el mecanismo o la información necesaria para utilizar dichos servicios.

El servidor mantiene una lista de registros de servicios, los cuales describen las características de los servicios ofrecidos. Cada registro contiene información sobre un determinado servicio. Un cliente puede recuperar la información de un registro de servicio almacenado en un servidor SDP lanzando una petición SDP. Si el cliente o la aplicación asociada con el cliente deciden utilizar un determinado servicio, debe establecer una conexión independiente con el servicio en cuestión. SDP proporciona un mecanismo para el descubrimiento de servicios y sus atributos asociados, pero no proporciona ningún mecanismo ni protocolo para utilizar dichos servicios.

Normalmente, un cliente SDP realiza una búsqueda de servicios acotada por determinadas características. No obstante hay momentos en los que resulta deseable

[6] IrDA define un estándar físico en la forma de comunicación (transmisión y recepción) de datos por rayos infrarrojos. IrDA se crea en 1993 entre *HP*, *IBM*, *Sharp* y otras empresas.
[7] OMA es una organización que desarrolla estándares abiertos para la industria de la telefonía móvil.

descubrir todos los servicios ofrecidos por un servidor SDP sin que pueda existir ningún conocimiento previo sobre los registros que pueda contener. Este proceso de búsqueda de cualquier servicio ofrecido se denomina navegación o *browsing*.

2.3.3.4 Principales usos y aplicaciones

Hoy en día, la aparición de bluetooth ha permitido cambiar radicalmente la forma en la que los usuarios interactúan con los teléfonos móviles y otros dispositivos, dando lugar así a un gran numero de aplicaciones y usos de esta tecnología.

Desde el punto de vista de los usuarios, bluetooth supone una tecnología que permite una comunicación fácil, instantánea, rápida, en cualquier lugar y que además su coste es bajo; sin olvidar su impacto en la forma de realizar los procesos, al sustituir los medios convencionales y posibilitar nuevos negocios y aplicaciones.

La aplicación de esta tecnología se puede percibir desde la implementación de una red inalámbrica hasta la posibilidad de transferir una fotografía de una cámara a un móvil para enviarla por correo electrónico o transferirla a la impresora para imprimirla en ausencia de cables. Desde el punto de vista profesional, la aplicación más práctica, es la posibilidad de montar una red inalámbrica en salas o entornos que ofrezcan dificultades para montar una LAN convencional. Para ello se utiliza un punto de acceso y cada puesto lleva instalado una *PC Card* con esta tecnología. A continuación se detallan algunas de las aplicaciones más importantes de bluetooth:

- **Transferencia de archivos**. El servicio consiste en la transferencia de archivos .doc, .xls, .ppt, .wav, y .jpg, carpetas y directorios de un dispositivo a otro. Además ofrece la posibilidad de ver el contenido de carpetas de dispositivos remotos.
- **Escritorio Inalámbrico**. Bluetooth nos ofrece la posibilidad de eliminar los cables que utilizamos en nuestros equipos: desde un teclado inalámbrico, pasando por el ratón, incluso un disco duro portátil que se comunique mediante esta tecnología.
- **Conexión a Internet**. Esta aplicación permite conexión inalámbrica, un usuario tiene acceso a Internet mediante un teléfono móvil o mediante un módem inalámbrico, tal cual como si fuera una línea telefonía fija.
- **Acceso Inalámbrico a LAN**. En este servicio, múltiples equipos terminales de datos usan puntos de acceso LAN, llamados LAP (*LAN Access Point*), como una conexión inalámbrica a una red de área local LAN. Una vez conectados, funcionan como si estuvieran conectados a una LAN vía red.
- **Sincronización Automática**. El servicio consiste en sincronizar automáticamente y de manera continua la información PIM (*Personal Information Managament*) en los distintos dispositivos bluetooth; básicamente la información es la concerniente a calendario, lista de direcciones y teléfonos, mensajes y notas.
- **Teléfono tres en uno**. Un mismo teléfono se puede utilizar como fijo, si se encuentra dentro del radio de acción del punto de acceso instalado, como teléfono móvil si nos encontramos fuera de radio de acción del punto de acceso, y por último, como medio de acceso a nuestros contactos, teléfonos, correo electrónico, etc.
- **Dispositivo Manos Libres Inalámbrico**. El dispositivo manos libres puede conectarse de manera inalámbrica al teléfono móvil, al ordenador portátil u otro

móvil, con el fin de actuar como un dispositivo remoto con entrada y salida de audio.

Como se puede apreciar, las aplicaciones de bluetooth son casi infinitas y permiten cambiar radicalmente la forma en la que los usuarios interactúan con los teléfonos móviles y otros dispositivos. Una de las primeras compañías en lanzar un producto bluetooth ha sido *Ericsson*.

Alianzas como las de *Nokia* y *Fuji* permitirán a los propietarios de cámaras digitales hacer fotos para luego transmitirlas a través del móvil a la impresora situada en casa o al disco duro del ordenador. Mientras, compañías como *Motorola* y *JVC* desarrollan continuamente tecnologías aún más avanzadas que harán estos avances extensibles al vídeo o al DVD. La compañía *Sony* ha hecho posible la implantación de microchips bluetooth en toda su gama de productos. En sólo un par de años caminaremos escuchando música en un reproductor MP3 mientras descargamos nuevas canciones y actualizamos el repertorio musical a través del móvil. La utilidad de bluetooth sólo está delimitada por la imaginación de los ingenieros y los usuarios.

Entre otras aplicaciones, bluetooth permite conectar cámaras de vigilancia, servir con mandos a distancia, permite utilizar un teléfono celular como inalámbrico, para abrir puertas, conectar electrodomésticos, pasar ficheros MP3 del móvil al PC, y por supuesto, para conectar todo tipo de dispositivos a Internet, formando puntos de acceso. Encuentra aplicación en la industria de automoción, en medicina para monitorización de los enfermos sin necesidad de tener cables conectados a su cuerpo, automatización del hogar, lectura de contadores, asociado a un lector de código de barras, etc.

En definitiva, bluetooth se esta convirtiendo en una tecnología de uso cotidiano, y sus características le han permitido ser utilizado en numerosos campos de aplicación, y muchos mas que llegaran en un futuro.

2.3.3.5 BlueCove (23)

El estádar JSR 82 está definido por *Java Community Process* (JCP) para el desarrollo de aplicaciones bluetooth únicamente en J2ME. Pues bien, BlueCove es una librería que aporta la implementación de JSR 82 para la plataforma J2SE o, lo que es lo mismo, para ordenadores personales de sobremesa o portátiles. Actualmente funciona con los siguientes controladores bluetooth correspondientes a:

- Mac OS X.
- WIDCOMM.
- BlueSoleil.
- Pila Bluetooth de Microsoft (Winsock)[8] (24), a partir de Windows XP SP2.

Originalmente fue desarrollado por *Intel Research* y en la actualidad está gestionado por desarrolladores voluntarios.

[8] Este es el controlador compatible empleado por el dispositivo bluetooth del portátil para el desarrollo.

2.3.4 NFC (*Near Field Communication*)

Entre las tecnologías inalámbricas de corto alcance o proximidad, ha surgido un gran interés en los últimos años por la tecnología *Near Field Communication* (NFC). Es una tecnología de plataforma abierta que comenzó a desarrollarse en el año 2002 en una acción conjunta de *Philips* y *Sony*, aprobada como estándar global ISO/IEC en Diciembre de 2003, con el fin de conseguir un protocolo compatible con las tecnologías *contactless* propietarias existentes en el mercado, FeliCaTM (*Sony*) y MifareTM (*Philips*).

La tecnología NFC es una tecnología de interconexión de dispositivos que funciona a través de inducción magnética en un pequeño rango de conectividad *wireless*, opera a la frecuencia de 13,56 MHz (banda en la que no se necesita licencia administrativa), a una distancia de pocos centímetros (de 0 a 20 cm., típicamente 10 cm.). Surge de la evolución de la tecnología de identificación de dispositivos por radiofrecuencia (RFID, *Radio Frecuency IDentification*), incluyendo funciones de las tecnologías de interconexión de redes y de tarjetas inteligentes.

En lo que resta de esta sección, trataremos los antecedentes de NFC, describiremos sus características, comparandolas con las de otras tecnologías inalámbricas, cómo funciona y en qué puede ser utilizado.

2.3.4.1 Un poco de historia, RFID

2.3.4.1.1. ¿Qué es RFID?

RFID responde a las siglas de *Radio-Frequency IDentification*. Es un método automático de identificación, que permite extraer y guardar datos remotamente usando los dispositivos conocidos como etiquetas (o transpondedores).

Una etiqueta de RFID es un objeto el cual puede ser aplicado o incorporado en un producto, un animal o una persona con el fin de la identificación usando ondas de radio. Algunas etiquetas se pueden leer a muchos metros del lector.

La mayoría de las etiquetas de RFID contienen por lo menos dos partes. Una es un circuito integrado que permite salvar y procesar la información, modular y demodular una señal de radiofrecuencia (RF) y otras funciones especializadas. La segunda es una antena para recibir y transmitir la señal.

Hay otro tipo de tarjetas, *Chipless RFID*, que permiten la identificación sin circuito integrado, de tal modo que se puedan implantar directamente sobre etiquetas, lo que supone un coste menor que las otras etiquetas tradicionales.

2.3.4.1.2. Campos de aplicación

Hoy en día, el uso de RFID está llegando a ser cada vez más frecuente, en gran medida debido a la reducción del precio de esta tecnología, lo que le esta permitiendo poder ser aplicada en una gran variedad de campos.

- **Medición en carreras**

La tecnología RFID ha sido incorporada a esta materia muy recientemente. Gracias a ello, se pueden registrar los tiempos de numerosos participantes en carreras donde la

participación es muy numerosa o donde es muy tediosa o difícil la medida de estos tiempos para cada uno de ellos.

Cada individuo lleva un dorsal con su número de participante y además una etiqueta RFID integrada que puede ser leída por la antena colocada en la meta (o en las zonas de paso). Los errores y accidentes en las mediciones se evitan puesto que cualquier persona puede comenzar y acabar en cualquier momento sin tener que ser tratado en modo de lotes.

Este método está siendo adoptado por muchas agencias del reclutamiento que tienen un PET (prueba de resistencia física) como procedimiento calificativo, especialmente en caso de que el volumen de candidatos sea muy grande.

- **Pasaportes**

Las etiquetas de RFID se están utilizando en los pasaportes expedidos por muchos países, como Malasia (principios de 2000), Nueva Zelanda (2005), Bélgica, los Países Bajos (2005), Noruega (2005), Irlanda (2006), Japón (2006), Paquistán, Alemania, Portugal, Polonia (2006), el Reino Unido, Australia y los Estados Unidos (2007).

Los primeros pasaportes de RFID ("e-pasaportes") fueron publicados en Malasia. Además de la información personal contenida en las páginas del pasaporte, éstos incluyen un registro de entradas y salidas del país (hora, fecha y lugar).

Inicialmente se establecía que las etiquetas podrían leerse solamente a una distancia de 10 centímetros, pero más tarde se vio que pueden leerse los pasaportes hasta una distancia de 10 metros.

Los pasaportes fueron diseñados para incorporar una fina lámina de metal que evitara que programas de lectura desautorizados accedan a la información cuando el pasaporte se cierra. Otro método de seguridad era que antes de que la etiqueta de un pasaporte pudiera ser leída, ésta debía ser registrada en un programa de lectura de RFID.

Algunos otros países europeos están planeando agregar huellas digitales y otros datos biométricos, mientras que otros ya lo han hecho.

- **Pago en transportes**

Hoy en día, en un gran numero de países, tanto de Europa, Asia y América, han incorporado esta tecnología en el sector de los transportes. Uno de estos usos implantados es el sistema de pago en peajes a través de etiquetas RFID. En muchas ciudades esta implantado el pago en el transporte publico con éstas tarjetas (Metrorail en USA, SUICa en Japón, etc.).

Estos nuevos usos permiten automatizar tareas, ahorrar tiempo en el cobro a los usuarios, además de suponer un ahorro de personal en algunos otros casos.

- **Seguimiento y control de mercancías**

En este campo, podemos ver como las mercancías han comenzado a incluir etiquetas RFID, lo que permite una lectura rápida en controles o a la entrada de grandes almacenes o mercados. Esto agiliza mucho el proceso de conteo de mercancías, pues con el método tradicional era necesario hacer un inventario. Pero con RFID el control de la mercancía es

tan sencillo como que el camión que la transporta pase por un debajo del lector de radiofrecuencia situado en las puertas del almacén.

❖ **Identificación animal**

Muy conocido es el uso de los chips en animales domésticos. Hoy en día es muy común que nuestras mascotas, como por ejemplo perros y gatos, posean un microchip incorporado con información personal acerca del animal. Esto es muy útil cuando un usuario va con su mascota a un veterinario pues se puede obtener de manera muy rápida una gran cantidad de información acerca del animal. También es de gran utilidad en el caso de animales extraviados o perdido, pues basta con leer su chip para poder localizar al dueño.

❖ **Implantes humanos**

Esta es una técnica no muy utilizada a día de hoy, pero que quizás en un futuro se empiece a emplear en numerosas aplicaciones. El implante consiste en un chip similar al que se le pone a las mascotas y contiene información del portador. Algunas discotecas o clubes de Barcelona y Rótterdam han permitido el uso de microchips en sus clientes VIP que lo emplean para el pago automático de bebidas.

Una posible aplicación sería la identificación de personas, aunque expertos en seguridad desaconsejan este uso debido a la relativa facilidad en que un usuario podría robarle la identidad a otro mediante un ataque de *man-in-the-middle*.

❖ **Tiendas y grandes superficies**

Este uso lleva mucho tiempo extendido, sustituyendo al tradicional código de barras por etiquetas RFID, en productos como libros, cd's o dvd's. La incorporación de estas etiquetas no solo permite el acceso a la información del producto mediante la lectura de su etiqueta si no que también actúa como elemento de seguridad.

❖ **Otros usos**

Se está empezando a usar RFID en museos, hospitales, zoológicos, campos de golf, casinos, etc. Como se puede ver, RFID es una tecnología que puede aplicarse a un gran número de campos cada vez mayor.

2.3.4.2 Principales características de NFC

Detallamos a continuación las características fundamentales de la tecnología NFC, recapitulando las ya mencionadas:

- NFC es una solución desarrollada por *Nokia*, *Sony* y *Philips* y esta basada en RFID y tecnologías interconectadas.
- Trabaja en la banda de los 13,56 Mhz, por lo que no se le aplica ninguna restricción y no requiere ninguna licencia para su uso.
- Las velocidades de transmisión soportadas actualmente son de 106, 212 y 424 Kbps.
- Se puede usar para configurar e iniciar otras conexiones *wireless* como son bluetooth, Wi-Fi o UltraWireband.
- Cuando se enciende el lector, emite una señal de radio de corto alcance que activa el microchip de la etiqueta, con lo que podremos leer una pequeña cantidad de datos que se encuentra almacenada en ella.

- Tecnología inherentemente segura, ya que por su corto alcance los dispositivos tienen prácticamente que tocarse.
- Proporciona un modo de acceso a los servicios muy familiar e intuitivo a los usuarios: "si quieres un servicio, tócalo". Por ejemplo, si quieres comprar un refresco con el terminal, "toca la máquina". Al contrario de lo que ocurre con los servicios por RFID o bluetooth, basados en el descubrimiento de la presencia del dispositivo en la proximidad.
- No precisa configuración por parte del usuario.

Figura 11 – Acción de "tocar" para compartir, informarse, pagar o picar billete (ver *2.3.4.4 Usos y aplicaciones*)

A continuación, se ilustra en forma de tabla una comparativa de las caracterisitcas de tecnologías inalámbricas similares:

	NFC	RFID	IrDa	Bluetooth
Set -up time	<0.1ms	<0.1ms	~0.5s	~6 sec
Range	Up to 10cm	Up to 3m	Up to 5m	Up to 30m
Usability	Human centric Easy, intuitive, fast	Item centric Easy	Data centric Easy	Data centric Medium
Selectivity	High, given, security	Partly given	Line of sight	Who are you?
Use cases	Pay, get access, share, initiate service, easy set up	Item tracking	Control & exchange data	Network for data exchange, headset
Consumer experience	Touch, wave, simply connect	Get information	Easy	Configuration needed

Tabla 5 – Comparación entre tecnologías inalámbricas

2.3.4.3 Modos de funcionamiento

NFC soporta dos modos de funcionamiento[9], estos son:

- **Pasivo**. Sólo un dispositivo genera el campo electromagnético y el otro se aprovecha de la modulación de la carga para poder transferir los datos. El iniciador de la comunicación es el encargado de generar el campo electromagnético.

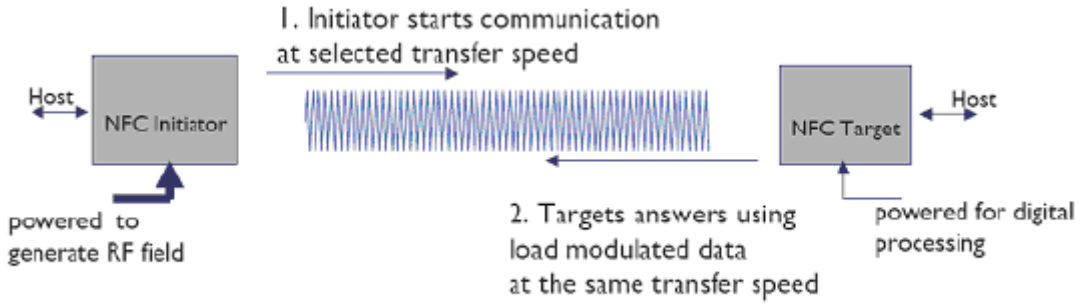


Figura 12 – Modo pasivo de funcionamiento NFC

- **Activo**: Ambos dispositivos generan su propio campo electromagnético, que utilizarán para transmitir sus datos. Ambos dispositivos necesitan energía para funcionar.

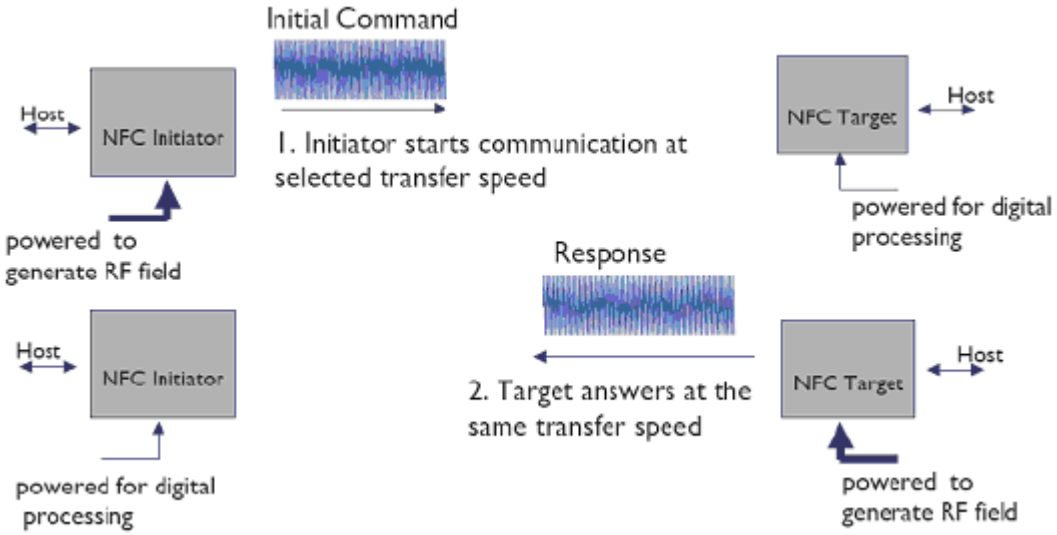


Figura 13 – Modo activo de funcionamiento NFC

2.3.4.4 Usos y aplicaciones

La tecnología NFC está principalmente destinada a ser utilizada con teléfonos móviles. Hay tres principales casos de uso para NFC:

[9] Todos los dispositivos del estándar NFCIP-1 deben soportar ambos modos.

- **Tarjeta de emulación**. El dispositivo NFC se comporta como una tarjeta de contacto, simplementa para su lectura.
- **Modo lector**. El dispositivo NFC está activo y permite leer una etiqueta RFID, por ejemplo para la publicidad interactiva.
- **Modo P2P**[10]. Dos dispositivos NFC se comunican e intercambian información.

Con ello, multitud de aplicaciones son posibles, tales como:

❖ **Pago a través del móvil**

El dispositivo actúa como un débito o crédito de tarjetas de pago. Los usuarios podrán realizar transacciones económicas tan sólo acercando el dispositivo al terminal de cobro, sin tener que teclear nada en el móvil. Además, los teléfonos incorporarán una serie de características de seguridad para proteger los datos financieros y garantizar la seguridad de las comunicaciones.

Para mayor funcionalidad, se podría facilitar un historial de transacciones fácilmente accesible por el usuario consumidor a través del mismo dispositivo. En esta línea, el acceso a la red móvil puede facilitar la recarga de la tarjeta de pago mediante una sencilla opción del interfaz de usuario, o incluso completar dicho servicio mediante la gestión financiera total del usuario a través de dicho interfaz, teniendo la posibilidad, por ejemplo, de realizar transferncias o similares.

Pueden surgir otros múltiples aplicativos en este escenario, como añadir a la tarjeta de crédito un elemento NFC que permita al usuario una forma de pago compatible con algún punto de venta en concreto, además de poder optar entre varias tarjetas de pago en un mismo dispositivo. De esta manera, el dispositivo móvil se convierte en una cartera virtual, llevando un número de tarjetas diferentes, algunas de crédito, algunas de débito, etc.

Figura 14 – Pago con NFC

[10] Este será el caso de uso que emplearemos en nuestro proyecto, cuyo término en el ámbito de la implementación es protocolo NFCIP.

❖ **Billetes de transporte u otros en el móvil**

Muchas de las principales ciudades del mundo usan el sistema de pago sin contacto dentro de la infraestructura de transporte. Estos sistemas se basan en las tarjetas del sistema de Identificación por Radiofrecuencia (RFID) para facilitar el acceso sostenible a los servicios de transporte y la rapidez y comodidad de pago. Consiste simplemente en una extensión de la infraestructura existente, aportando la tecnología sin contacto y evitando la necesidad de poseer un billete físico propiamente dicho.

Típicamente, un usuario compra una tarjeta de plástico con un cierto valor monetario en un chip incrustado en la tarjeta. A medida que el usuario accede al sistema de transporte público u otros servicios, el costo del pasaje se va tomando de la tarjeta, lo que deja un nuevo saldo en la misma. Una vez que la tarjeta no tiene valor, el usuario puede optar entre deshacerse de ella o recargar el saldo mediante la adición de más dinero a la tarjeta. Este planteamiento tiene grandes beneficios en términos de facilidad de uso y velocidad de acceso.

Estos son motivos más que suficientes por los que las tarjetas de contacto han comenzado una nueva era para el transporte y venta de entradas. Sin embargo, con la inclusión de NFC en los teléfonos móviles los usuarios pueden comprar tickets o recargarlos remotamente a través de la interfaz dispuesta. Así, mediante la recarga *online* o las tarifas de acceso mensual, también se garantizan menos colas en las cabinas de venta de billetes, además de tratarse de una medida más sostenible.

Adicionalmente, el teléfono también puede utilizar la tecnología actual de redes de telefonía móvil para acceder a información de trayectos del transporte público, mapas, última información sobre el tráfico, espectáculos,…

El Nokia 6131 NFC tiene una funcionalidad similar a las tarjetas inteligentes estándar de contacto que se usan en todo el mundo como tarjetas de crédito y tickets para sistemas de transporte público. Las aplicaciones de pago y expedición de billetes se almacenarán en el elemento seguro del dispositivo NFC. El elemento seguro es un pequeño chip capaz de almacenar múltiples aplicaciones, como por ejemplo la tarjeta SIM, tarjeta de memoria segura u otras tarjetas inteligentes adicionales incluidas en el dispositivo NFC.

En definitiva, mediante la sustitución de una tarjeta inteligente por un dispositivo móvil NFC, los usuarios pueden acceder a todos los mismos servicios a través de la interfaz de usuario y el acceso a la red móvil, además de optar a información adicional.

Figura 15 – Picar billete con NFC

❖ **Intercambio de información**

Con NFC seremos capaces de recoger información de nuestro entorno. El teléfono móvil se puede utilizar para leer las etiquetas RFID en carteles y pósters con el fin de obtener información sobre éste o, dicho de otro modo, NFC permite a los dispositivos móviles leer información cargada en las etiquetas o tarjetas NFC. Por ejemplo en carteles "inteligentes", señales de las paradas de bus, medicinas, certificados, alimentos envasados y muchos más.

Así, un mensajero podría tocar en la entrega de un paquete local para recibir información sobre el tráfico y orientaciones del siguiente punto de recogida. Añadiendo tarjetas NFC a los carteles y anuncios de revistas, los lectores pueden tener acceso a los servicios móviles ya existentes, como las líneas directas, SMSs (del tipo "envía PARTICIPO al…") y red o Internet basado en contenidos y servicios para teléfonos NFC.

Como el dispositivo móvil aboga cada vez más a convertirse en el hogar de los contenidos digitales, se hará más apremiante la posibilidad de compartir fácilmente este contenido. NFC puede permitir un entorno en el que la gente pueda tocar los dispositivos para compartir tarjetas de visita, para pasar sus fotografías a una impresora o para compartir su música con un amigo. La tecnología NFC tiene el potencial de impactar en la comercialización y las promociones de la industria, desde que "el tocar" de manera activa por parte de los usuarios muestra un interés de recibir información de un producto o servicio.

Figura 16 – Intercambio de información con NFC

Figura 17 – Lectura de información con NFC

- **Emparejamiento bluetooth**

Actualmente, un gran número de actividades asociadas con la transferencia de datos entre dispositivos requiere cierto grado de interacción con el usuario para que se realicen. Por ejemplo, muchos dispositivos con bluetooth requieren de un proceso de emparejamiento que tendrá lugar antes de que sus servicios se puedan utilizar. A pesar de ser un proceso relativamente sencillo, puede que las funcionalidades dependientes de la comunicación bluetooth requieran ser inmediatamente accesibles y el proceso de emparejamiento puede inhibir el uso de la tecnología.

La más actual especificación básica para el estándar de bluetooth incluye la capacidad de emparejar dispositivos vía NFC; simplemente será necesario activar el bluetooth en todos ellos, juntarlos y aceptar el proceso de emparejamiento. El proceso de búsqueda, espera, emparejamiento y autorización puede ser reemplazado por tan sólo un simple "contacto" de los teléfonos móviles y la confirmación del proceso de emparejamiento en cada uno de ellos.

De forma similar pueden los usuarios de NFC obtener acceso a una red inalámbrica. En lugar de la larga duración del proceso de búsqueda del punto de acceso, un usuario puede simplemente tocar un punto de red WLAN compatible con NFC y todo el proceso podría ser automatizado, incluyendo el pago de cualquier coste mediante el monedero virtual del dispositivo.

- **Gestión de identidad y procesos de negocios**

En multitud de oficinas o fábricas el personal de trabajo está obligado a llevar una etiqueta de identidad para acceder a los locales de trabajo. Esto se debe a que, como las empresas se hacen cada vez más complejas y de carácter mundial, muchos trabajadores necesitan tener acceso a múltiples locales restringidos.

La gestión de este proceso puede ser complicada, más aún en ambientes donde hay diferentes niveles de seguridad para los diferentes trabajadores. NFC puede permitir que la gestión de la identidad que se añade a un dispositivo móvil proporcione una única solución integrada para la gestión de la identidad. Así, el dispositivo móvil se puede utilizar para proporcionar acceso a determinados lugares y, por supuesto, negarlo a otros. Como parte de esta solución integral es interesante también, como venimos recalcando en

todas las aplicaciones, que el acceso se pueda actualizar a través de la red inalámbrica o móvil, es decir, que los trabajadores no estén obligados a visitar un sitio físicamente para cambiar su perfil de acceso.

También, un guardia de seguridad podría entrar en contacto con diferentes puntos de reconocimiento, proporcionando un sendero digital de sus movimientos durante una patrulla.

❖ **Otras nuevas aplicaciones**

Muchas aplicaciones en el campo de NFC son extensiones para actualizar y mejorar soluciones existentes. Por ejemplo, soluciones de pago y venta de entradas han sido conformadas a NFC ampliamente en todo el mundo.

Otras aplicaciones podrían incluir:

- Control de acceso físico a lugares o recintos.
- Control de acceso a una red informática.
- Pago en gasolineras (en el mismo surtidor).
- Inclusión de información médica para uso en emergencias.
- Documentos de identidad.
- *Mobile Commerce*.
- Domótica.
- Apertura de vehículos.
- Llaves NFC para acceder a casa o a la oficina o llaves de habitaciones de un hotel.
- Etc.

Figura 18 – Llave NFC

2.3.5 SDKs de *Nokia*

Nokia emplea la tecnología Java, J2ME, para proporcionar una plataforma en la construcción de aplicaciones abierta a los desarrolladores, permitiéndoles crear aplicaciones para dispositivos que soportan dicha tecnología. Con esta plataforma

estándar se da un amplio margen para la creatividad y se libera a los desarrolladores de preocupaciones por las particularidades de cada dispositivo diferente.

Como ya hemos mencionado, entre los medios de desarrollo experimentados para esta parte hemos tenido: *Nokia 6131 NFC SDK* y *S40 Nokia 6212 NFC SDK*. Ambos comparten las siguientes características:

- Permiten al desarrollador crear, emular y ejecutar aplicaciones Java (MIDlets) usando la API de Comunicación Sin Contacto (JSR 257).
    - Este JSR nos permite usar las características NFC.
- Ofrecen soporte para MIDP 2.0.
- Capaces de emular el elemento seguro.
- Emuladores de tecnología bluetooth.
- Incluye el *Nokia Connectivity Framework* (NCF) *Lite*.
- Integración con Eclipse.
- Para facilitar su uso, vienen acompañadas de:
    - Emulador del dispositivo móvil.
    - APIs de Java.
    - MIDlets de ejemplo.
    - Documentación.

2.3.5.1 NFCIP-1 en SDKs de *Nokia*

A pesar de que todavía no se ha estandarizado un protocolo P2P para NFC, *Nokia* incorpora uno en sus extensiones al JSR 257 para las conexiones *peer-to-peer* de NFC, denominado NFCIP. Es un protocolo muy simple de tipo petición-respuesta, donde uno de los dispositivos debe ser el iniciador y el otro actuará como destino.

El paquete *com.nokia.nfc.p2p* contiene la clase *NFCIPConnection* para la comunicación entre dos dispositivos NFC. El modo de conexión se establece cuando la conexión se abre usando la clase *javax.microedition.io.Connector*: el que inicia la conexión debe abrirlo con la URL "nfc:rf; type=nfcip; mode=initiator" y el otro extremo con "nfc:rf; type=nfcip; mode=target". Hay que tener en cuenta que el método *Connector.open (java.lang.String)* permanece en ejecución hasta que se encuentra un dispositivo NFC. No se puede implementar de otro modo, ni siquiera registrando un *TargetListener* en el *DiscoveryManager* para recibir notificaciones sobre un dispositivo NFC. Así, también se puede definir un valor de tiempo de espera (en milisegundos) en la URL de conexión. Por ejemplo, con "nfc:rf; type=nfcip; mode=initiator; timeout=5000" se esperará 5 segundos. Por defecto, si no se indica dicho campo, la llamada de apertura de la conexión se bloqueará indefinidamente, igual que si se indica el valor 0.

En el modo "iniciador" uno tiene primero que realizar un envio, entonces recibe datos, envía datos, recibe datos,… y así se pueden hacer tantas secuencias de envio-recepción como sea necesario. Por cada llamada de envío debe haber una llamada de recepción. En el modo "destino" la condición es análoga, pero recibiendo primero datos y luego enviando. Esto supone, evidentemente, que la conexión NFCIP debe finalizar siempre tras una recepción por parte del que la inicia o, en otras palabras, que se producirá un error si se cierra tras un envío, en sentido iniciador – destino, sin una recepción posterior.

Capítulo 3

Descripción del Sistema: Alto Nivel

Como venimos indicando, el sistema en su conjunto está compuesto por dos partes principalmente, el apartado web y la parte móvil. Según se puede apreciar en la *Figura 1 – Esquema de dimensionado del proyecto* de la introducción, ambas partes conforman una arquitectura cliente-servidor, siendo el equipo que concentra el SI el servidor que presta servicios en los dos ambientes, en uno como servidor web y, en la parte móvil, como servidor bluetooth. De esta forma, se ha debido llevar a cabo todo el desarrollo a través de tres proyectos distintos, que es el término que emplean las IDE en ámbito de programación, considerando cada uno de una parte u otra:

❖ **Web**

- Proyecto de tipo *Web Dinámica*, denominado ***PFCJMAGServidorWeb_v1.0***. Cuya implementación nos la facilitó, en la primera etapa, el *plugin* WTP y, en la segunda, el IDE específico para entorno web *glassfish-tools-bundle-for-eclipse 1.2*. El funcionamiento de este proyecto se sustenta sobre la plataforma J2EE y da como resultado la aplicación web del sistema.

❖ **Móvil**

- Proyecto de tipo *J2ME MIDlet Suite*, denominado ***PFCJMAGMovil_v1.0***. Para esta implementación nos ayudamos del *plugin* eclipseMe, mencionado anteriormente. Como su nombre indica, el proyecto se sustenta sobre la plataforma J2ME y contendrá las aplicaciones del sistema que se ejecutan sobre los dispositivos móviles.
- Proyecto común de Java (J2SE), denominado ***PFCJMAGServidorMovil_v1.0***. Es la aplicación del servidor que atiende al entorno móvil mediante comunicación

bluetooth, por lo que implementa JSR 82 sobre J2SE gracias a BlueCove. Por estas características, ha sido necesario aislarlo de los demás en un proyecto aparte.

Antes de abordar cada una de las partes, es conveniente describir la parte común para ambos proyectos correspondientes al servidor, como es el SI que comparten. Anotar, por el momento, una pequeña diferencia existente en la relación de estos proyectos con el gestor de Bases de Datos MySQL, como es la iteración llevada a cabo a través de alguna conexión, gracias al conector JDBC en ambos casos, pero mediante un conjunto de conexiones en el caso del servidor web configurable en GlassFish. Por otro lado, se verá que no deben existir problemas de concurrencia sobre el recurso de conexión a BD, aunque no se ha tratado la concurrencia sobre los datos, estableciendo simplemente la configuración del gestor de BBDD en modo de ejecución automática (*autocommit*) de forma que cada sentencia SQL conforma una transacción individual por sí misma (25). Ahora sí, nos centraremos en el diseño del modelo de datos compartido.

3.1 Modelo de Datos

En esta parte del diseño hemos tenido en cuenta cierta flexibilidad para la aplicación del sistema. Así, nos hemos propuesto que el Sistema de Información conste de las siguientes especificaciones:

- Constancia de los usuarios del sistema, con información básica acerca de cada cual.
- Diseño de rutas a través de un número indefinido de paradas.
- Asignación de dichas rutas en diferentes fechas a los usuarios, a lo cual se le denomina como *itinerario*, y al que se le puede añadir información acerca de incidencias y de su cumplimiento.

Con estas premisas, ha resultado el modelo de datos ilustrado en el siguiente diagrama relacional:

Figura 19 – Diagrama relacional del modelo de datos

De las relaciones mostradas por el diagrama se puede concluir:

- Un usuario podrá tener múltiples rutas asignadas, de igual forma que una ruta puede estar asignada a más de un usuario. Esta relación N – N está basada en diferentes fechas de asignación, naciendo de esta misma una nueva tabla en nuestro modelo de datos que denominaremos *itinerario*. Es decir, un itinerario es la correspondencia de una ruta a un usuario en alguna fecha señalada y con información adicional sobre incidencias u observaciones y consumación de dicho itinerario.

- Un lugar podrá constar como parada en más de una ruta, de la misma manera que una ruta, evidentemente, podrá tener más de una parada en diferentes lugares. Esta relación N – N resulta en una nueva tabla para nuestro modelo de datos que llamaremos *parada*. Sin embargo, esta tabla no requiere añadir ninguna información adicional, siendo únicamente el producto de dicha relación y quedando tal y como veremos más en detalle en el diagrama del modelo de datos.

Habiendo matizado las relaciones existentes en el SI que nos proponemos diseñar, podemos ahora pasar a representar el diagrama del modelo de datos relacional definitivo:

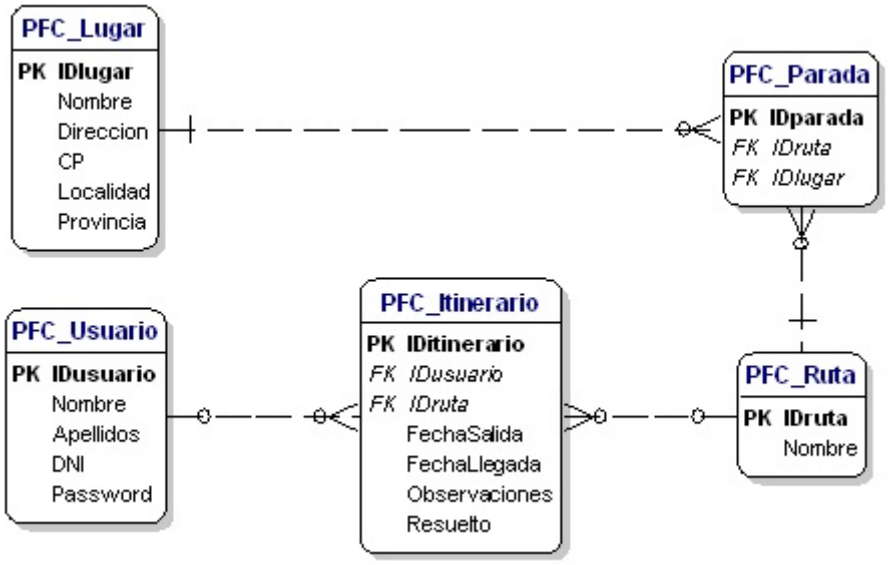


Figura 20 – Diseño de la Base de Datos del sistema (pfcjmag_bd)

Este es el resultado del diseño de la BD. Cabe mencionar como criterio de diseño que el orden de paradas para una ruta está determinado por el mismo orden ascendente seguido por el *IDparada* para esa ruta o, lo que es lo mismo, para una misma *IDruta*. Es decir, según nuestro diseño, podemos tener valores *IDparada* para una ruta que no sigan secuencia alguna, pero siempre se irá recorriendo la ruta desde el lugar que le corresponda menor *IDparada* hasta el mayor.

Otra anotación interesante que nos sugiere dicho diseño es que un mismo lugar podrá constar como parada en más de una ocasión para una misma ruta.

En la siguiente sección, que describe el sistema a bajo nivel, trataremos en mayor detalle cada uno de los atributos correspondientes a las diferentes tablas.

3.2 Entorno Web

El desarrollo de esta parte, como ya se ha visto en los capítulos anteriores, se ha realizado en su totalidad a través de herramientas facilitadas por *Sun Microsystems*, o lo que viene a ser *Oracle*. Se trata de la plataforma J2EE, que consta de un servidor de aplicaciones web como es GlassFish.

El diseño de dicho proyecto web se ha basado en las siguientes especificaciones:

- Tomar la tecnología en auge en dicho ambiente, es decir, servlets y JSPs.
- Como lógica de negocio primordial y exclusiva tendremos operaciones con Base de Datos. Señalar que se ha optado por el empleo de un conjunto de conexiones (ver *2.2.7 Conjunto de Conexiones*) que se debe configurar en el servidor GlassFish (proceso detallado en *A.2.1.5 Configuración del conjunto de conexiones…*).
- Tratándose de un SI corriente, la aplicación web constará básicamente de listados presentados en forma de tabla que muestren la información vigente y de funciones y formularios para alterar dicha información.

Con estas tres premisas, se han formulado ciertos criterios de diseño como son:

- Dado que, como se ha indicado, todos los servlets necesitarán de operaciones sobre la BD, ya sea consulta, inserción, actualización o borrado, hemos visto conveniente en el diseño que los servlets incluyan el elemento necesario para dicha funcionalidad, es decir, que adopten la metodología necesaria para manejar conexiones a la BD. De este modo, elaboramos una clase que herede de *javax.servlet.http.HttpServlet*, facilitada por el API, que añada funciones de gestión sobre la conexión a BD, es decir, asignación y liberación de conexiones para solicitudes por parte del cliente – navegador. En estas funciones no ha sido necesaria la implementación de sincronización puesto que el propio conjunto de conexiones ya controla la concurrencia sobre las mismas[11]. Esta será la clase que tomemos como padre para todos los servlets visibles en el navegador web.
- Elaboración de una clase *Formulario* en cada caso, para su uso por parte de los diferentes servlets que lo requieran. Este criterio de diseño se adapta a la filosofía presentada en el apartado *2.2.3 JavaBeans*.
- Implementación de una clase que aporta funcionalidad de acceso global (*static*) para la impresión en formato HTML en todos los servlets y JSPs que supongan la visualización de cualquier página web en el navegador, compartiendo la misma apariencia web en toda la aplicación.

Estos son los puntos en los que se ha basado la estructura del proyecto web, existiendo variaciones en algunos apartados en la medida que ha sido necesario para conseguir ciertas particularidades en la funcionalidad, y que se verán en el capítulo descriptivo a bajo nivel.

[11] Sin embargo, como ya adelantamos, la concurrencia sobre los datos a nivel de interfaz de usuario no está controlada, es decir, un usuario puede pretender modificar ciertos datos sin ser consciente de que, mientras él se encuentra en el formulario de modificación, otro usuario ha podido modificarlos desde otra sesión web o incluso desde la aplicación móvil de usuario con la *Entrega de Itinerarios* (descrita a continuación).

3.3 Entorno Móvil

En esta parte se han desarrollado hasta tres aplicaciones para ser ejecutadas en tres dispositivos diferentes, dos de ellos móviles (mediante J2ME, sobre *CLDC 1.1* y *MIDP 2.0*) y el otro el servidor donde se aloja la BD (mediante J2SE).

Entre estas tres aplicaciones se han implementado varias modalidades de comunicaciones. Como se puede apreciar en la *Figura 1 – Esquema de dimensionado del proyecto* y se ha mencionado inicialmente, se ha incluido para dotar de mayor complejidad al desarrollo un móvil denominado *intermediario*, cuya misión es simplemente actuar como pasarela para la información que viaja extremo a extremo de una comunicación sobre tecnología NFC a otra sobre tecnología bluetooth, y viceversa. Así, este móvil intermediario se comunicará con el otro móvil a un extremo mediante el protocolo NFCIP y con el servidor al otro extremo gracias a BlueCove.

Existen varias comunicaciones implementadas para diferentes fines. Principalmente, las dos comunicaciones más empleadas por los usuarios serán las denominadas *Recepción de Itinerarios* y *Entrega de Itinerarios*. La primera trata de la obtención por parte del usuario de sus itinerarios correspondientes, es decir, de las rutas que le han sido asignadas en su deber laboral aún pendientes de resolver y con fechas de salida pasadas o actuales. La segunda opción consiste en la actualización de la información registrada por el usuario a lo largo de su trabajo respecto a los anteriores itinerarios recibidos. Podríamos decir que, dentro de la finalidad del proyecto, la primera comunicación interviene en la distribución de la información y la segunda en la centralización o sincronización de la misma.

Cabe hacer alusión a lo delicado de mantener una comunicación NFC dado que, al ser de contacto, cualquier leve movimiento que separe a sus partícipes puede suponer la interrupción de la misma. Así, para las comunicaciones citadas en el párrafo anterior se contempla más de un caso (de los que daremos cuenta en la parte *4.3.4 Implementación de las comunicaciones inalámbricas*) en los que se indica al usuario, mediante oportunos mensajes a través de la pantalla de los dispositivos móviles, los momentos en los que debe tomar contacto o permanecer sin alejarse del móvil intermediario, todo en función del tiempo de ejecución marcado por el transcurso de la otra parte de la comunicación, bluetooth. O sea, con el establecimiento de la conexión bluetooth sujeto o no a la búsqueda del servidor bluetooth se determina el establecimiento y liberación de la conexión NFCIP.

Respecto al almacenamiento de los itinerarios, que deben permanecer en el dispositivo móvil del usuario mientras se suceden ambas comunicaciones, haremos hincapié en la siguiente sección de bajo nivel (*4.3.1 Móvil Usuario*).

Aunque aprovecharemos esta referencia, pues del almacenamiento en el dispositivo móvil también depende la implementación de un sistema *Login* o de Validación de Acceso que permite la identificación del usuario para precisar las opciones de la aplicación, de la que se supone su utilización en cualquier lugar. Además, se provee de seguridad a la aplicación, aparte de que con esta funcionalidad se permite la utilización de un mismo dispositivo móvil por parte de más de un usuario para sus fines; situación extremadamente inconsistente para los datos, por otro lado, si el usuario emplea sucesiva

e indistintamente más de un dispositivo y no procura centralizar regular y ordenadamente la información mediante la *Entrega de Itinerarios*.

Y, a propósito, no debemos olvidarnos de una última modalidad de comunicación que, por su similitud a la *Recepción de Itinerarios*, hemos relegado a un segundo plano. Se trata de la *Actualización de Login* y es necesaria para el funcionamiento del Sistema de Acceso de usuario, pues la información que este requiere se integra en el SI que alberga la BD y el modo de habilitarlo y actualizarlo es solicitándolo al servidor mediante las comunicaciones inalámbricas dispuestas en el entorno móvil.

Por último, recordemos que, en esta ocasión, el servidor bluetooth hace uso del conector JDBC para interactuar con la BD, sin tener que preocuparse tampoco por la concurrencia pues, dada la implementación en la que el intermediario atiende las solicitudes dirigidas al servidor, no es posible que un segundo usuario realice ninguna solicitud mientras el intermediario está ocupado atendiendo otra anterior. Con esto, el servidor bluetooth sólo precisará de una misma conexión a BD que, de hecho, mantendrá abierta durante su ejecución para sus servicios en la parte móvil, mientras que la conexión bluetooth se restablecerá a cada solicitud que reciba.

3.4 Casos de Uso

En este apartado vamos a representar un caso de uso completo del sistema mostrando su utilidad mediante un ejemplo de implantación en una supuesta empresa, pongamos que, encargada de máquinas expendedoras de productos consumibles, como por ejemplo, chicles, comestibles dulces y salados,…

Presumamos que la empresa opera únicamente en la Comunidad Autónoma de Madrid y que dispone de máquinas en los siguientes emplazamientos (de norte a sur):

Figura 21 – Lugares hipotéticos del Caso de Uso

- Centro de Salud Rascafria – Calle de Artiñuelo, S/N, 28740 Rascafria.
- Centro Comercial Puerta de Miraflores – Calle del Río, S/N, 28792 Miraflores De La Sierra.
- Estación de Valdesqui (Puerto de Navacerrada) – Carretera Cotos-rascafria, KM.2, 28470 Cercedilla.
- Centro de Educación Ambiental Valle Fuenfria – Carretera de las Dehesas, KM 2, 28470 Cercedilla.
- Centro Ecuestre La Rocina – Carretera Guadalix, KM 2, 28770 Colmenar Viejo.
- Montessori School – Avenida de Mataespesa, 45, 28430 Alpedrete.
- Real Club de Golf La Herreria – Ctra. de Robledo de Chabela S/N, 28200 San Lorenzo de El Escorial.
- Centro Comercial La Rotonda – Plaza del Toro, 1, 28760 Tres Cantos.
- Hospital Infanta Sofia – Paseo Europa (Antigua Ctra. Burgos), 34, 28703 San Sebastián De Los Reyes.
- Centro Comercial La Vega – Avenida Olímpica, 9, 28108 Alcobendas.
- Karting Carlos Sáinz – Europólis, 28232 Las Rozas de Madrid.
- Museo Casa Natal de Cervantes – Calle Mayor, 48, 28801 Alcalá de Henares.
- Palacio De Velazquez – Parque Del Retiro, S/N, 28009 Madrid.
- Centro Comercial Zocoslada – Avenida de España, 23, 28821 Coslada.
- Estación de Metro Puerta del sur – Avda Alcalde Joaquín Vislumbrales esquina con Avda. de Leganés, Alcorcón.
- Plaza de Toros de Leganes – Calle del Maestro, 4, 28911 Leganés.
- Universidad Carlos III de Madrid Facultad de Ciencias Sociales y Jurídicas Facultad de Humanidades, Comunicación y Documentación – C/ Madrid, 126 - 28903 Getafe.
- Universidad Rey Juan Carlos – Calle del Tulipán, 28933 Móstoles.
- Pabellón Polideportivo Fernando Martín – C/ Grecia, 2, Fuenlabrada.
- Centro De Salud Mental Navalcarnero – Calle de la Doctora, S/N, 28600 Navalcarnero.
- Parador De Chinchon – C/ Huertos, 1, 28370 Chinchon.
- Palacio Real de Aranjuez – Avenida de Palacio, 28300 Aranjuez.

Llegados a este punto, vamos a distinguir dos papeles diferentes desarrollados por el personal de la empresa, uno será el de reponedor, cuya misión será abastecer las existencias de las máquinas expendedoras; y el otro será el superior del anterior, es decir, el que dirija los desplazamientos de los reponedores en el día a día, que puede ser llevado a cabo por una única persona, director, o por un conjunto de personas coordinadas entre sí, dirección. Es oportuno reseñar aquí que los reponedores serán los denominados *usuarios* del sistema, sirviéndose de la aplicación móvil correspondiente, y la dirección será la que emplee la aplicación web, de modo que se encargue de configurar los desplazamientos de los reponedores o *usuarios*.

En una vista a mejora (*6.2 Trabajos futuros*), podría dirigirse la aplicación web también a usuarios reponedores en modo lectura (excepto en los campos destinados a su cumplimentación: observaciones y estado de resolución), pudiendose añadir también las funcionalidades de recepción y entrega de itinerarios desde el móvil a través de la web, es decir, con móvil usuario, ordenador personal y servidor web de extremo a extremo; a diferencia de cómo se han desarrollado actualmente estas funcionalidades, móvil usuario, móvil intermediario y servidor bluetooth de extremo a extremo, donde se precisan todas

las partes en un mismo lugar dada la comunicación de alcance limitado entre ellos y la inexistencia del componente web. Esta primera mejora no sería completa sin un sistema Login o de Acceso a usuario también para la aplicación web, que permita identificar al usuario que está haciendo uso de la misma y, en función de ello, establezca sus opciones.

3.4.1 Entorno Web

Una vez aclaradas las responsabilidades del personal de la empresa, lo primero por parte de la dirección, además de registrar al personal encargado de la reposición de estas máquinas en nuestro sistema como *usuarios* mediante el acceso web, sería registrar también como *lugares* la localización de cada una de estas máquinas, o grupos de máquinas, anteriormente listadas. Ambas acciones son fácilmente realizables mediante la cumplimentación del formulario correspondiente a través de las opciones *Nuevo Usuario* y *Nuevo Lugar*, accesibles cada una desde sus propios apartados, *Lugares* y *Usuarios*, donde se muestra una lista de los lugares y usuarios existentes, respectivamente, y las opciones para modificarlos o eliminarlos. Dichos formularios tienen el siguiente aspecto:

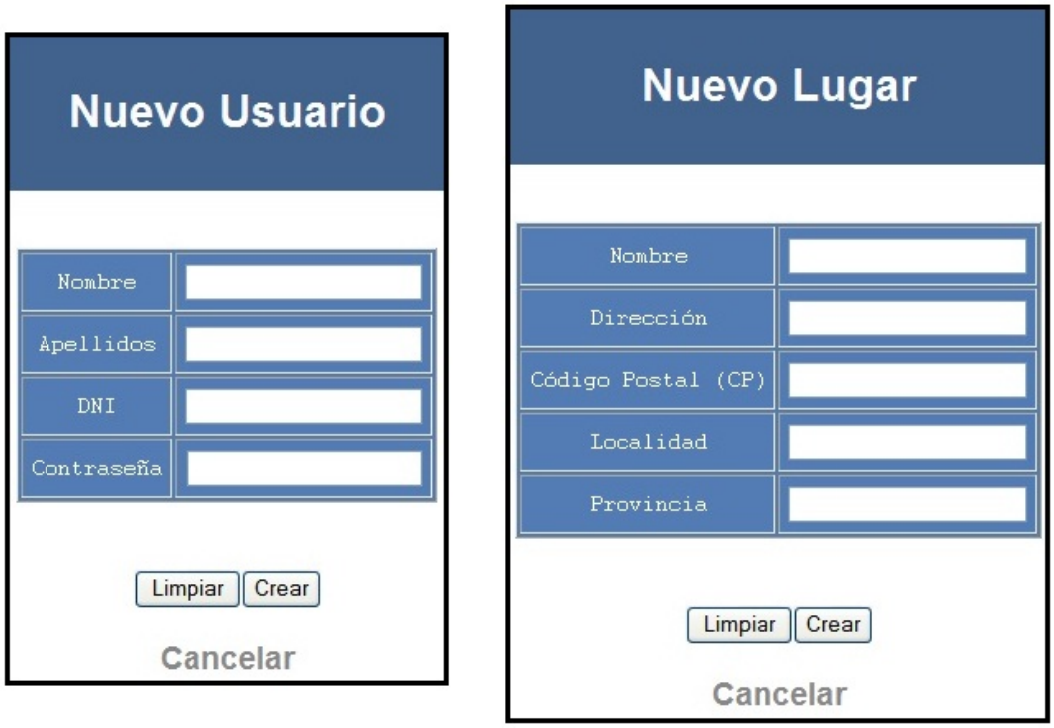
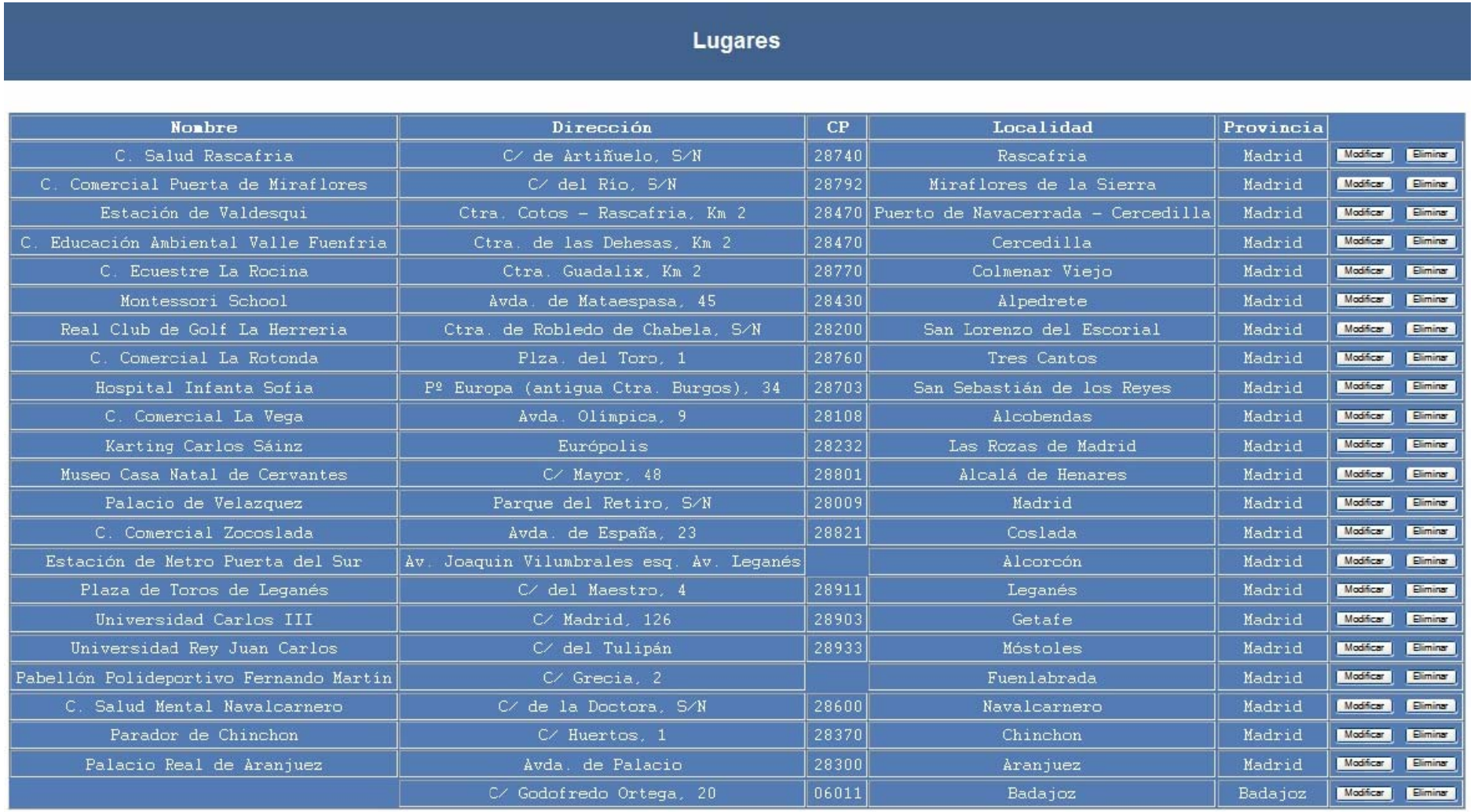


Figura 22 - Formularios para *Nuevo Usuario* y *Nuevo Lugar*, respectivamente

A continuación, mostramos el resultado de los usuarios creados en el sistema. La creación de los lugares tiene un aspecto similar[12], aunque dado su mayor tamaño lo ilustramos en la siguiente página apaisada sin incluir los botones inferiores de *Volver* y *Nuevo Lugar*.

[12] Obsérvese que se ha añadido un lugar más a los listados anteriormente para este caso de uso, en la localidad de Badajoz y sin nombre, con la intención de probar posteriormente en el formulario de ruta el despliegue del campo *parada* con lugares tanto con nombre como sin él (gracias a lo expuesto en el apartado *B.2 Vista alternativa de campos Nombre*).

Figura 23 - Lista de Usuarios



Lugares

Nombre	Dirección	CP	Localidad	Provincia	
C. Salud Rascafria	C/ de Artiñuelo, S/N	28740	Rascafria	Madrid	Modificar Eliminar
C. Comercial Puerta de Miraflores	C/ del Río, S/N	28792	Miraflores de la Sierra	Madrid	Modificar Eliminar
Estación de Valdesqui	Ctra. Cotos - Rascafria, Km 2	28470	Puerto de Navacerrada - Cercedilla	Madrid	Modificar Eliminar
C. Educación Ambiental Valle Fuenfria	Ctra. de las Dehesas, Km 2	28470	Cercedilla	Madrid	Modificar Eliminar
C. Ecuestre La Rocina	Ctra. Guadalix, Km 2	28770	Colmenar Viejo	Madrid	Modificar Eliminar
Montessori School	Avda. de Mataespasa, 45	28430	Alpedrete	Madrid	Modificar Eliminar
Real Club de Golf La Herreria	Ctra. de Robledo de Chabela, S/N	28200	San Lorenzo del Escorial	Madrid	Modificar Eliminar
C. Comercial La Rotonda	Plza. del Toro, 1	28760	Tres Cantos	Madrid	Modificar Eliminar
Hospital Infanta Sofia	Pº Europa (antigua Ctra. Burgos), 34	28703	San Sebastián de los Reyes	Madrid	Modificar Eliminar
C. Comercial La Vega	Avda. Olímpica, 9	28108	Alcobendas	Madrid	Modificar Eliminar
Karting Carlos Sáinz	Európolis	28232	Las Rozas de Madrid	Madrid	Modificar Eliminar
Museo Casa Natal de Cervantes	C/ Mayor, 48	28801	Alcalá de Henares	Madrid	Modificar Eliminar
Palacio de Velazquez	Parque del Retiro, S/N	28009	Madrid	Madrid	Modificar Eliminar
C. Comercial Zocoslada	Avda. de España, 23	28821	Coslada	Madrid	Modificar Eliminar
Estación de Metro Puerta del Sur	Av. Joaquin Vilumbrales esq. Av. Leganés		Alcorcón	Madrid	Modificar Eliminar
Plaza de Toros de Leganés	C/ del Maestro, 4	28911	Leganés	Madrid	Modificar Eliminar
Universidad Carlos III	C/ Madrid, 126	28903	Getafe	Madrid	Modificar Eliminar
Universidad Rey Juan Carlos	C/ del Tulipán	28933	Móstoles	Madrid	Modificar Eliminar
Pabellón Polideportivo Fernando Martín	C/ Grecia, 2		Fuenlabrada	Madrid	Modificar Eliminar
C. Salud Mental Navalcarnero	C/ de la Doctora, S/N	28600	Navalcarnero	Madrid	Modificar Eliminar
Parador de Chinchon	C/ Huertos, 1	28370	Chinchon	Madrid	Modificar Eliminar
Palacio Real de Aranjuez	Avda. de Palacio	28300	Aranjuez	Madrid	Modificar Eliminar
	C/ Godofredo Ortega, 20	06011	Badajoz	Badajoz	Modificar Eliminar

Figura 24 – Lista de Lugares

A partir de este momento, se deben crear las rutas que recorren los lugares registrados. Esta parte de diseño de las rutas depende de los criterios que se crean oportunos dar mayor o menor relevancia, en función de los fines de la empresa, y se debe hacer un ejercicio de análisis exhaustivo, teniendo en cuenta siempre la opción de mejora con el paso del tiempo y la especialización. Esta debe ser función, normalmente, de la directiva y entre esos criterios puede tomar generalmente los siguientes:

1. Diseñar rutas que recorran los lugares con cierto orden, intentando rentabilizar las distancias recorridas para disminuir el gasto en desplazamientos.
2. Junto con lo anterior, también es importante tener en cuenta las vías utilizables en las rutas. Esto es, quizás dos lugares estén cerca en distancia pero no tengan una conexión suficientemente buena, e interese incluirlos en diferentes rutas que empleen otras vías mientras no se penalice excesivamente de este modo en otros criterios. Este punto es de vital importancia para casos de transporte pesado, con vehículos de alto tonelaje.
3. Tener en cuenta la similitud de los lugares que forman parte de cada ruta. Por ejemplo, si una empresa tiene destinos comerciales y de mantenimiento hacer clara distinción, pues incluso serán visitados por diferente personal (esta situación puede recalar en una mejora con nuevos campos del estilo a *tipos de ruta* o *lugar* y de *usuario*, a tener en cuenta para la asignación en un itinerario). Desde este ejemplo tan básico hasta similitudes de cualquier tipo que se puedan reconocer, como la semejanza de los clientes de cara a los productos solicitados en cada destino que, con una clara especificación de las necesidades del caso, puede dar lugar a otra remodelación de la BD añadiendo nuevos atributos al lugar como los productos requeridos, tratándose cada producto como un nuevo atributo *booleano* y representado en los formularios mediante cajas de confirmación (*checkbox*, idénticos al del campo actual *resuelto*); un ejemplo de estos campos en el actual caso de uso sería: refrescos, chicles, dulces, patatas, etc. (ver *6.2 Trabajos futuros*).
4. Tratar de normalizar la necesidad del servicio prestado en los destinos de una misma ruta. Dicho de otro modo, intentar que los destinos de una misma ruta requieran la misma periodicidad de atención. Así, las rutas tenderán a ser estables y se requerirá menor creación de rutas "eventuales". Si se logran normalizar en este sentido las rutas, el funcionamiento de la empresa también quedará normalizado y, por tanto, simplificado. En consecuencia, se puede considerar este punto crucial para el análisis de las rutas.
5. Por último, a la hora de diseñar las rutas, es de vital importancia considerar el tiempo estimado para la atención de cada una en vista a la posterior asignación de las mismas en un itinerario donde se deberá indicar una fecha de salida y de llegada (este parámetro se podría añadir también al modelo de datos, *6.2 Trabajos futuros*). Con esto, también se debe tener en cuenta el lugar de inicio y el final, o más bien, la estima de tiempo desde la sede de la empresa hasta el comienzo de la ruta y de retorno del responsable, de nuevo a la sede, desde la última parada. Incluso, podría resultar adecuado incluir como primera y última parada de todas las rutas el lugar donde se ubique la sede de la empresa.

Para el caso que nos ocupa, vamos a mostrar nuestro diseño de rutas sobre el mismo plano que representaba los destinos que alojaban máquinas expendedoras de la empresa en cuestión:

Figura 25 – Rutas diseñadas para el Caso de Uso

Este ha sido el resultado obtenido a raíz de la siguiente interpretación de los anteriores criterios:

1. El orden de los destinos ha sido prácticamente el único criterio de relevancia tomado para este ejemplo.
2. A las condiciones de las vías o carreteras no se les ha dado relevancia, pues tratándose de municipios madrileños importantes no debería, en principio, haber problemas en este sentido. Sin embargo, es importante resaltar que existe una ruta que puede resultar de mayor peligro, en suma con las condiciones climatológicas, pues atraviesa un puerto de carretera. En este sentido, surge una sencilla mejora como sería la inclusión de un nuevo atributo en el modelo de datos para las rutas que podría llamarse *dificultad* o *peligrosidad*, asignable en cierto rango (bajo, medio y alto, por ejemplo; ver en *6.2 Trabajos futuros*).
3. Dado este ejemplo la única similitud razonable que puede surgir, como en la mayoría de empresas comerciales, es la variedad o categoría de los productos, suponiendo que existan clientes al que se le prestan unos u otros productos, de mayor o menor calidad.
4. Suponemos que, efectivamente, el diseño obtenido no pesa sobre este punto tan importante. Es más, nuestra suposición se basa en que todos los destinos precisan de una periodicidad de reposición similar.
5. En este último punto hemos considerado oportuno, dada las características de la empresa, que cada ruta se pueda desempeñar en una única jornada laboral, es decir, en el mismo día. Por ello, podremos prescindir de indicar una fecha de llegada en cada itinerario que generemos, pues supondremos que es la misma que la fecha de salida. Atendiendo a esta condición, podemos observar que las rutas tienen una media de cuatro destinos; teniendo una con el máximo de seis paradas, sin ser su recorrido muy superior al de las demás y con su origen y fin muy cercanos al centro, Madrid capital, donde imaginamos que se encuentra la sede de

la empresa con sus efectivos; y otra con el mínimo de tres paradas, pero de recorrido equiparable y puntos de partida y llegada más alejados de Madrid que la anterior.

Tras toda esta exposición, y con nuestro diseño de rutas finalizado, sólo faltaría dar registro a las rutas resultantes, de nuevo, por parte de la dirección y a través del formulario proporcionado por la opción *Nueva Ruta*, localizada en el apartado de *Rutas*, donde se muestra un listado de las rutas existentes y las opciones para modificarlas o eliminarlas. Dicho formulario tiene el siguiente aspecto (en la *Modificación* de una de las rutas ya creadas):

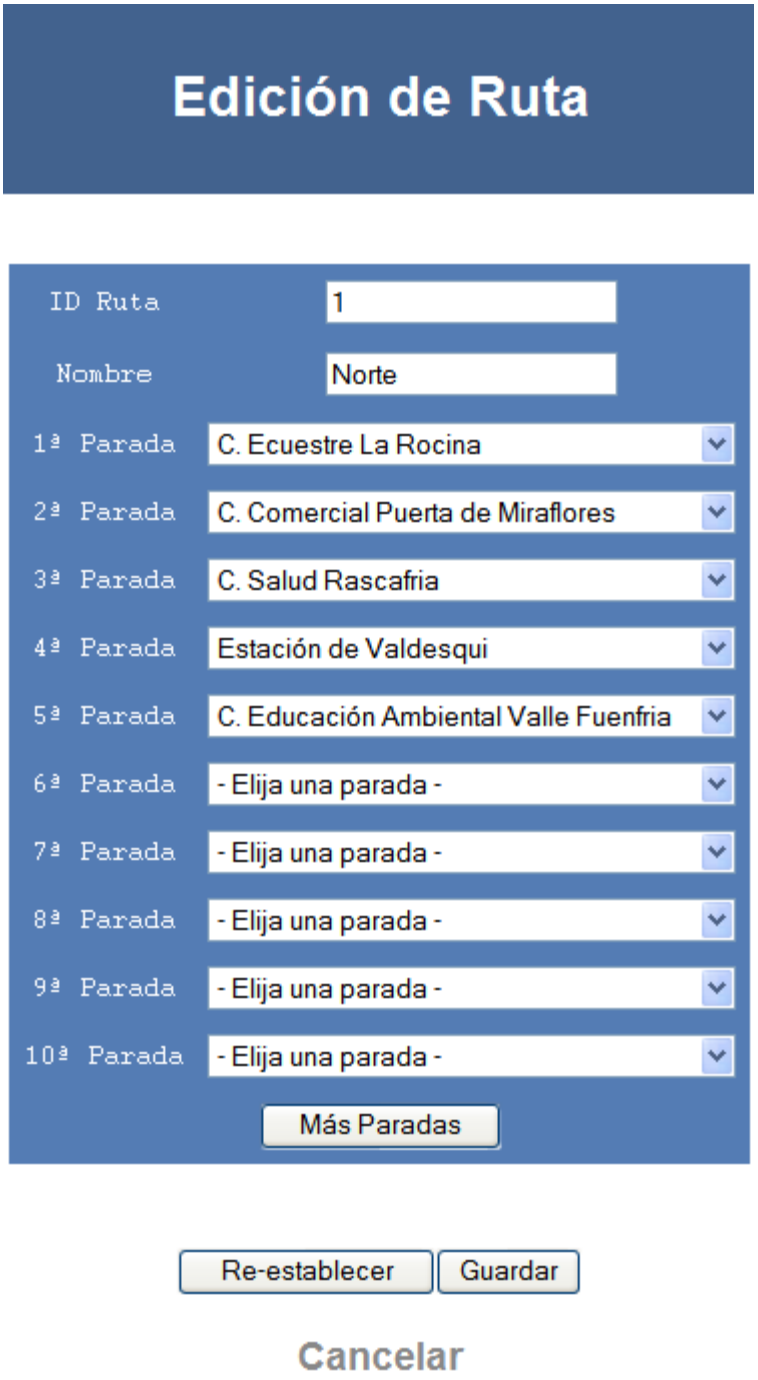


Figura 26 – Formulario para la Edición de Ruta

Como se puede observar, es posible dar un nombre a la ruta (sin ser obligatorio) y el resto simplemente trata de ir indicando el orden de paradas entre los lugares existentes, pudiendo incluir paradas de forma indefinida gracias al botón *Más Paradas* (para saber más sobre esta función ver *4.2.1 Casos excepcionales en el diseño*). Esta y el resto de rutas creadas se pueden apreciar en la siguiente imagen perteneciente al listado de rutas que figura en su correspondiente apartado de la aplicación web:

Figura 27 – Lista de Rutas

Y bien, el sistema ya consta de la información más estable. Ya sólo faltaría la información de *itinerarios* que se genera rutinariamente con la asignación de las *rutas* a los *usuarios* para su realización. Se puede tender a la normalización de dicha información, pero no cabe duda de que, junto con las *rutas* en menor medida, será la más dinámica. Al contrario, los *lugares* y *usuarios* se irán añadiendo con el establecimiento de máquinas en nuevos emplazamientos y la incorporación de nuevo personal; o se irán modificando, si una máquina sufre un leve desplazamiento (por cambio de edificio del cliente, por ejemplo) o la dirección postal donde se encuentra varía y para la actualización de la información de usuario (por ejemplo, un cambio de contraseña); o eliminando, si se retira una máquina de donde estaba o se prescinde de los servicios de algún usuario y se le excluye del sistema.

Llegados hasta aquí, no podemos posponer el pronunciarnos respecto a las restricciones en el borrado de información del sistema impuestas por el modelo de datos. En lo que concierne a la exclusión de un *usuario* del sistema se ha entendido elemental evitar todo impedimento, de modo que no existe ningún problema en hacerlo en cualquier momento, sólo que los *itinerarios* que éste tuviera asignados quedarán modificados sin asignar a nadie. No ocurre igual con los *lugares* y las *rutas*, de tal forma que si se desea eliminar un lugar, lógicamente, primero se deberá excluir de las rutas donde figuraba mediante la modificación o eliminación de estas. No obstante, en el supuesto de eliminación de una ruta también deberemos vigilar que no figure en ningún itinerario, lo cual se puede considerar el mayor inconveniente (detallado en *4.1 Modelo de Datos*).

3.4.1.1 Generación y manejo de *Itinerarios*

En principio, vamos a suponer que cada máquina requiere una visita de reposición en torno a cada tres días. Quizás este supuesto no se acerque a la realidad, pero aquí se

pretende nada más que el planteamiento y la esquematización de un caso práctico. A raíz de este ejercicio inicial, se puede variar la rutina que marquemos como guste y convenga pues, en realidad, esa es una de las ventajas que dispone este sistema, la facilidad de hacer llegar a los encargados cierta flexibilidad en las tareas laborales del transporte.

Abordando este esquema, lo primero que nos encontramos es que durante la semana existen dos días, sábado y domingo, no laborables. Con esto, lo más sensato que podríamos pensar, teniendo en cuenta la premisa de atender cada lugar o cada ruta en intervalos de tres días, es repartir todas las rutas en dos sesiones de reparto a la semana: una que sea los lunes y jueves y, la otra, los martes y viernes; solventando así de la mejor manera posible el período de fin de semana.

Por otro lado, disponemos de hasta cuatro personas encargadas de la reposición y de cinco rutas, tal y como se ha visto. De este modo, habrá para una sesión tres rutas y tres encargados ocupados, y para la otra las dos rutas restantes y dos encargados ocupados. Con este planteamiento, vamos a equiparar el trabajo entre los cuatro usuarios del sistema de la forma más equilibrada posible:

- Si en la primera sesión, los lunes y jueves, libra de prestar su servicio un usuario, cada semana del mes será uno diferente el que libre esta sesión; o mejor, de los dos días de la sesión en cada semana, librará uno y otro, por parejas como se marca en el siguiente punto.
- Si la segunda sesión, los martes y viernes, se salda con dos usuarios, haremos turnos por parejas para que cada viernes (por aquello de ser último día laboral de la semana) lo atienda una pareja de usuarios diferente, por ejemplo, la que libra esa semana en la primera sesión. Librando así cada pareja un martes, una semana, y un viernes, la siguiente.

Dando hasta aquí por zanjado el esquema inicial de itinerarios para la hipotética empresa, vamos a mostrar el modo de introducir dicha información. Bien es cierto que, en total, se trata de hasta diez itinerarios por semana que deberán ser generados por la dirección en el sistema. Aquí surge otra posible mejora de generación automática, la cual preste alguna opción de creación múltiple de itinerarios predefinidos en un esquema para un período de tiempo dado, tal y como el que hemos ideado (su implementación sería mediante un script SQL, según se indica en *6.2 Trabajos futuros*).

Mostramos a continuación el formulario a completar para la generación manual de un itinerario, ya con la información de alguno de los planteados. Dicho formulario es accesible mediante la opción *Nuevo Itinerario*, localizada en el apartado de *Itinerarios* donde se muestra un listado de los itinerarios existentes y las opciones para modificarlos o eliminarlos.

Figura 28 – Formulario para Nuevo Itinerario

Podemos observar en el formulario de itinerario la aparición previa del listado de rutas existentes, de este modo resultará mucho más sencillo para el usuario de la aplicación web hacer la asignación de rutas a los encargados o, lo que es lo mismo, el manejo de itinerarios. Además de los campos propios de la asignación, ruta y usuario, existen otros relativos a fechas que ubican en el calendario cuándo se ha de comenzar y cuándo se debe finalizar el recorrido de la ruta. Los otros dos campos restantes están destinados a los encargados del itinerario, para que los cumplimenten marcando si consideran el itinerario resuelto y, por tanto, que no requiere volver a atenderse (campo *Resuelto*) y con incidencias o cualquier tipo de información suscitada a raíz de la realización del itinerario (campo *Observaciones*); todo mediante la aplicación móvil, por lo que lo veremos con mayor detalle en su apartado próximo. Fundamentalmente, ambos campos tienen para la dirección el sentido de supervisión del trabajo de los reponedores a través también de la web, aunque también puede ser útil el campo de *observaciones* para que la dirección, previamente a la recepción y ejecución del itinerario por parte de su responsable, pueda darle alguna indicación o similar.

A raíz de dicha supervisión, la dirección puede ver oportuno el modificar o generar itinerarios futuros, con lo que estos dos últimos campos pueden ser un factor clave que facilite el dinamismo requerido por la empresa.

En la siguiente figura, se muestra el resultado de la asignación de rutas inicial propuesta para el período de las dos primeras semanas de noviembre (01/11/2010 – 12/11/2010). Además, en el análisis que trata de distribuir la asignación de las rutas entre los reponedores es importante también tener otros factores en cuenta que consideren la afinidad de estos mismos por unas u otras rutas, como podría ser el conocimiento previo de la zona o de las vías o la costumbre de asignación. Como se puede observar respecto a esto último, en nuestras asignaciones hemos tratado de acostumbrar a cada usuario a ciertas rutas dadas, siempre dentro de lo posible puesto que es inevitable que existan

algunos usuarios más polivalentes que otros. Nótese también que todos los usuarios trabajan cinco días, teniendo los días libres lo más salteado posible.

Itinerarios

	Asignado al usuario							
Ruta	Nombre	Apellidos	DNI	Fecha de Salida	Fecha de Llegada	Observaciones	Resuelto	
Centro - Este	Juan Manuel	Ardoy de Gracia	80073019Z	01/11/2010			No	Modificar Eliminar
Centro - Este	Juan Manuel	Ardoy de Gracia	80073019Z	04/11/2010			No	Modificar Eliminar
Centro - Este	Juan Manuel	Ardoy de Gracia	80073019Z	08/11/2010			No	Modificar Eliminar
Centro - Este	Patricia	Garcia Benitez	012345678B	11/11/2010			No	Modificar Eliminar
Norte	Patricia	Garcia Benitez	012345678B	02/11/2010			No	Modificar Eliminar
Norte	Juan Manuel	Ardoy de Gracia	80073019Z	05/11/2010			No	Modificar Eliminar
Norte	Juan Manuel	Ardoy de Gracia	80073019Z	09/11/2010			No	Modificar Eliminar
Norte	Pedro	del Monte Barrido	87654321	12/11/2010			No	Modificar Eliminar
Oeste	Pedro	del Monte Barrido	87654321	02/11/2010			No	Modificar Eliminar
Oeste	María	Jimenez de Peralta	12345678A	05/11/2010			No	Modificar Eliminar
Oeste	María	Jimenez de Peralta	12345678A	09/11/2010			No	Modificar Eliminar
Oeste	Patricia	Garcia Benitez	012345678B	12/11/2010			No	Modificar Eliminar
Sur	Pedro	del Monte Barrido	87654321	01/11/2010			No	Modificar Eliminar
Sur	Patricia	Garcia Benitez	012345678B	04/11/2010			No	Modificar Eliminar
Sur	Pedro	del Monte Barrido	87654321	08/11/2010			No	Modificar Eliminar
Sur	Pedro	del Monte Barrido	87654321	11/11/2010			No	Modificar Eliminar
Sur - Este	María	Jimenez de Peralta	12345678A	01/11/2010			No	Modificar Eliminar
Sur - Este	María	Jimenez de Peralta	12345678A	04/11/2010			No	Modificar Eliminar
Sur - Este	Patricia	Garcia Benitez	012345678B	08/11/2010			No	Modificar Eliminar
Sur - Este	María	Jimenez de Peralta	12345678A	11/11/2010			No	Modificar Eliminar

Figura 29 – Lista de Itinerarios

Los itinerarios que puedan surgir de forma más esporádica podrán ser perfectamente dados de alta para que sean atendidos, a lo sumo, al día siguiente, según el funcionamiento actual del sistema. Esta limitación viene impuesta por el diseño y el caso de uso genérico del sistema, ya que la recepción de itinerarios se realiza mediante comunicación inalámbrica de corto alcance; se terminará de comprender perfectamente al completar la parte móvil del caso de uso, donde los encargados manejan y se responsabilizan de sus itinerarios. Ejemplos de itinerarios eventuales pueden ser:

- Itinerarios de reposición excepcionales por eventos que agoten las existencias de forma extraordinaria a las previsiones.
- Itinerarios nuevos, retrasados y, quizás, reasignados por no haber podido ser resueltos en su momento por cualquier circunstancia (encargado ausente por razones de salud, condiciones climatológicas que impidan el acceso a determinado lugar, etc.)
- Itinerarios extraordinarios de reparación o mantenimiento de las máquinas.
- Etc.

También puede ser relevante mencionar aquí que en la generación de un itinerario no es imprescindible indicar un usuario, simplemente es necesaria una ruta y una fecha de salida que sabemos se habrá de prestar servicio, pero aún sin determinar quién será el encargado.

Por último, ilustraremos en forma de diagrama UML lo que concierne al manejo de la aplicación web y su comportamiento, en base a todo lo explicado en esta parte web del caso de uso.

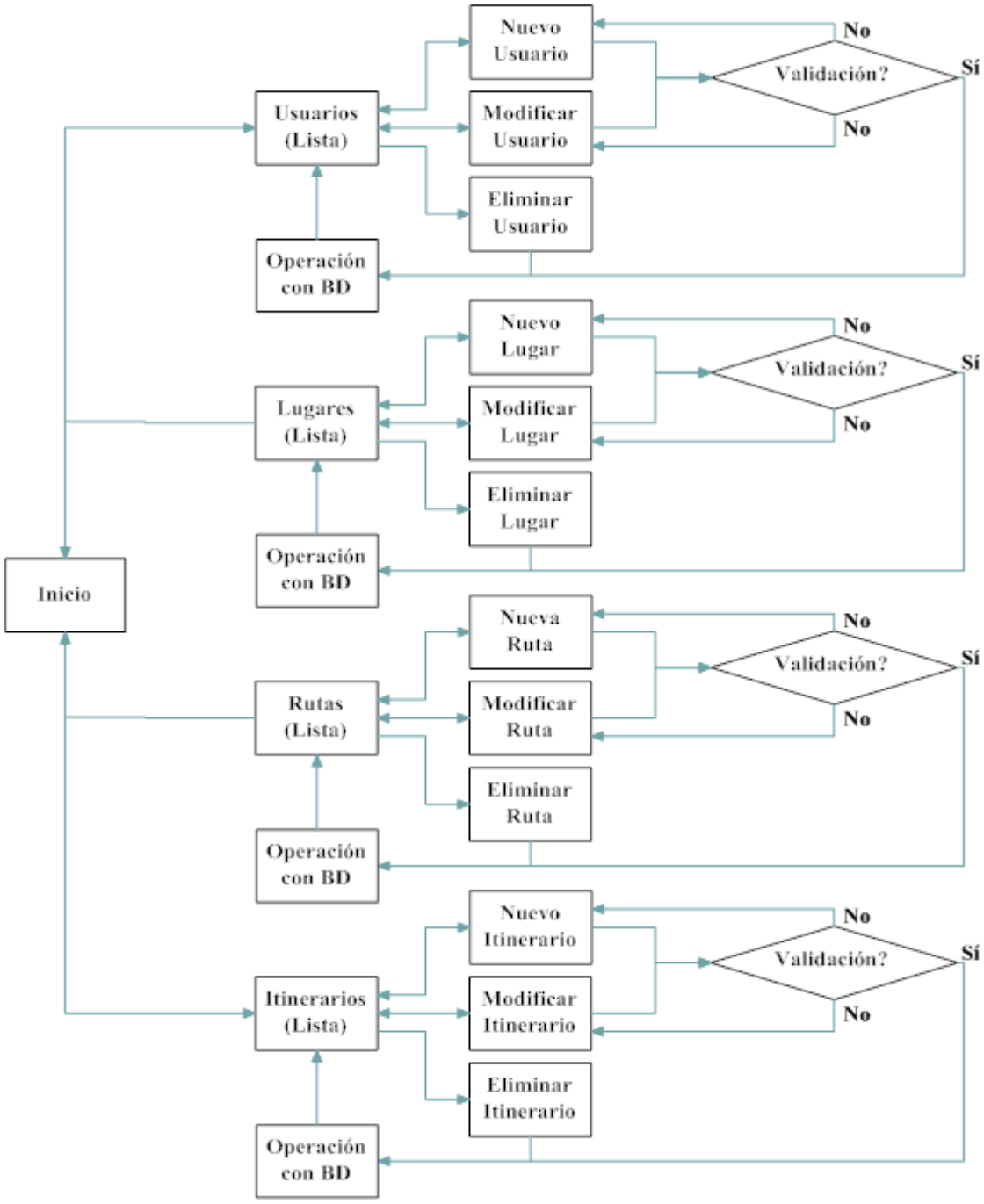


Figura 30 – Diagrama UML del caso de uso de la aplicación web

Para completar dicho diagrama, destacar que tanto las listas de cada apartado como la página de inicio de la web (*home*) son accesibles desde cualquier lugar de la aplicación gracias a una barra superior de enlaces. También podemos seguir nuestra navegación a través del mapa web o, lo que viene a ser, el diagrama mostrado gracias a la ruta web que figura bajo la citada barra y que puede servir de ayuda para volver a páginas anteriores.

3.4.2 Entorno Móvil

En esta parte entra en acción el personal encargado de la reposición de las máquinas expendedoras. Su cometido en torno al sistema es bastante simple. Su herramienta es su dispositivo móvil, el cual albergará una aplicación en la que se tendrá que validar como usuario para acceder (mayor detalle en *4.3.1 Móvil usuario*).

Tras el acceso a la aplicación móvil, existen tres opciones:

- ***Recibir Itinerarios***, que permite al usuario obtener los itinerarios que le han sido asignados con fecha de salida anterior o igual a la actual y, claro está, pendientes de resolver.
- ***Ver Itinerarios***, que permite visualizar la información de los itinerarios recibidos, fechas de salida y llegada y paradas de la ruta fundamentalmente, y modificar las observaciones y el campo resuelto.
- ***Entregar Itinerarios***, que da la posibilidad al usuario de devolver al SI los itinerarios que recibió con la información que ha considerado oportuno registrar, centralizando así toda la información y facilitando la supervisión de los superiores del trabajo que ha llevado a cabo.

En caso de que el usuario pretenda recibir sus itinerarios cuando, en realidad, el móvil ya alberga unos (que también le corresponden), se da la opción de eliminar dicha información almacenada en el móvil, con el riesgo de perder las modificaciones al no haber sido centralizada; o de cancelar, para tener opción de hacer la entrega de sus itinerarios vigentes en el móvil antes de volver a hacer la nueva recepción de itinerarios. De esta manera se contempla una posible confusión del usuario, permitiendo la cancelación, o un modo rápido de repetir una recepción de itinerarios por el motivo que sea, dando por hecho que, en verdad, la información patente en ese momento en el móvil no ha sufrido ninguna modificación.

Dadas las tecnologías de comunicación empleadas en la recepción y entrega de itinerarios, inalámbricas de corto alcance, el caso de uso en esta parte se reduce a lo siguiente:

- **Al comienzo de su jornada laboral**, el reponedor en su paso por la sede de la empresa procederá con la opción de *Recibir Itinerarios*, de modo que quedará determinado su trabajo durante dicha jornada.
- **A lo largo de la jornada**, el reponedor podrá consultar la información relativa a su trabajo, esto es, principalmente, las rutas que debe atender y las paradas de las que constan. A su vez, podrá ir registrando información relacionada con la realización de las mismas.
- **Al cabo de su jornada**, en su paso de nuevo por la sede de la empresa, el reponedor deberá ejecutar la opción de *Entregar de Itinerarios* para sincronizar la información registrada durante la jornada con el SI, quedando su móvil listo para la recepción de la próxima jornada.

A continuación, se muestra un diagrama UML que representa el caso de uso de la opción *Ver Itinerarios*, pues en las otras dos opciones de recepción y entrega de itinerarios el usuario no debe intervenir más que inspeccionando los mensajes

descriptivos de cada operación para aproximar su móvil al intermediario cuando se le indique (detallado en el apartado *4.3.4 Implementación de las comunicaciones inalámbricas*). Estas dos opciones de comunicación ya se verán integradas más adelante en el caso de uso genérico de la empresa ficticia, en conjunto con la parte web.

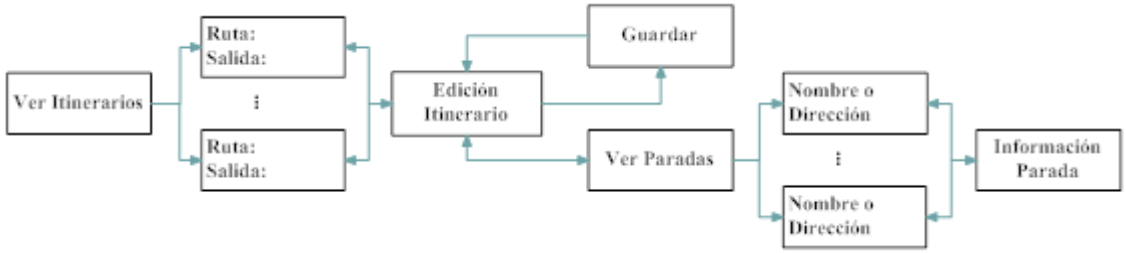


Figura 31 – Diagrama UML del caso de uso de la aplicación móvil del usuario

3.4.3 Caso de Uso completo, web y móvil

Para culminar este apartado de casos de uso, presentamos un último diagrama UML que intenta mostrar el caso de uso esencial y genérico del sistema, que integra ambos entornos y viene a ser la recapitulación de todo lo anterior.

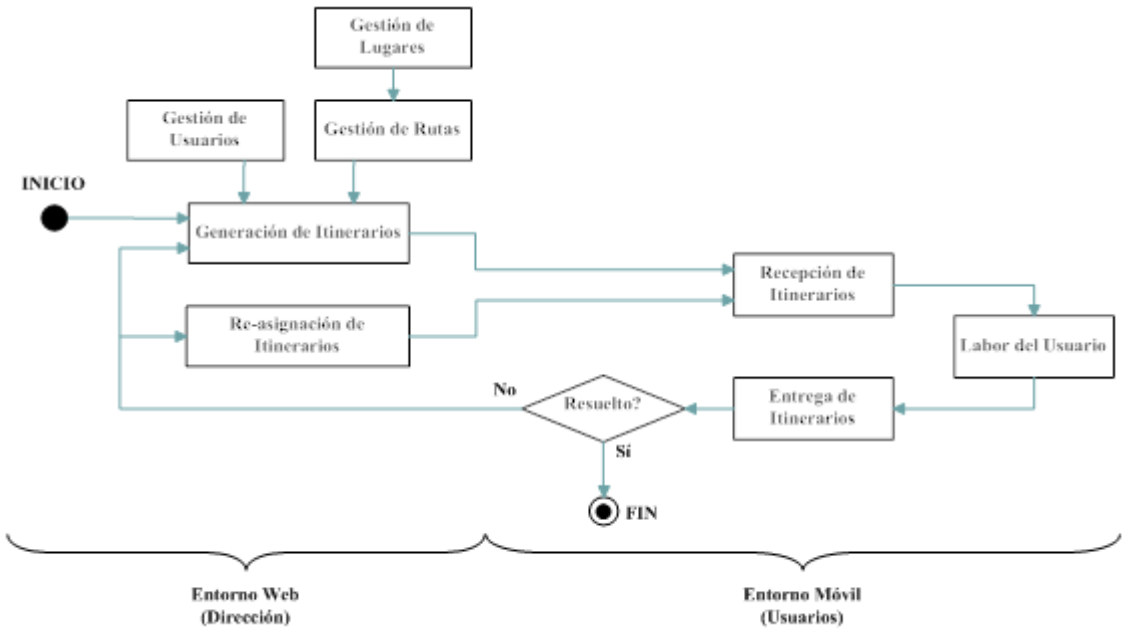


Figura 32 – Diagrama UML del caso de uso genérico del sistema

Es conveniente hacer un inciso sobre varios puntos del diagrama:

- La acción de *Labor del Usuario* se refiere a su empeño en la realización de los itinerarios que ha recibido, y debe ser de una duración aproximadamente igual a la del tiempo estimado para los mismos (en nuestro caso de uso se trata de una jornada laboral).

- Si tras la *Labor del Usuario* este determina que no ha quedado resuelto el itinerario, pueden existir más de una vía de actuación por parte de la dirección (en el diagrama se representa mediante la bifurcación de realimentación existente):
    - Hacer una reasignación de los itinerarios no resueltos, modificándolos en fecha o usuario asignado, o en ambos.
    - Eliminar el itinerario no resuelto o marcarlo como resuelto (porque pueda interesar conservar la información) para, seguidamente, generar otro itinerario que solvente al anterior.

Incluso si un usuario determina que un itinerario ha quedado resuelto, pero en la supervisión la dirección, por cualquier motivo, como puede ser la misma observación registrada por el usuario o un comunicado a posteriori por parte del cliente insatisfecho, considera que no ha sido así se puede actuar de forma análoga al último punto anterior, pero a la inversa: desmarcando el campo resuelto para hacer una nueva asignación o dejarlo como está y generar otro itinerario nuevo.

Capítulo 4

Descripción del Sistema: Bajo Nivel

4.1 Modelo de Datos

En este apartado nos corresponde mencionar los criterios adoptados a la hora de decidir el formato de todos los campos que conforman la Base de Datos. Señalar, a priori, que todas las claves primerias (*Primary Keys,* PKs) son campos *Integer* y *Autoincrement* y que, por definición, también serán *Not Null*.

A continuación, se presenta el diseño de todos los atributos que conforman el modelo de datos:

	Campo	Tipo	Longitud	PK/FK	NOT NULL
PFC_Lugar	IDlugar	INTEGER	-	PK (Autoincrement)	✓
	Nombre	VARCHAR	40		
	Dirección	VARCHAR	40		✓
	CP	VARCHAR	5		
	Localidad	VARCHAR	40		✓
	Provincia	VARCHAR	40		
PFC_Ruta	IDruta	INTEGER	-	PK (Autoincrement)	✓
	Nombre	VARCHAR	40		
PFC_Parada	IDparada	INTEGER	-	PK (Autoincrement)	✓
	IDruta	INTEGER	-	FK	✓
	IDlugar	INTEGER	-	FK	✓
PFC_Usuario	IDusuario	INTEGER	-	PK (Autoincrement)	✓
	Nombre	VARCHAR	40		
	Apellidos	VARCHAR	40		
	DNI	VARCHAR	10		✓
	Password	VARCHAR	40		✓
PFC_Itinerario	IDitinerario	INTEGER	-	PK (Autoincrement)	✓
	IDusuario	INTEGER	-	FK (On delete set Null)	
	IDruta	INTEGER	-	FK	✓
	FechaSalida	DATE	-		✓
	FechaLlegada	DATE	-		
	Observaciones	VARCHAR	512		
	Resuelto	BOOL	-		✓

Tabla 6 – Diseño de los atributos del modelo de datos

Como se deduce de la anterior tabla, existen ciertos campos imprescindibles de tener algún valor no nulo en diferentes tablas:

- ***PFC_Lugar***: la definición de un lugar requiere obligatoriamente de una *dirección* y una *localidad*.
- ***PFC_Usuario***: la inclusión de un usuario en el sistema requiere obligatoriamente de su *DNI* y *password* o contraseña, ambos campos empleados en el acceso a la aplicación móvil del usuario (sección *4.3.1 Móvil usuario*).
- ***PFC_Itinerario***: en un itinerario es indispensable que figure si se ha *resuelto* o no, además de la *ruta* y la *fecha de salida*. De este modo, se permite la existencia de itinerarios que atiendan ciertas rutas para futuras fechas sin saber aún quien se encarga.

El caso del campo *CP* merece mención en lo que a su diseño se refiere. Es razonable pensar que este campo debería ser de tipo *INTEGER*, pero se opta por el tipo *VARCHAR* puesto que es interesante manejar dicha información tal y como la trata el usuario. En este sentido, si el usuario indica ceros a la izquierda del valor, el tipo *INTEGER* los suprimiría a la hora de almacenarlo en BD. Esta filosofía también la comparte el campo *DNI*, además por el hecho añadido de que se da la posibilidad de que el DNI pueda constar de una letra como último carácter. Esta decisión será respaldada por la validación implementada en ambos casos (ver en el próximo apartado el punto *4.2.2 Validación de formularios*).

Por último, resaltar el caso de la clave foránea (*Foreign Key,* FK) *IDusuario* en *PFC_Itinerario*. Por defecto, según el manual de referencia de MySQL (26), si no se precisa el comportamiento respecto a la clave foránea en la eliminación o actualización de un registro padre vinculado a otro hijo, se considera que no se debe llevar a cabo ninguna acción o, lo que es igual, se restringe la eliminación o actualización tanto en el registro padre como en el hijo. Pues bien, hemos considerado adecuada esta determinación en todos los casos de clave foránea de nuestro modelo, excepto en el caso de *IDusuario* vinculado a *PFC_Itinerario*, en el que se ha definido para la eliminación la acción *Set Null*. Es decir, en el supuesto de eliminar un usuario que tiene asignado uno o más itinerarios, se permitirá la eliminación del usuario, estableciendo los campos foráneos de ese usuario en *PFC_Itinerario* con valor *Null*.

Sin embargo, no se ha considerado igual para la otra clave foránea de *PFC_Itinerario*, *IDruta*, pues no tendría el mismo sentido y tampoco está permitido al haber sido definida como *Not Null*. Con esto se deduce que si se desea eliminar alguna ruta, se deberán eliminar previamente todos los itinerarios en los que consta. De igual modo se ha considerado para los lugares, de forma que para eliminar un lugar debemos cerciorarnos de que no consta en ninguna ruta. Otra posibilidad a considerar puede ser la definición de estas FKs como *Cascade* en la eliminación, de modo que no dependería del usuario el deshacerse de todas las vinculaciones existentes, sino que el propio gestor de BBDD eliminaría automáticamente todos los registros vinculados al que desea eliminar el usuario.

4.2 Entorno Web

En esta parte continuaremos con la exposición de la estructura y el funcionamiento de la aplicación web en mayor detalle, para lo que nos ayudaremos de diagramas UML. Inicialmente, vamos a representar un diagrama UML de estructura que refleja la distribución de las clases Java y sus paquetes. Seguidamente, pasamos a ilustrar otro diagrama UML de comportamiento o funcionamiento, que precisa más detalladamente cómo procede la aplicación web, pasando de la representación anterior de clases a la de instancias de éstas en objetos. Finalmente, a raíz de ambas representaciones procederemos con la explicación del diseño a bajo nivel e implementación de la parte web.

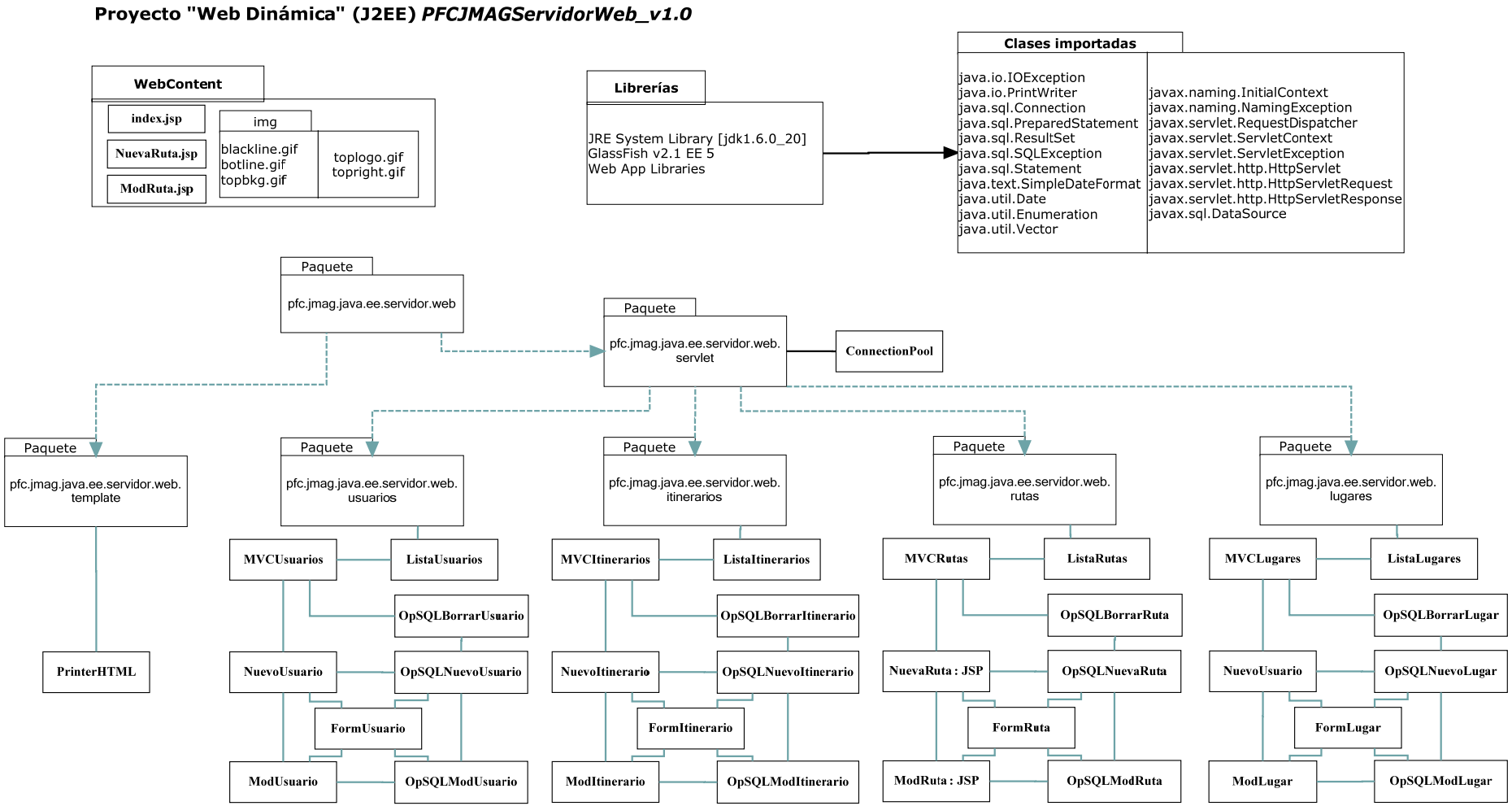


Figura 33 – Diagrama UML de estructura web

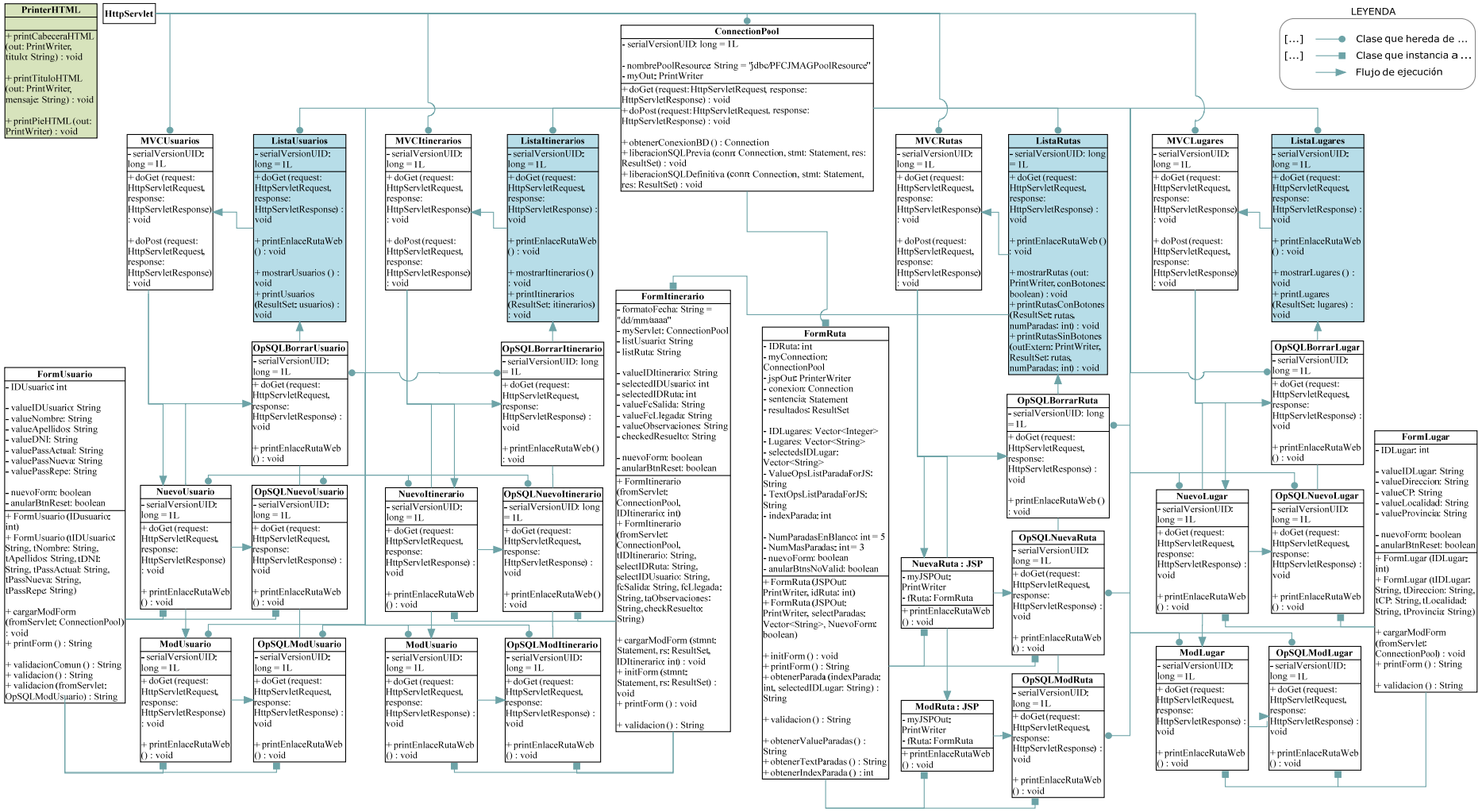


Figura 34 – Diagrama UML de comportamiento web

Sobre las figuras, observamos la división de la estructura en cuatro apartados: *usuarios*, *lugares*, *rutas* e *itinerarios*. Esta división del SI que muestra la aplicación web surge del propio modelo de datos. A continuación, pasamos a describir punto por punto la composición de clases Java que comparte cada uno de los apartados:

- **Formulario** (*Form_*). Se trata de la implementación del formulario que atiende a los campos o atributos de cada componente del SI. Se emplea en las acciones de creación o modificación y alberga la funcionalidad de validación de campos conforme a lo definido en el modelo de datos para evitar errores y aportar robustez; para tal fin, consta de un segundo constructor que, en caso de error de validación, permite volver a mostrarse con los datos que han motivado el error y la descripción del mismo. Esta sería la clase que responde al JavaBean.
- **Lista** (*Lista_*). Es uno de los servlets personalizados, es decir, que hereda de la clase *ConnectionPool* que aporta funciones de conexión a BD, como se explicó en *3.2 Entorno Web*. Como tal, se puede considerar que provee la página de inicio de cada apartado mostrando un listado de todos los elementos existentes para la parte en cuestión, dando también el acceso a las opciones de creación, modificación y eliminación propias de éstos y a la de volver a la página de inicio de la aplicación web.
- **MVC** (*MVC_*). Este sí es un servlet común (*HTTPServlet*) o, lo que es lo mismo, heredado directamente del API, pues no requiere ninguna función de lógica de negocios. Su cometido es simplemente el de reconocer la opción elegida desde la *Lista*, entre creación, modificación, eliminación y vuelta a la página de inicio, y redirigir el navegador al servlet o JSP correspondiente que atienda dicha petición. De este modo, esta clase responde al Controlador de la filosofía MVC (ver sección *2.2.4 MVC*), permitiendo la navegación ordenada a través de los servlets de cada apartado.
- **Nuevo** (*Nuevo_*). De nuevo, uno de los servlets tipificados del desarrollo, excepto en los casos de *usuarios* y *lugares* que no requieren conexión a BD pues sus formularios no tienen ningún campo que requiera información de esta. Su cometido es, sencillamente, el de presentar el formulario apropiado en blanco para una creación. Las opciones que presenta el formulario en este caso son: *Crear*, *Limpiar*, para volver a poner el formulario en blanco, y *Cancelar*, que sale de la creación sin llevar a cabo ninguna operación para devolvernos a la *Lista*.
- **Modificación** (*Mod_*). Esta es similar a la anterior, sólo que el formulario que presenta contiene los datos recogidos en BD para el elemento que se ha optado por modificar, por lo que esta vez se hereda obligatoriamente de nuestro servlet particular en todos los casos. Las opciones que presenta el formulario en este caso son: *Guardar*, *Restablecer*, para volver a mostrar los datos vigentes en BD, y *Cancelar*, que sale de la modificación sin llevar a cabo ninguna operación para devolvernos a la *Lista*.
- **OpSQLNuevo** (*OpSQLNuevo_*). Otro de los servlets a nuestra medida, que sucede a la opción de *Crear* de la clase *Nuevo* y procede, en primer lugar, a la validación de los datos del formulario que se desean insertar en el SI. En caso de ser exitosa la validación, se procede con las sentencias SQL convenientes y se muestra mediante mensaje el resultado satisfactorio, dando tras este la única posibilidad de *Volver* a la lista. Si, por el contrario, los datos introducidos en el formulario no son conformes a la validación, aparece de nuevo el formulario con

dichos datos anteponiendo el mensaje preciso de error de validación y facilitando así la corrección[13].

- **OpSQLModificación** (*OpSQLMod_*). Este es análogo al anterior, sólo que sucede a la opción de *Guardar* de la clase *Modificación* y, con ello, procede a la validación de los datos que se desean actualizar en el SI. De nuevo, si la validación es favorable se da paso a las sentencias SQL adecuadas para la actualización y se muestra el resultado; y, si los datos no son convenientes, se anuncia en qué parte requieren corrección seguido del mismo formulario que se ha tratado de guardar.
- **OpSQLBorrar** (*OpSQLBorrar_*). Por último, otro de los servlets desarrollados con conexión a BD para la eliminación de algún elemento del SI tras la opción de *Eliminar* hayada en la *Lista* de estos elementos. Tras la ejecución de su lógica de negocio, aparece en mensaje el resultado obtenido y la opción de *Volver* a la *Lista*. (Únicamente en el caso de acometer la restricción de eliminación de un *lugar*, según el apartado anterior *4.1 Modelo de Datos*, el mensaje será el mismo que el devuelto por el propio gestor MySQL.)

Como culmén de este punto, citar simplemente la implementación de la página de inicio mediante un JSP (*index.JSP*) que mantiene la apriencia de toda la aplicación web, dando la bienvenida y la posibilidad de acceder a cada uno de los apartados.

4.2.1 Casos excepcionales en el diseño

A la hora de diseñar el formulario HTML para una ruta hemos visto oportuno ofrecer al usuario algún método para añadir paradas indefinidamente, y así aprovechar la flexibilidad que nos ofrece el modelo de datos ideado. Al fin y al cabo, esta funcionalidad no se concibe de forma más sencilla si no es corriendo a cargo del navegador.

Por este motivo, nos planteamos para el diseño de este formulario el contenido de una función que responda a la acción de pulsado sobre un botón denominado *Más Paradas*, el cual añadirá nuevos campos de parada adicionales a los ya presentes en el formulario, respetando el orden de paradas existente.

Al mismo tiempo, este botón requiere información sobre los posibles lugares de parada existentes en BD. Por ello, desde la función del script ejecutado por el navegador es necesario consultar información al servidor web, al menos al inicio mientras se carga la página[14]. Ante tal problemática, surgen dos opciones:

- Emplear **JavaScript**, que integrado con JSP permite precisamente esa particularidad adicional de que la funcionalidad a cargo del navegador pueda tratar metódicamente con el servidor al cargarse la página a visualizar.
- Emplear **Ajax**, similar a JavaScript pero más avanzado, por lo que junto a JSP también permite, respecto a lo que nos interesa, justamente lo mismo pero no sólo

[13] En este nuevo formulario se suprime el botón de *restablecer*, tal y como se detalla en *4.2.2 Validación de formularios*.

[14] Con esto, tal y como se ha anotado anteriormente, no se contempla la concurrencia de datos para la interfaz de usuario, de modo que desde la aparición de un formulario de ruta en el navegador podrá cambiar la información de lugares por parte de otro acceso a la aplicación web que no se verá reflejado en los controles de paradas de dicho formulario.

al inicio, sino también después, cuando ya se ha cargado la página, mientras se visualiza o se interactua con ella.

Según lo descrito, JavaScript es suficiente para lo mínimo que necesitamos, aunque con Ajax podría obtenerse la información siempre reciente de modo que, si exitiesen cambios en los lugares mientras se visualiza la página, serían efectivos en los nuevos campos de parada generados mediante el botón *Más Paradas* (como se puede deducir de la segunda nota al pie de la anterior página). Dado que es un hecho que no hemos tenido en cuenta en otros casos (respecto a la concurrencia en el interfaz de usuario, anotada ya en más de una ocasión) y conlleva mayor complejidad en su implementación, nos decantamos finalmente por la utilización de código JavaScript.

Todo este planteamiento supone un cambio en la estructura ideada para este caso particular del formulario de ruta, donde sustituyamos las clases que representan a servlets con dicho formulario por ficheros JSPs, puesto que desde los mismos servlets de Java no se admite la inclusión de código JavaScript (como ha quedado patente en el capítulo del estado del arte, *2.2.2 Servlet – JSP*). Con dicho cambio conseguiremos que el script, en la carga inicial del navegador, acceda a la información que necesita mediante el uso de *tags* o etiquetas, que nos permitan ejecutar código Java contenido en el servidor como si de un servlet se tratase. De esta forma, extendemos este apartado con una descripción más detallada sobre las etiquetas de JSP y ultimamos el diseño definitivo de esta parte.

Antes recalcar que, dada esta particularidad se podría reformular la estructura del proyecto web sustituyendo en todos los casos los servlets por JSPs (más versátil sin duda por motivos como el que nos hemos encontrado y los ya comentados en *2.2.2.2 JSP*); pero creemos preferible conservar los servlets mientras sea posible puesto que, a pesar de la multitud de sentencias *println* que se deben realizar a través de él para obtener resultados HTML en el navegador del cliente, sigue siendo menos costoso en proceso que la conversión transparente que se debe realizar de un JSP a servlet para su funcionamiento (ver *2.2.2.2 JSP*). De hecho, de los cuatro servlets que pueden emplear el formulario de ruta – *NuevaRuta*, *ModRuta*, *OpSQLNuevaRuta* y *OpSQLModRuta* – únicamente hemos definido como JSP los dos primeros por considerarlos los estrictamente necesarios. Respecto a los otros dos, encargados de la validación y ejecución de las transacciones SQL oportunas, y que también pueden recargar el formulario por no superar la validación, hemos considerado trivial que estuvieran dotados de la funcionalidad *Más paradas* pues su formulario es copia del que no ha pasado la validación y su finalidad es la de rectificar los errores, con lo que se entiende que no habrá cambios sustanciales en el mismo, como añadir más paradas. Además, se sumaba el inconveniente de la complejidad que ha supuesto la implementación como JSP respecto al manejo de la conexión a BD, por no ser posible extender en el JSP del servlet personalizado (como se precisa en el apartado *6.3 Limitaciones*), cuando éstos requieren fundamentalmente fácil acceso a BD para ejecutar sus sentencias SQL.

En un fichero JSP, los elementos que se encuentran entre las etiquetas **<%** y **%>** son elementos Java, mientras el resto es considerado HTML puro. Además de estas etiquetas, existen otras variaciones empleadas en nuestro desarrollo que son las siguientes: (19)

- **<%!** Declaración Java **%>**, para declarar funciones y atributos.
- **<%=** Expresión Java **%>**, para evaluar métodos o variables.

Así, los servlets que manejen este formulario de ruta, ahora en forma de JSP, serán en implementación lo más parecido al resto de servlets, con la particularidad de que en este caso se cambia la gestión de la conexión con BD al JavaBean o formulario, dados la imposibilidad de extender el JSP de nuestro servlet "padre". Por otro lado, la información que requiere JavaScript, por la que no olvidemos se cambia todo el esquema, será consultada mediante metodología que dispone también el formulario de ruta. En referencia al momento de solicitar dicha información, debemos forzar que sea tras la construcción e impresión del formulario, llevado a cabo como decimos gracias al código Java incluido en las etiquetas de la forma más parecida al resto de servlets equivalentes de otros apartados, para así obtener también el índice de la última parada cargada inicialmente en la página. Por ello, se declara el código JavaScript al final del fichero JSP o tras dicho código Java.

Por último, comentar que la implementación contempla el caso en que el navegador no permita la ejecución de código JavaScript, por motivos de seguridad, incompatibilidad o cual sea, manifestándolo mediante un mensaje bajo el formulario. En esta situación, no se puede plantear ninguna solución para la funcionalidad aquí perseguida, únicamente se puede sugerir el siguiente remedio en caso de querer aumentar la ruta en paradas con respecto a las que aparecen en el formulario: crear o guardar cumplimentando todas las paradas posibles y posteriormente modificar, pues según dicta la implementación aparecerán siempre en la edición cinco nuevas paradas más en blanco siguiendo a las que ya existían, y así sucesivamente se podrán incluir también de forma indefinida las paradas deseadas.

4.2.2 Validación de Formularios

En el manejo de todos los campos de los formularios destinados a alterar la información del sistema, ya sea en su inserción o modificación, se ha controlado su formato y longitud para evitar errores por parte del gestor MySQL.

Por un lado, la longitud se controla gracias a la definición HTML del atributo *MaxLength* en cada campo del formulario al límite impuesto por el modelo de datos. Por otro lado, el formato y otros aspectos de validación se han controlado mediante la implementación Java en la clase *Formulario* (JavaBean) de un método de validación, siguiendo ciertos criterios para cada caso:

- **Lugar**
- ***Dirección y Localidad***. Como atributos *Not Null* se comprueba que conste algún valor no nulo.
- ***CP***. En caso de tomar algún valor, nos cercioramos de que sea de la longitud correspondiente (cinco dígitos), marcada como máxima mediante HTML, y de que los caracteres introducidos sean numéricos.

- **Ruta**

  Paradas. En la lista de paradas que muestra este formulario se controla, además de que conste al menos una parada en primer lugar, el orden de las mismas, es decir, que todos los campos de parada con algún valor aparezcan sucesivamente desde el primero. Este criterio de validación se ha considerado para evitar

confusiones, exigiendo al usuario que respete el orden de los campos de parada denotado en sus etiquetas.

❖ **Usuario**

- ***DNI***. Como atributo *Not Null* se comprueba que conste algún valor no nulo, tras lo cual también se comprueba su longitud mínima (ocho dígitos, sin considerar la letra) y que el valor sea numérico, estrictamente en su longitud mínima; pues de ser mayor la longitud, se tiene en cuenta que pueda llevar añadido en el último carácter la letra, la cual también se comprueba que no sea cualquier otro carácter que no comprenda de la 'A' a la 'Z' (admitiendo la letra en minúscula, pues será convertida e interpretada automáticamente en mayúscula, al igual que en el proceso de validación de usuario de la aplicación móvil).
- ***Contraseña*** (creación). Como atributo *Not Null* se comprueba que conste algún valor no nulo.
- ***Cambio de Contraseña*** (edición). Para esta funcionalidad nos encontramos con tres campos: *Contraseña Actual*, *Nueva Contraseña* y *Repetición Nueva Contraseña*. Si se dejan en blanco dichos campos tal funcionalidad no surte efecto, siendo perfectamente válida la operación. Sin embargo, para cambiar de contraseña es necesario rellenar los tres campos de forma que la *Contraseña Actual* efectivamente coincida con la registrada hasta el momento y las otras dos sean una réplica la una de la otra; de no cumplirse alguna de estas condiciones, habiéndose rellenado alguno de los tres campos, la operación no se llevará a cabo y se informará al usuario de las inconformidades.

❖ **Itinerario**

- ***Ruta* y *Fecha de Salida***. Esta será la información imprescindible para la existencia de un itinerario. Dado el modo de funcionamiento de cualquier empresa a la que se pueda destinar este sistema, se ha visto lógico el que pueda existir la planificación de un servicio sobre una ruta sin saber aún quien será el encargado de atenderlo.
- ***Fecha de salida* y *de Llegada***. Estos campos, como veremos en su correspondiente apartado, son susceptibles de mejora. Hasta donde nos ocupa, hemos llevado a cabo una implementación básica que se encarga de controlar que el campo conserve el formato "dd/mm/aaaa" (definido en un atributo final del formulario itinerario, *FormItinerario*). Para ello se controla: que la longitud sea completa, es decir, que el valor comprenda la longitud máxima determinada por el atributo HTML *MaxLength*, como ya indicamos; y que el carácter de separación sea '/' y que, entre estos, se incluya una fecha válida o existente. Otro aspecto de validación implementado es la imposibilidad de incluir una fecha de salida posterior a la de llegada.
- ***Observaciones***. Este campo, por su extensión, hemos visto conveniente que fuera algo más extraordinario que los que venimos empleando comúnmente en nuestros formularios HTML, de etiqueta *input*. Nos referimos a un campo de etiqueta *textarea* al que ya no podemos definir un atributo *MaxLength*, por lo que debemos controlar en su validación la longitud máxima marcada por el modelo de datos (512 caracteres).

Si alguna de estas validaciones no son superadas a la hora de ejercer la operación en cuestión, supone la reaparición del formulario con los campos tal y como se cumplimentaron y con mensajes indicativos de los errores de validación cometidos, para

dar opción a la rectificación y reintento de la operación con BD. En el caso de este formulario de corrección, no se incluye el campo HTML *Reset* en forma de botón, pues es una funcionalidad que restablecería la información inicialmente cargada que, en este caso, se trataría de la información no validada y ello carecería de cierto sentido.

También merece mención el hecho de que las claves primarias de cada tabla, que comienzan todas por *ID*, se muestran en la aplicación web únicamente en el formulario de edición como referencia y definiéndolo en HTML con el atributo *readonly* para impedir perceptiblemente su modificación (apreciable en la *Figura 26 – Formulario para la Edición de Ruta*).

Resaltar que, por otra parte, sólo se ha comprobado el cumplimiento de la restricción de borrado por FK para las rutas en itinerarios, ya que se podía dar la indeseable situación de que dicha violación se produjese en la última operación de la eliminación, con la supresión definitiva de la ruta, tras haber eliminado ya todas sus paradas. Esta situación podría variar de planteamiento si no se hubiera optado en el diseño por la opción *autocommit* del gestor de BBDD, con la que cada sentencia se ejecuta independientemente, como se indicó al inicio del *Capítulo 3 – Descripción del Sistema: Alto Nivel*. (Se menciona el otro caso no contemplado de restricción de borrado por FK para lugares en el punto de la clase Java para el borrado, *OpSQLBorrar*, en la descripción del comportamiento de esta sección *4.2 Entorno Web*.)

4.3 Entorno Móvil

Todo el desarrollo llevado a cabo para su funcionamiento en dispositivos móviles ha resultado en dos *MIDlets* o, lo que es lo mismo, dos aplicaciones para móviles, una para el funcionamiento del móvil intermediario y la otra para el móvil de usuario; estando ambas empaquetadas en un mismo *MIDlet Suite*.

Por otro lado, consideramos también parte de este entorno la aplicación que corre a cargo del servidor, donde se aloja el SI, esta vez atendiendo peticiones de los móviles mediante acceso inalámbrico de corto alcance, o la aplicación que denominaremos servidor bluetooth. Hacemos ahora una representación de la estructura resultante de esta parte mediante un diagrama UML de paquetes y clases, para a continuación explicar brevemente cada una de sus tres aplicaciones y ahondar en mayor detalle, a lo largo de los siguientes subapartados, sobre sus respectivos comportamientos en lo que respecta a las comunicaciones inalámbricas en las que intervienen.

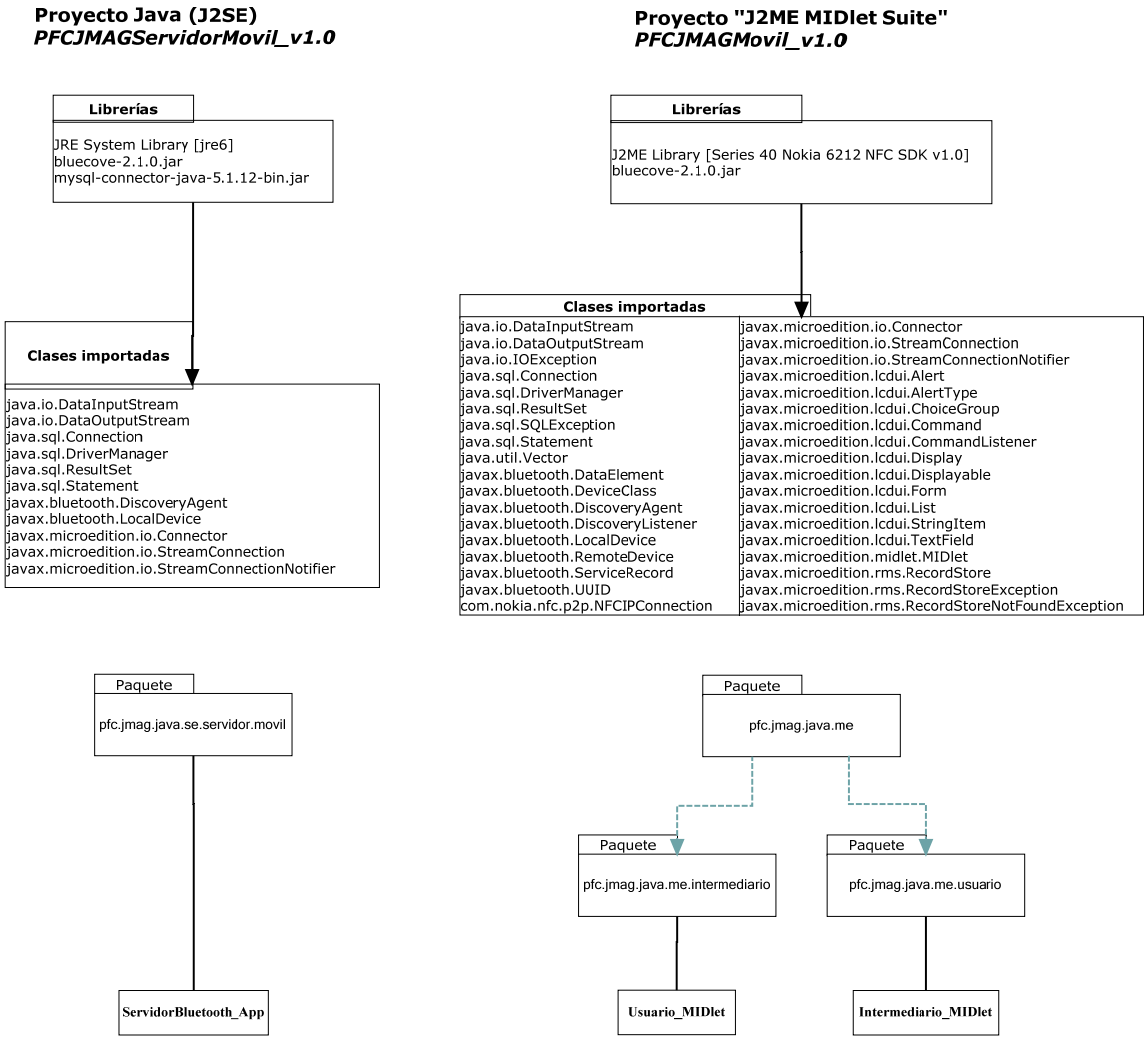


Figura 35 – Diagrama UML de estructura del Entorno Móvil

- ***Usuario_MIDlet***. Es la aplicación destinada al uso por parte de los encargados de los desplazamientos. Esta ha sido la más comentada hasta el momento, con Sistema de Validación del Acceso de usuario y opciones de recibir, ver y entregar itinerarios.
- ***Intermediario_MIDlet***. Es la aplicación que permite la transición entre las tecnologías inalámbricas empleadas para la comunicación entre servidor y móvil usuario, bluetooth y NFC respectivamente. A esta aplicación y al dispositivo que la ejecute se les denomina intermediario, por el hecho de mediar entre ambas comunicaciones.
- ***ServidorBluetooth_App***. Esta aplicación emplea el dispositivo bluetooth del servidor para atender las solicitudes originadas por el usuario desde su móvil a través de la aplicación *Usuario_MIDlet*, y por vía del móvil intermediario y de su correspondiente aplicación *Intermediario_MIDlet*.

4.3.1 Móvil Usuario (*Usuario_MIDlet*)

4.3.1.1 Almacenamiento de Itinerarios

Para almacenar datos en el móvil empleamos la clase *RecordStore* facilitada por el API de MIDP. Cada usuario tiene un *RecordStore* o un apartado de almacenamiento para sus itinerarios en el móvil, de modo que al identificarse en el acceso, una vez validado, se accede al *RecordStore* oportuno para obtener la información vigente en el móvil del usuario en cuestión. De este modo es como se permite el uso del mismo dispositivo móvil por más de un usuario, como se citó en *3.3 Entorno Móvil* de la descripción a alto nivel.

Como ya se ha mencionado, la aplicación permite la visualización de la información relativa a los itinerarios del usuario mediante la opción *Ver Itinerarios*. Aun así, buscando una mayor optimización, no se accede siempre al *RecordStore* para leer su información, sino que se lee de una variable de sesión de la aplicación que accede al *RecordStore* para tomar su información cuando es estrictamente necesario. Las acciones que implican lectura o escritura de dicha variable de sesión son:

- Tras el reconocimiento y validación del usuario se hace una primera lectura del *RecordStore* para adquirir sus datos en la variable de sesión, si es que existen.
- Análogo al inicio de sesión anterior, se lee el *RecorStore* para actualizar la variable de sesión tras la recepción de itinerarios pues, en el caso más elemental de que no existiesen ya itinerarios en el móvil para el usuario, la información provista por la comunicación con el intermediario – servidor se almacena en el *RecordStore*. El otro caso, en el que existen itinerarios del usuario antes de la recepción, se da la opción al usuario de eliminar dicha información para poder hacer la recepción; de modo que, si el usuario así procede, se borrará el *RecordStore*, con la actualización de variable de sesión incluida, es decir, borrado de su contenido, y se pasará en seguida al caso elemental recién descrito.
- En el guardado de un itinerario, esto es, de los cambios producidos en el formulario de edición de un itinerario, que se realiza directamente sobre el *RecordStore* para, de nuevo, actualizar posteriormente la variable de sesión.
- En la entrega de itinerarios que, si resulta satisfactoria, eliminará el *RecordStore* correspondiente y, con ello, limpiará la variable de sesión.

4.3.1.2 Almacenamiento del Sistema de Acceso de Usuario

El Sistema de Validación de Acceso para la aplicación de usuario requiere de la información de usuarios del sistema. Para poder permitir el uso de esta aplicación en cualquier lugar es necesario tener dicha información almacenada en el mismo dispositivo donde se ejecute, si no se desea disponer de alguna comunicación de larga distancia como puede ser la red móvil (GSM, 3G,…), que no es del alcance de este proyecto y nada recomendable en este caso dado su alto coste y, además, su requisito más exigente de mejora en seguridad (ver *6.2 Trabajos futuros* al respecto). Así, existe la funcionalidad *Actualizar Login*, accesible como opción desde la misma pantalla de acceso, que permite la actualización de la mencionada información mediante un proceso similar al de la solicitud de recibir itinerarios. Para el almacenamiento de esta información se emplea otro *RecordStore* aparte, el cual se lee tan sólo para realizar la validación del usuario que accede a la aplicación.

4.3.2 Móvil Intermediario (*Intermediario_MIDlet*)

Destacaremos las principales características de esta aplicación:

- No almacena nada.
- Se trata de un hilo de ejecución continua que actúa a modo de servidor a la escucha de peticiones de los móviles de usuario, a pesar de que el auténtico servidor es el ordenador que alberga el SI (que es a la vez servidor web) y el móvil intermediario sólo ejerce como pasarela entre comunicación NFC y bluetooth.
- Para la búsqueda del servicio bluetooth que atiende las solicitudes que le llegan por parte del usuario emplea tres parámetros:
    - Nombre del servidor bluetooth o, más concretamente, del equipo o componente que presta servicio a través de su dispositivo bluetooth.
    - Nombre del servicio bluetooth.
    - UUID del servicio bluetooth (27).
- No admite concurrencia en las solicitudes, es decir, no puede atender a más de un usuario a la vez.
- Su pantalla refleja, mediante mensajes a modo de *log*, la evolución de las comunicaciones originadas por cada petición que atiende, dando la opción de *Limpiar* la pantalla al finalizar cada una.

4.3.3 Servidor Bluetooth (*ServidorBluetooth_App*)

El servidor bluetooth, a través del intermediario, aporta la respuesta a las solicitudes por parte del usuario de recepción y entrega de itinerarios y de actualización del sistema de acceso de usuario.

Su ejecución es también continua, al igual que en la aplicación intermediaria, a la espera de atender solicitudes. Como no podía ser de otra manera, en su arranque realiza la configuración del dispositivo bluetooth, con el que prestará físicamente su servicio, para que pueda ser descubierto en un proceso de búsqueda; y sobre éste incorpora dicho servicio bluetooth específico del sistema, con ciertos parámetros de publicación conocidos por la aplicación intermediaria (los tres enumerados en el punto anterior, definidos en ambas aplicaciones como constantes).

De la implementación de esta aplicación, cabe destacar en la recepción de itinerarios lo que se ha considerado oportuno obtener del SI para esta petición del usuario que se trata, como ya se ha indicado con anterioridad, de los itinerarios que le corresponden con fecha actual o anterior y sin resolver. Hacemos esta mención de nuevo pues se podría considerar de otro modo y variar la consulta que se hace a BD (especificada en *B.4 Sentencias SQL en el Entorno Móvil*).

4.3.4 Implementación de las comunicaciones inalámbricas: NFC y bluetooth

Esta parte se refiere a las diferentes situaciones que se pueden dar en las comunicaciones inalámbricas. Son cuatro casos los que nos encontramos y surgen a raíz de los dos condicionantes siguientes:

- **Búsqueda bluetooth**. El MIDlet intermediario deberá realizar una búsqueda de dispositivos bluetooth, por ejemplo en su arranque, para tratar de localizar al servidor bluetooth y, consecutivamente, al servicio que atiende las solicitudes del usuario. Una vez obtenido el parámetro (URL) para la conexión al servicio bluetooth, se conserva para futuras conexiones sin tener que volver a realizar el proceso de búsqueda.
- **Actividad del servidor bluetooth**. En el momento de la conexión bluetooth es posible que, por cualquier motivo, el proceso servidor bluetooth se haya caído o interrumpido.

Cabe destacar entre todos los casos que si se debe realizar una búsqueda bluetooth la comunicación NFC se cerrará para volver a restablecerse tras dicho proceso de búsqueda. Se ha considerado esta implementación para no forzar innecesariamente al usuario a que permanezca en contacto con el móvil intermediario inmóvil y evitar así abundantes errores. Todas estas eventualidades son perfectamente reveladas por mensaje a través de las pantallas de ambos móviles, intermediario y usuario, como veremos detalladamente en las figuras que siguen. Con esto y los dos condicionantes anteriormente señalados, enumeramos a continuación los casos contemplados para las comunicaciones que se dan en el entorno móvil:

- El primer caso se espera que sea el común para el arranque de ambas aplicaciones que intervienen en la comunicación suscitada a raíz de la petición de algún usuario, la del móvil intermediario y la del servidor bluetooth. El intermediario deberá realizar la búsqueda del dispositivo y servicio bluetooth puesto que desconoce al comienzo la URL para la conexión. Dado que el servidor se encuentra activo, la búsqueda y el establecimiento de la conexión bluetooth serán exitosos. Con el final de la comunicación se cerrará la conexión bluetooth, al igual que la de NFC (la segunda restablecida), y quedará la URL en forma de variable, que le permite conectar con el servidor bluetooth, en la aplicación intermediaria mientras aguante activa o lo permitan otros motivos aquí expuestos.
- El segundo caso sería una variante del anterior, en el que la aplicación intermediaria ha sido arrancada e involucrada en una comunicación por algún usuario, pero servidor bluetooth se encuentra inactivo. En tal situación, la búsqueda será en vano y así será comunicado al usuario en la segunda conexión NFC que se ha de establecer tras dicho proceso.
- Los siguientes casos carecen del proceso de búsqueda bluetooth, con lo que, reincidiendo, la conexión NFC será única de principio a fin. Con la URL del servicio bluetooth ya dispuesta se trata de establecer la conexión directamente. En este caso, consideraremos que el servidor bluetooth permanece activo, pues en algún momento lo estuvo para ofrecer la URL de su servicio bluetooth, o al menos no se encuentra inactivo, pues es posible que desde la búsqueda haya caído y se haya vuelto a restablecer. Así, la comunicación será satisfactoria si ninguna otra

eventualidad lo impide y, es más, mucho más breve puesto que habremos prescindido del proceso de búsqueda bluetooth.

- En el último caso, además de disponer ya de la URL del servicio bluetooh, con lo que se prescinde de realizar la búsqueda bluetooth, nos encontramos con que el servidor ha caído y se encuentra inoperativo. Así, con la misma rapidez que en el anterior caso, se le comunicará dicho incidente al usuario, sugiriéndole que vuelva a intentarlo tras el restablecimiento del servidor bluetooth. En este caso, se limpiará la variable de la aplicación intermediaria que conserva la URL del servicio bluetooth, obligando en la siguiente comunicación a llevar a cabo de nuevo una búsqueda para resolver si la aplicación servidora vuelve a estar en pie o no, tal y como detallan los dos primeros puntos.

Existe otro caso que por su infrecuencia no lo hemos considerado, aunque la implementación resulte robusta ante el mismo y hagamos una mínima referencia. Es el caso en el que, tras una búsqueda satisfactoria del servicio bluetooth, se proceda a su conexión y justo se encuentre inoperativo. Es presumible que las posibilidades de este caso sean ínfimas dado que el proceso servidor debe caer justo entre el final de la búsqueda exitosa y el intento de conexión, que se trata de un tiempo de ejecución inapreciable; aún así, dicha trivialidad está contemplada y en caso de producirse será tratada convenientemente y comunicada al usuario.

A continuación se muestran unos diagramas UML de secuencia que reflejan el comportamiento y transcurso de los cuatro casos arriba descritos para las dos comunicaciones primordiales, recepción y entrega de itinerarios, que tienen lugar entre los tres elementos del entorno móvil. Se obvia el diagrama correspondiente a la actualización de login por ser análoga a la recepción de itinerarios, como ya se indicó. Síncronos a los diagramas, se incorporan los mensajes que aparecen en la pantalla de cada elemento.

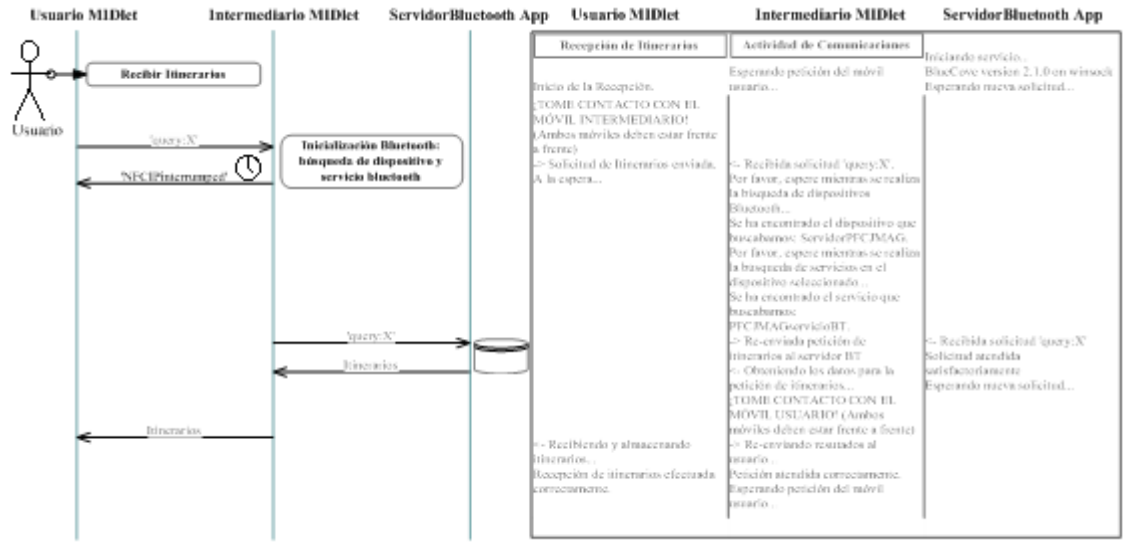


Figura 36 – Diagrama UML secuencial de la Recepcion de Itinerarios (caso 1)

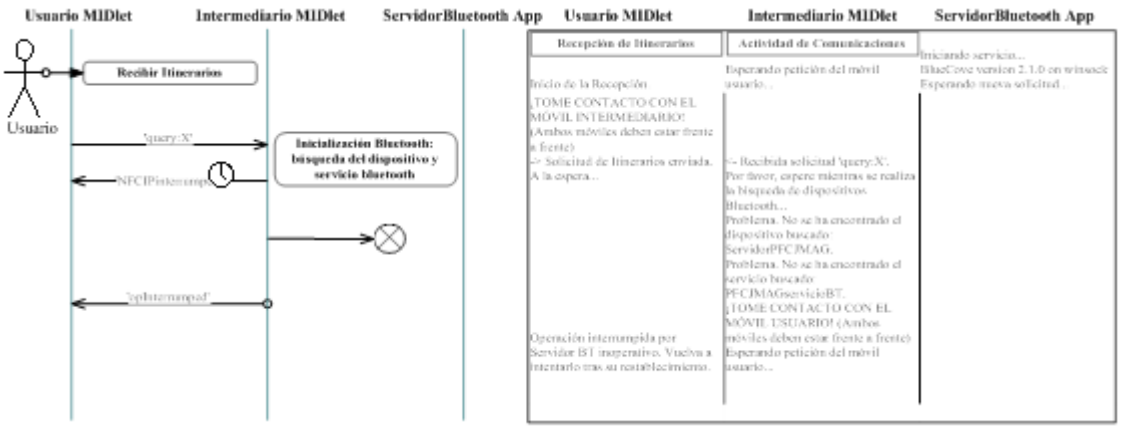


Figura 37 – Diagrama UML secuencial de la Recepcion de Itinerarios (caso 2)

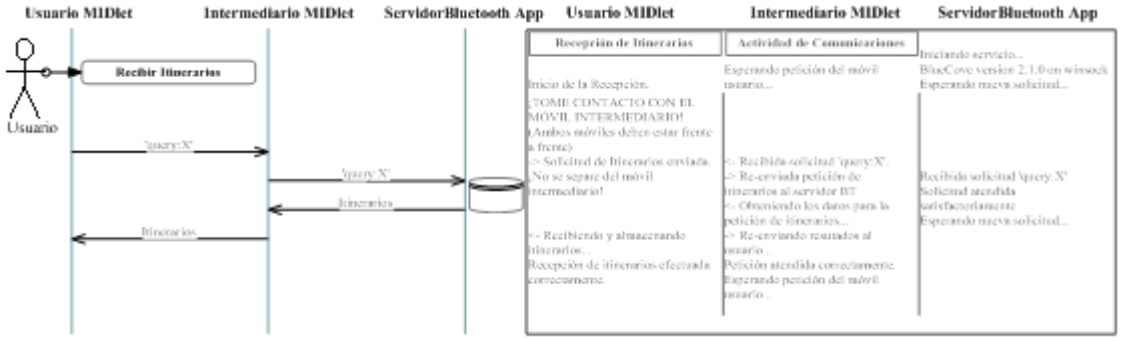


Figura 38 – Diagrama UML secuencial de la Recepcion de Itinerarios (caso 3)

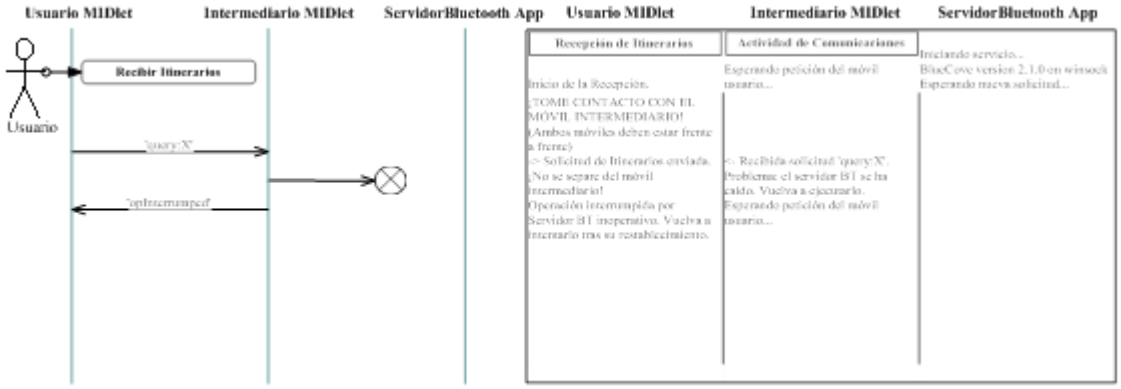


Figura 39 – Diagrama UML secuencial de la Recepcion de Itinerarios (caso 4)

Para la recepción de itinerarios existe una particularidad extra sobre la que ya hemos hecho mención en este mismo apartado y el de *3.4 Casos de Uso*. Si en el momento de la recepción existieran ya unos itinerarios en el móvil del usuario, se da la opción a priori de eliminarlos para poder realizar la recepción que se pretendía, advirtiendo de la posible pérdida de los datos actuales en el móvil; o de cancelar la recepción, opción bastante factible si el usuario no era consciente de este hecho.

En caso de que no existiesen itinerarios asignados al usuario y la recepción sea nula, así se le notifica al usuario al cabo de todos los mensajes de la comunicación.

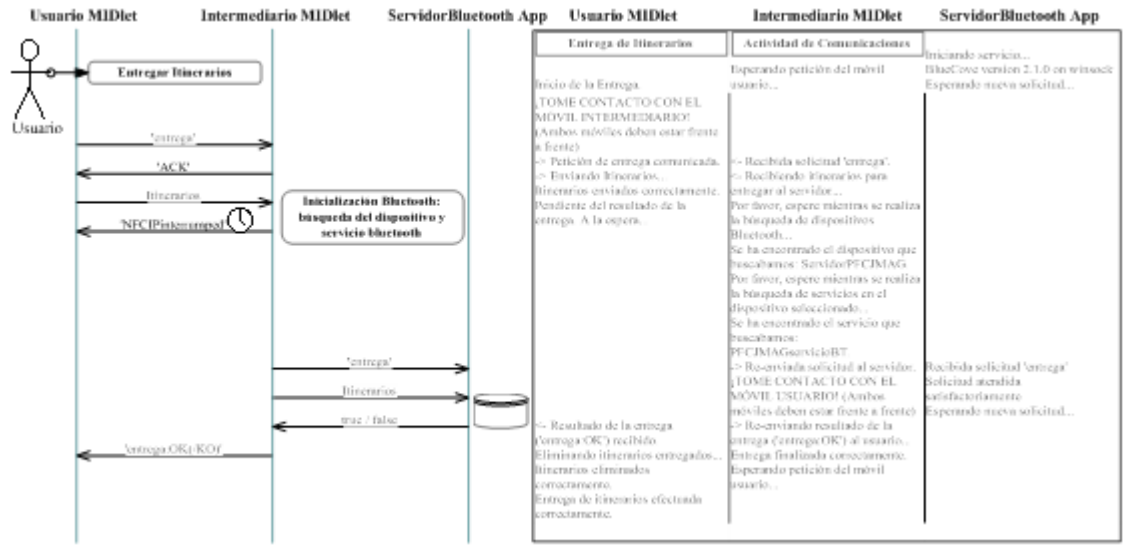


Figura 40 – Diagrama UML secuencial de la Entrega de Itinerarios (caso 1)

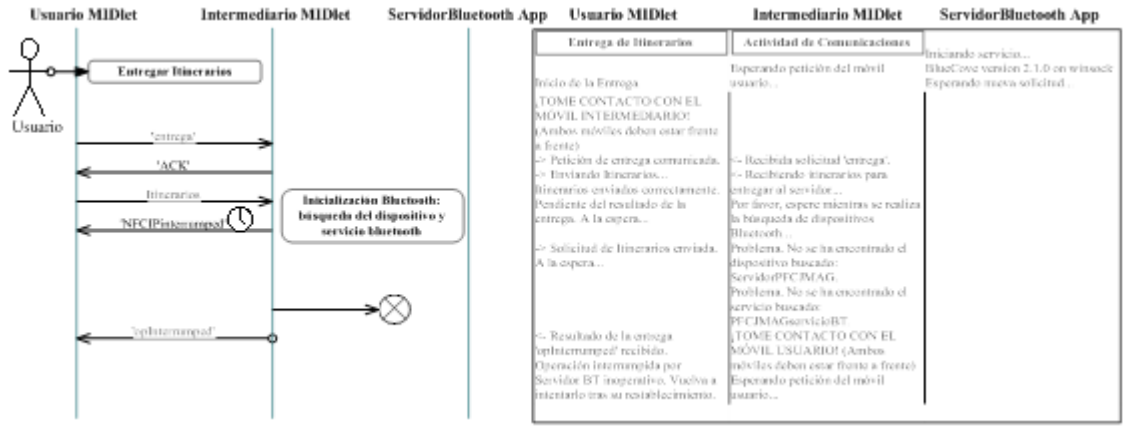


Figura 41 – Diagrama UML secuencial de la Entrega de Itinerarios (caso 2)

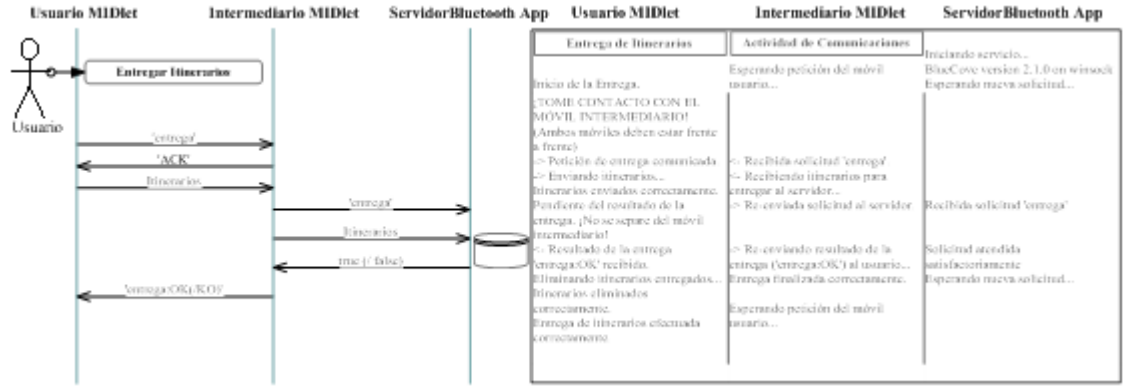


Figura 42 – Diagrama UML secuencial de la Entrega de Itinerarios (caso 3)

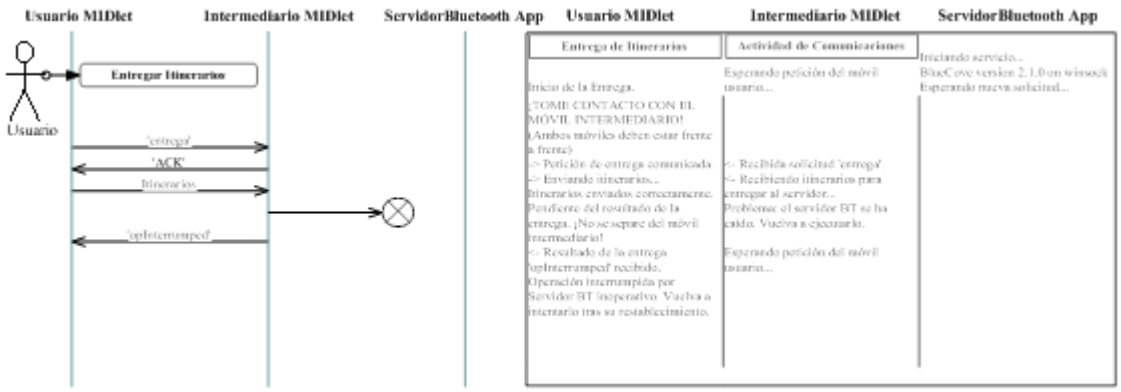


Figura 43 – Diagrama UML secuencial de la Entrega de Itinerarios (caso 4)

Respecto a la entrega de itinerarios, cabe destacar la respuesta final que recibe el usuario como resultado de la entrega, que en la comunicación bluetooth se tratará de un booleano para indicar éxito o fracaso de la entrega, y en la comunicación NFC se traducirá a "entrega:OK" o "entrega:KO", respectivamente. En caso de ser correcto este resultado, se procederá al borrado del móvil usuario de los itinerarios entregados.

4.3.5 Optimización de las comunicaciones

La primera optimización concierne a la parte NFCIP de cualquiera de las comunicaciones entre ambos móviles. En un principio, análogo a como está implementada la parte bluetooth, en esta parte se implementaron tantas secuencias de envío y recepción como campos del SI se pretendían comunicar, confirmando con el texto "ACK" en la respuesta dada la condición en NFCIP de que por cada envío debe efectuarse su correspondiente recepción (según lo expuesto en *2.3.5.1 NFCIP-1 en SDKs de Nokia*). Tras innumerables pruebas de las comunicaciones en las que surgían errores, pensamos que podían ser originados por esta parte debido a tantas secuencias de envío y recepción y decidimos cambiar la implementación integrando todos los campos objetos de la comunicación en un solo envío, con forma de cadena de caracteres y estableciendo un carácter limitador o de diferenciación entre el final de un campo y el comienzo del siguiente (el carácter elegido fue 'ç' por ser extraordinario, aunque quizás no lo sea en algunos idiomas, y por no encontrarlo entre los caracteres disponibles en el móvil). Esta nueva disposición dio resultado, compensando la nueva restricción que nace de esta optimización como es la prohibición en el sistema de dicho carácter, o dando pie a mejorar la optimización mediante la interpretación de otro carácter de escape de este prohibido.

Por otro lado, se ha diseñado otra optimización en lo que respecta a la comunicación de recepción de itinerarios por parte del móvil usuario. Esta optimización se ha implementado en el envío de itinerarios como respuesta que hace el servidor bluetooth cuando recibe dicha petición, una vez ha realizado la consulta pertinente, y abarca toda la transmisión de extremo a extremo. Y es que, dada toda la información de paradas que se desea obtener de la ruta de un itinerario y el algebra empleada en su consulta SQL, nos encontramos con que el resto de información del itinerario y de la ruta está duplicada tantas veces como paradas tiene la ruta correspondiente; es decir, de no haber optimizado esta parte, todos los atributos correspondientes a la tabla *pfc_itinerario* (fecha de salida y

de llegada y observaciones) y el nombre de la ruta (si no tuviera sería su vista alternativa, ver apartado *B.2 Vista alternativa de campos Nombre*), se enviarían de forma redundante. Así, esta mejora consiste simplemente en excluir toda información redundante en esta transmisión, quedando únicamente dicha información junto con la primera e indispensable parada de la ruta asociada.

Finalmente, esta segunda optimización no supondría inconveniente para la parte bluetooth del envío, dado que su implementación, en cualquier transmisión, es de tantos envíos de campos como se precise (sin requerir también recepciones como en NFCIP) y sincroniza al cabo de cada registro con un *booleano* para indicar si se continua con más o no. Sin embargo, para la parte NFCIP habrá registros en los que se haya suprimido la información redundante y otros en los que no, perdiendo así la sincronización innata de tener un número fijo de campos para todos los registros enviados. Por tanto, la primera optimización debe adecuarse con la incorporación de la segunda, de modo que incluyamos también otro carácter limitador, pero esta vez entre registros; así, en la recepción, será posible desencadenar toda la información íntegra en un *String* de forma correcta. Esta vez, el carácter elegido es 'Ç', la misma letra que en el caso anterior pero en mayúscula y por los mismos motivos; que, por implementación, se incluye al cabo de todos los registros. Entonces, tendríamos otro carácter prohibido con las mismas consecuencias y consideraciones citadas para el anterior.

Capítulo 5

Pruebas

5.1 Entorno Web

En esta parte, cabe únicamente la variedad de pruebas que se le pueden realizar al gestor de un Sistema de Información, tratando de forzar los casos de error de validación más habituales.

Prueba	Resultado Esperado	Resultado obtenido
Creación de nuevo usuario, sin DNI ni contraseña.	Creación exitosa tras corrección de las invalidaciones notificadas de DNI y contraseña obligatorios.	✔
Creación de nuevo usuario, sin Nombre ni Apellidos.	Creación exitosa tras corrección de las invalidaciones notificadas por valores erróneos de DNI: '1234567', '1234567A', '12345678AB', '12345678-B' y '012345678@'.	✔
Modificación de usuario para cambio de contraseña.	Modificación exitosa tras corrección de las invalidaciones notificadas por: repetición de contraseña no coincidente y contraseña actual errónea.	✔
Eliminación de usuario que tenga asignado algún itinerario.	Eliminación exitosa, quedando los itinerarios del usuario sin asignar a nadie.	✔
Creación de nuevo	Creación exitosa tras corrección de las	✔

lugar, sin dirección, CP, ni localidad.	invalidaciones notificadas de dirección y localidad obligatorias.	
Modificación de lugar con valores de nombre y provincia en blanco.	Modificación exitosa tras corrección de las invalidaciones notificadas por valores erróneos de código postal: ‘1234’ y ‘1234p’.	✔
Eliminación de lugar que sea parada de alguna ruta, que a su vez tenga asignado algún itinerario.	Eliminación exitosa tras modificar las rutas donde figuraba el lugar excluyéndolo como parada.	✔
Creación de nueva ruta.	Creación exitosa tras corrección de la invalidaciones notificadas por: ninguna parada indicada y parada en blanco seguida de otras sí indicadas.	✔
Modificación de ruta, cambiando el lugar de inicio y eliminando la última parada.	Modificación exitosa. (En esta prueba se llevan a cabo dos sentencias SQL de diferente tipo: *update* y *delete*.)	✔
Modificación de ruta, cambiando la última parada y añadiendo otra.	Modificación exitosa. (En esta prueba se llevan a cabo dos sentencias SQL de diferente tipo: *update* e *insert*.)	✔
Eliminación de ruta que tenga asignado algún itinerario.	Eliminación exitosa tras eliminar los itinerarios donde figuraba la ruta.	✔
Creación de nuevo itinerario sin ruta ni fecha de salida.	Creación exitosa tras corrección de las invalidaciones notificadas de ruta y fecha de salida obligatorias, por valor de observaciones excesivamente extenso y por valores erróneos de fecha de llegada: ’01-01-2010’, ’01/01/10’ y ‘01/01/201p’.	✔
Modificación de itinerario con valor de usuario en blanco	Modificación exitosa tras corrección de las invalidaciones notificadas por valores erróneos de fecha de salida: ‘32/01/2010’, ‘01/13/2010’, ‘31/02/2010’ y día de salida posterior al de llegada.	✔
Eliminación de itinerario.	Eliminación exitosa.	✔

Tabla 7 – Pruebas del Entorno Web

5.2 Entorno Móvil

Las pruebas que concebiremos aquí serán, en su mayoría, aquellas sobre los casos definidos para cada tipo de comunicación, añadiendo tan sólo algunas que no conllevan comunicación como la validación de usuario y el manejo y modificación de la información adquirida por el móvil.

Prueba	Resultado Esperado	Resultado obtenido
Actualización de Login – Caso 1.	Actualización exitosa. Notificaciones detalladas del transcurso de la operación.	✔
Actualización de Login – Caso 2.	Actualización desechada, con búsqueda bluetooth frustrada por servidor bluetooth caído. Notificaciones detalladas del transcurso de la operación.	✔
Actualización de Login – Caso 3.	Actualización exitosa. Notificaciones detalladas del transcurso de la operación.	✔
Actualización de Login – Caso 4.	Actualización desechada, por servidor bluetooth caído y sin búsqueda bluetooth. Notificaciones detalladas del transcurso de la operación.	✔
Acceso de usuario con DNI y contraseña incorrectos.	Validación de usuario exitosa tras notificación de DNI incorrecto, corrección de DNI, notificación de contraseña incorrecta y corrección de contraseña.	✔
Recepción de Itinerarios – Caso 1.	Recepción exitosa. Notificaciones detalladas del transcurso de la operación.	✔
Recepción de Itinerarios – Caso 2.	Recepción desechada, con búsqueda bluetooth frustrada por servidor bluetooth caído. Notificaciones detalladas del transcurso de la operación.	✔
Recepción de Itinerarios – Caso 3.	Recepción exitosa. Notificaciones detalladas del transcurso de la operación.	✔
Recepción de Itinerarios – Caso 4.	Recepción desechada, por servidor bluetooth caído y sin búsqueda bluetooth. Notificaciones detalladas del transcurso de la operación.	✔
Visualización de Itinerario y sus paradas, marcándolo como resuelto y añadiendo Observaciones.	Navegación adecuada por las opciones de cada Itinerario y conservación, tras salida de la aplicación, de los cambios realizados y guardados en ambos campos editables.	✔
Entrega de Itinerarios – Caso 1.	Entrega exitosa. Notificaciones detalladas del transcurso de la operación.	✔
Entrega de Itinerarios – Caso 2.	Entrega desechada, con búsqueda bluetooth frustrada por servidor bluetooth caído. Notificaciones detalladas del transcurso de la operación.	✔
Entrega de Itinerarios – Caso 3.	Entrega exitosa. Notificaciones detalladas del transcurso de la operación.	✔
Entrega de Itinerarios – Caso 4.	Entrega desechada, por servidor bluetooth caído y sin búsqueda bluetooth. Notificaciones detalladas del transcurso de la operación.	✔

Tabla 8 – Pruebas del Entorno Móvil

Capítulo 6

Conclusiones y Trabajos Futuros

6.1 Conclusiones

Lo que cubre el sistema desarrollado por este proyecto es una pequeña y sustancial porción de algo tan importante como la logística.

Algunos definen logística como la administración de inventarios, otros piensan en ella como el transporte de bienes y otros dicen que logística se encarga del sistema de entrega de mercancías. Todos ellos están en lo correcto. En general, logística es un sistema muy amplio de administración de toda la cadena de abastecimiento, desde la materia prima hasta la distribución de los bienes elaborados al consumidor.

Diversos son los autores que coinciden en la importancia que revierte para las organizaciones las funciones logísticas: "*[…] para una empresa que opere en una economía de alto nivel, es vital una buena gestión de las actividades logísticas. Frecuentemente los mercados son de ámbito nacional e internacional, mientras que la producción se puede concentrar en unas pocas zonas. Son los sistemas logísticos los que proporcionan el puente entre las áreas de producción y los mercados, separados en tiempo y distancia […]*" (28); por otra parte, también se plantea que "*[…] La logística contribuye a la competitividad empresarial con la reducción de los costos (reducción de niveles de inventario, minimización de recorridos de transporte de reparto, incremento del aprovechamiento de las capacidades de almacenamiento, etc.) y en el incremento del Nivel del Servicio al Cliente (disminución del ciclo pedido-entrega, adecuada estrategia de canales, disminuir las posibilidades de ruptura de inventario, etc.), es decir, se pueden*

lograr importantes ventajas competitivas a partir de un adecuado diseño y aplicación de la logística en la empresa […]" (29).

Al igual que cualquier empresa en este nuevo mundo y milenio digital, en nuestro país las empresas competitivas serán las que definan y desarrollen su eficiencia sobre la base de la velocidad de respuesta a todas sus necesidades, tanto internas como externas, las que conozcan y dominen su plataforma de información, las que sean capaces de flexibilizar e integrar cada situación (nuevas regulaciones, resoluciones, emisión de información empresarial, emisión de información estadística, entre otras) a este activo sistema de información que constantemente es cambiable a cada momento y con cada decisión. Bill Gates visualiza los sistemas de información de la empresa "*como un sistema nervioso, parecido al del cuerpo humano, donde ante una acción, existe una reacción*", es por ello que el resultado de los sistemas empresariales estará acentuado por el uso de la tecnología digital y sus aplicaciones en este ámbito será el manual de trabajo de la empresa. En definitiva, los cambios tecnológicos vienen a ser el factor clave en la logística.

Con todo esto, parece ridículo el alcance del presente proyecto, y más cuando ya existen soluciones en el mercado que abarcan por completo el campo logístico que pueda desempeñar una empresa (15). Pero aun así, sigo defendiendo la practicidad de mi proyecto, pues ha quedado demostrado que podría ser una herramienta más que útil en ciertas empresas como la ilustrada en el apartado *3.4 Casos de Uso*, y también la versatilidad y la sólida base que puede suponer para proyectos de ampliación a este que abarquen más campos logísticos (en el siguiente punto citaremos algunas ideas al respecto).

Además, he de resaltar que dicha practicidad surge del empleo de las comunicaciones inalámbricas más actuales que se desenvuelven en los dispositivos móviles, más que fuertemente adoptados por la sociedad actual diría que necesarios, a unos precios de desarrollo bastante comedidos (como se puede apreciar en el último anexo, *C.3 Memoria Económica*). Razones de más para que supongan un nuevo mundo de posibilidades en aplicaciones para todos los ámbitos de la vida moderna (ver *2.3.4.4 Usos y aplicaciones* de NFC).

Aun así, es cierto que la tecnología NFC, al menos la parte que me ha tocado trabajar de comunicación entre dispositivos sin intervención de tarjetas (NFCIP), aún debe depurar su funcionamiento, pues nos hemos hallado con problemas en casos de múltiples sucesiones de envío y recepción de bajo contenido, solventados con la reducción de tales secuencias a un solo envío-recepción del contenido sumatorio, tal y como se ha explicado en el apartado *4.3.5 Optimización de las comunicaciones*.

6.2 Trabajos Futuros

Respecto a los trabajos futuros, caben grandes posibilidades de mejora del actual proyecto. Entre ellas, vamos a distinguir mejoras que se pueden llevar a cabo conforme al mismo diseño planteado, es decir, mejoras en la implementación o desarrollo; y mejoras

que quizás requieran una nueva interpretación y puedan extender o plantear cambios en el diseño. Comenzamos con las primeras, que vienen a ser mejoras menores:

- En todo el sistema, se debe especificar la prohibición de los caracteres prohibidos, 'ç' y 'Ç' debido a la *4.3.5 Optimización de las comunicaciones*, perfeccionado con la implementación y aceptación de dichos carcáteres como literales mediante otro carácter de escape.

❖ **Móvil**

- **Seguridad.**
  - **En el Sistema de Acceso de Usuario**. El almacenamiento de la información que mantiene el Sistema de Acceso de usuario se hace en claro, factor clave de debilidad para el mismo y, por consiguiente, para toda la aplicación móvil de usuario. Una mejora sencilla sería almacenar dicha información cifrada mediante algún proceso empleado también en cada acceso, para lo que podríamos ayudarnos, por ejemplo, del API opcional para J2ME de SATSA (*Security and Trust Services API*, JSR 177) (21). Esta mejora es extendible a todo el almacenamiento realizado en el móvil usuario (*4.3.1 Móvil usuario*).
  - **En las comunicaciones inalámbricas**. La mejora anterior se puede ampliar de modo que toda la información que viaja entre el servidor bluettoth y el móvil usuario lo haga encriptada. Para esta mejora sería recomendable adentrarse en los términos de seguridad que aportan ambas tecnologías, bluetooth (30) y NFC, y determinar su rentabilidad.
- **Autoarranque** de MIDlets, sobretodo del **MIDlet intermediario**, con el simple contacto gracias a *PushRegistry* (31).
- **Funcionalidades móviles del usuario sin intermediario**. Esta es una clara mejora, en la que se añadan las funcionalidades de recepción y entrega de itinerarios en la aplicación móvil usuario sin la intervención del punto intermediario ni de NFC, implementando la comunicación entre servidor bluetooth y móvil usuario mediante bluetooth únicamente (caso más simple, descartado según *1.3 Motivación personal*).

❖ **Web**

- **Apariencia**. La apriencia web tampoco ha sido un factor concluyente para el desarrollo, por lo que se sugiere alguna mejora en este sentido, al menos en vía al cumplimiento de las normas de accesibilidad W3C (20). Es probable que dicha mejora esté estrechamente vinculada a una reformulación que optimice el diseño MVC actual.
- **Seguridad.**
  - La única forma de invocar un servlet es a través de un servidor web, lo cual da un alto nivel de seguridad, especialmente si el servidor web está protegido tras un cortafuego (*firewall*), esto significa que el cliente no puede borrar ni modificar nada del propio servidor. Aunque también se pueden usar características nativas de seguridad como el encriptamiento de mensajes.
  - **Sistema de Acceso y Validación de Usuario**. Se puede diseñar un Sistema de Acceso de usuario para la aplicación web, similar al diseñado para la aplicación móvil usuario, definiendo perfiles de usuario para

establecer la funcionalidad en el acceso a la aplicación (lectura, escritura,…); dando así cobertura también a los usuarios transportistas.

- **Funcionalidad de generación de itinerarios automática.** Consistiría en añadir la funcionalidad de generar múltiples itinerarios para un período dado según cierto esquema producto de un análisis que responda a una rutina variable, como el que ya vimos en 3.4.1.1 *Generación y manejo de itinerarios* para la reposición de máquinas expendedoras. A bajo nivel, la implementación de esta mejora sería mediante scripts SQL.
- **Filtros de búsqueda** en las páginas de los **listados**, sobretodo para el de itinerarios por su relevancia y potencialmente abundante contenido, que permita reducir dicho listado a grupos de itinerarios que compartan ciertos criterios de búsqueda.
- **Campos de fechas** mejorados mediante la inclusión de un calendario, tan comunes en el ámbito web, fácilmente desarrollable mediante JavaScript (*2.2.8 HTML y JavaScript*).
- **Ventana de confirmación** para la opción ***Eliminar***, fácilmente implementable también mediante JavaScript.
- Añadir un nuevo criterio de **validación** para el formulario de **ruta**, en el que no se permita que dos paradas seguidas sean en el mismo lugar (no llevado a cabo actualmente por surgir prácticamente al final del proyecto, con el desarrollo más que zanjado).
- Reestructuración del mapeo de los servlets sobre URIs más intuitivas y extendibles.

❖ **Base de Datos**

- **Seguridad**. La seguridad respecto al SI corre a cuenta del gestor MySQL (32).
  - Sería recomendable tras la implantación idear un plan para la realización de **copias de seguridad**, por si es necesaria la recuperación de los datos del sistema en algún momento (33).
- **Ampliación** del actual **Sistema de Información** con nuevas inclusiones en el modelo de datos que puedan resultar útiles, como:
  - Dificultad o peligrosidad de una ruta.
  - Tiempo estimado para una ruta
  - Cargo del usuario (por ejemplo, transportista, distribuidor o reponedor, técnico de mantenimiento, inspector,…).
  - Tipo de ruta (asignables a un cargo u otro).
  - Fecha de última modificación de una ruta.
  - Fecha de alta de un lugar como parada de una ruta.
  - Etc.

Por último, pasamos a una enumeración de posibles ideas conjugables con lo desarrollado y que podrían ser de más o menos provecho según las circunstancias:

- Para el caso de uso descrito, máquinas dispensadoras con capacidad de comunicar al sistema sus niveles de existencias y de emitir alarmas cuando se alcancen ciertos mínimos.

❖ **Móvil**

- Comunicaciones inalámbricas sustituidas por **comunicación móvil (GSM o UMTS)** de largo alcance, para conseguir un sistema en tiempo real; con mayor

flexibilidad en la generación o modificación de itinerarios y en la notificación de sus resultados, pero también de mayor coste e implicaciones de seguridad.
- **Integración** con dispositivos que dispongan de **sistema de navegación por GPS**, que den soporte al desarrollador y se puedan seguir las rutas a través de él.

❖ **Base de Datos**
- **Ampliación el Sistema de Información** en función del fin de la empresa, como puede ser en nuestro caso de uso, con datos característicos sobre las máquinas expendedoras y sus productos y existencias.

Se puede observar que gran parte de las mejoras, no contempladas para el desarrollo actual, surgen a raíz de la seguridad. Este no ha resultado un tema prioritario para el proyecto, por lo que ha quedado en gran medida excluido.

6.3 Limitaciones

En la realización del proyecto nos hemos encontrado con ciertas limitaciones que convienen dar cita:

- Como ya dijimos en *1.5.2.2 Segunda fase del desarrollo*, el simulador de *Nokia* proporcionado por su *S40 Nokia 6212 NFC SDK* no reproduce correctamente la función P2P de NFCIP mediante la aplicación auxiliar *NFC Manager*, con la que se puede indicar la conexión NFC entre dos emulaciones en ejecución pero no se observa ninguna respuesta y tan sólo aparece un mensaje de error de la comunicación en alguno de los dos emuladores tras cerrar el otro. Debido a esta limitación, tampoco se han podido adjuntar capturas de pantalla en la presente memoria para mostrar el efecto de la parte móvil desarrollada.
- La que se considera como mayor limitación del desarrollo, sobretodo por el efecto visual pues respecto al comportamiento sí es el esperado, es que tras acceder a alguna de las funcionalidades de comunicación mediante NFCIP en el MIDlet de usuario (recepción o entrega de itinerarios o actualización de *login*) el cambio de pantalla que debería producirse para representar el transcurso de la comunicación a modo de *log* no ocurre hasta que la propia comunicación NFCIP finaliza y se cierra. Ante esta problemática hemos intentado implementar en diferentes hilos de ejecución (*Threads*) las operaciones que afectan a la pantalla del móvil (*Display*) y las de la comunicación NFCIP, pero no hemos obtenido ningún resultado mejor.
- En el punto *4.2.1 Casos excepcionales en el diseño* se menciona la complejidad que ha supuesto la implementación de servlets como JSPs respecto al manejo de la conexión a BD, y es que según el diseño la conexión a BD la aportaba el servlet padre personalizado del que se hereda en caso de querar disponer de esta, y al tratar de implementar ciertos servlets excepcionales como JSPs no se ha podido heredar o extender de dicho servlet particular. Esto es debido a que los JSPs ya extienden por defecto de la clase servlet del API y, a pesar de existir la declaración *extends*, no se puede redefinir.

6.4 Valoraciones personales

Existen varios puntos sobre los que me gustaría pronunciarme en este apartado.

Debo comenzar diciendo que antes de iniciar el proyecto, cuando acudí a mi tutor, lo que me animó a aventurarme en su realización fue mi interés por ambas plataformas extendidas, J2EE y J2ME, de la plataforma básica de Java, J2SE. Este había nacido durante la carrera donde habíamos dado algunas pinceladas de desarrollo en ambos campos mediante prácticas. Puedo determinar mientras redacto estas líneas que este interés se ha consolidado en entusiasmo, lo cual me convence, pese a los dilemas personales sobre mi continuidad en la carrera a lo largo de los años que me ha ocupado, de que no me confundí de estudios.

Creo preciso además dar las gracias a mi tutor (que me disculpe si no lo he hecho en el apartado de agradecimientos, ya que lo he reservado de un modo más personal) pues con la temática que me propuso me he dado cuenta de la importancia de la logística. Cuando me comentó de qué trataría, sobre la gestión y distribución de planes de rutas, consideré que mi proyecto trataba sobre logística. Sin embargo, en mi proceso abierto de nutrirme de información durante la realización del proyecto, he descubierto que la logística es un campo mucho más extenso de lo que pensaba y, con ello, de vital importancia en el mundo actual en el que nos movemos; y que lo tratado por este proyecto es una pequeña parte de este campo.

Ahora evalúo como una experiencia muy positiva la culminación de este proyecto, pues con él se ha despertado mi ingenio, hasta tal punto de habérseme ocurrido alguna idea que luego informándome he descubierto rápidamente que ya está en vías de desarrollo (para los curiosos se trata de los denominados Sistemas Inteligentes de Transporte, ¡y pensar que no se le hubiera ocurrido a nadie antes!).

También, y procurando no extenderme mucho más, destacar la dificultad a la que me he enfrentado cuando inicias un proyecto del que tienes las ideas del desarrollo a grandes rasgos y prácticamente te supera al no saber por dónde seguir adelante, dándote la sensación de que todo tu trabajo no avanza sino que da vueltas. De esta experiencia individual destaco la importancia que tiene la planificación de los proyectos, el trabajo en grupo y la coordinación colectiva para elaborar proyectos que normalmente serán de mayor envergadura, y los pasos constructivos a seguir en estos casos que se nos han enseñado en nuestros estudios de Ingeniería, aunque ciertamente creo por ahora sólo intuirlos y que terminaré adquiriendolos solventemente con el resto de experiencias profesionales que me estén por llegar.

En esta línea, no me gustaría olvidar tampoco la satisfacción que me ha supuesto el haber lidiado con problemas patentes tras el desarrollo y que suponen que el diseño o el pensar a priori cómo llevar a cabo tus pretensiones no es definitivo hasta que se prueba una vez realizado. Con esto me refiero a los ajustes indicados en la parte *4.3.5 Optimización de las comunicaciones*.

Por último y a conciencia, pues deseo que sea mi puntualización más breve, tras esta buena experiencia y otra anterior de proyecto fin de carrera sin concluirse ajeno a mi

voluntad, intento depurar lo aprendido en todos los sentidos haciendo válido el dicho de que “nadie da duros a cuatro pesetas”.

Glosario

Acrónimos

ACK	*ACKnowledgement* (acuse de recibo)
ANSI	*American National Standards Institute*
API	*Application Programming Interface*
BBDD / BD	*Bases de Datos / Base de Datos*
CDC / CLDC	*Connected (Limited) Device Configuration*
CGI	*Common Gateway Interface*
CP	*Código Postal*
FK	*Foreign Key*
GNU GPL	*GNU General Public License*
GPS	*Global Positioning System*
GSM	*Global System for Mobile Communications / Groupe Special Mobile*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
IDE	*Integrated Development Environment*
IEEE	*Institute of Electrical and Electronics Engineers*
J2EE	*Java 2 Standard Edition*
J2ME	*Java 2 Enterprise Edition*
J2SE	*Java 2 Micro Edition*
JCP	*Java Community Process*
JDBC	*Java DataBase Connectivity*
JDK	*Java Development Kit*
JNDI	*Java Naming and Directory Interface*
JRE	*Java Runtime Environment*

JSP	*JavaServer Pages*
JSR	*Java Specification Request*
JVM / KVM	*Java Virtual Machine / Kilobyte JVM*
LAN	*Local Area Network*
MIDP	*Mobile Information Device Profile*
MVC	*Modelo Vista Controlador*
NFC	*Near Field Communication*
P2P	*Peer to Peer*
PFCJMAG	*Proyecto Fin de Carrera de Juan Manuel Ardoy de Gracia (acrónimo del sistema objeto de esta memoria)*
PK	*Primary Key*
POO	*Programación Orientada a Objetos*
RFID	*Radio Frequency IDentification*
SDK	*Software Development Kit*
SI	*Sistema de Información*
SO	*Sistema Operativo*
SQL	*Structured Query Language*
UML	*Unified Modeling Language*
UMTS	*Universal Mobile Telecommunications System*
URL	*Uniform Resource Locator*
Winsock	*WINdowsSOCKet*
WTK	*Wireless ToolKit*
WTP	*Web Tools Platform*

Referencias

Bibliografía

Ref.1 Páginas o documentos electrónicos en la red

1. **Wikipedia.** Transporte. *Wikipedia, la enciclopedia libre.* [En línea. Último acceso: 11/10/2010 22:00h] http://es.wikipedia.org/wiki/Transporte.
2. **Eclipse.** Eclipse. *Página de Eclipse.* [En línea. Último acceso: 11/10/2010 22:00h] http://www.eclipse.org/.
3. **Forum Nokia.** Nokia 6131 NFC SDK 1.1. *Herramientas para el desarrollo en dispositivos móviles.* [En línea. Último acceso: 11/10/2010 22:00h] http://www.forum.nokia.com/info/sw.nokia.com/id/ef4e1bc9-d220-400c-a41d-b3d56349e984/Nokia_6131_NFC_SDK.html.
4. —. Guía de Instalación de Nokia 6131 NFC SDK 1.1. *Documentación e información de recursos.* [En línea. Último acceso: 11/10/2010 22:00h] http://www.forum.nokia.com/info/sw.nokia.com/id/86dc4e2a-fde2-454f-a513-e8f8a587a0ab/Nokia_6131_NFC_SDK_Installation_Guide_v1_1_en.pdf.html.
5. **Java Sun Microsystem - Oracle.** Descarga de J2SE 5.0 Update 21. *Descarga de productos y tecnologias Java.* [En línea. Último acceso: 11/10/2010 22:00h] http://java.sun.com/products/archive/j2se/5.0_21/index.html.
6. —. Descarga J2EE 5 Update 8, con servidor de aplicaciones GlassFish 2.1.1. *Desacarga de productos y tecnologías Java.* [En línea. Último acceso: 11/10/2010 22:00h]

http://java.sun.com/javaee/downloads/previous_u8/index.jsp?userOsIndex=6&userOsId=windows&userOsName=Windows.
7. **Google Code.** Descarga de la librería BlueCove 2.1.0. *Descargas del proyecto BlueCove.* [En línea. Último acceso: 11/10/2010 22:00h] http://code.google.com/p/bluecove/downloads/detail?name=bluecove-2.1.0.jar&can=2&q=.
8. **MySQL.** Descarga de MySQL 5.1. *Descargas en MySQL.* [En línea. Último acceso: 11/10/2010 22:00h] http://www.mysql.com/downloads/mysql/.
9. —. Descarga de MySQL Connector/J. *Descargas de MySQL.* [En línea. Último acceso: 11/10/2010 22:00h] http://www.mysql.com/downloads/connector/j/.
10. **Forum Nokia.** Series 40 Nokia 6212 NFC SDK. *Herramientas para el desarrollo en dispositivos móviles.* [En línea. Último acceso: 11/10/2010 22:00h] http://www.forum.nokia.com/info/sw.nokia.com/id/5bcaee40-d2b2-4595-b5b5-4833d6a4cda1/S40_Nokia_6212_NFC_SDK.html.
11. —. Guía de instalación de S40 Nokia 6212 SDK. *Documentación e información de recursos.* [En línea. Último acceso: 11/10/2010 22:00h] http://www.forum.nokia.com/info/sw.nokia.com/id/824b4a87-4306-4b06-a31e-58f8dffd0070/S40_Nokia_6212_NFC_SDK_Installation_Guide.html.
12. **Java Sun Microsystem - Oracle.** Descarga de Java SE JDK 6 Update 20. *Descarga de productos y tecnologias Java.* [En línea. Último acceso: 11/10/2010 22:00h] http://www.oracle.com/technetwork/java/archive-139210.html.
13. **GlassFish Community.** Descarga de plugins GlassFish. *Plugins para IDEs de GlassFish.* [En línea. Último acceso: 11/10/2010 22:00h] https://glassfishplugins.dev.java.net/download/.
14. —. Descarga de GlassFish Tools Bundle For Eclipse. *Descarga de herramientas de GlassFish Community.* [En línea. Último acceso: 11/10/2010 22:00h] http://dlc.sun.com.edgesuite.net/glassfish/eclipse/.
15. **Logísitca y Transporte, y otros.** Sistemas Logísticos en el mercado. *Estado del Arte.* [En línea. Último acceso: 11/10/2010 22:00h] http://www.logisticaytransporte.org/; http://www.exagontechnologies.com/logistica.html; http://www.gsit.net/?p=14; http://www.addingsolutions.com/SistemaGestion-ColorBaby.html; http://www.fecsa.net/es/sistemalogistico.asp.
16. **Sun Microsystem.** Página de Sun Microsystem. [En línea. Último acceso: 11/10/2010 22:00h] http://www.oracle.com/us/sun/index.html.
17. **Oracle Corporation.** Página de Oracle Corporation. [En línea. Último acceso: 11/10/2010 22:00h] http://www.oracle.com/index.html.
19. **Osmosis Latina.** Curso Java Web. *Cursos.* [En línea. Último acceso: 11/10/2010 22:00h] http://javaweb.osmosislatina.com/.
20. **W3C.** Página de inicio. *Consorcio World Wide Web.* [En línea. Último acceso: 11/10/2010 22:00h] http://www.w3c.es/.
21. **Sun Microsystem.** SATSA - JSR 177. *Tecnologías J2ME.* [En línea. Último acceso: 11/10/2010 22:00h] http://java.sun.com/products/satsa/.
22. **Wikipedia.** Bluetooth. *Wikipedia, la enciclopedia libre.* [En línea. Último acceso: 11/10/2010 22:00h] http://es.wikipedia.org/wiki/Bluetooth.
23. **Google Code.** BlueCove. *Página de BlueCove.* [En línea. Último acceso: 11/10/2010 22:00h] http://bluecove.org/.
24. **Wikipedia.** Winsock. *Wikipedia, la enciclopedia libre.* [En línea. Último acceso: 11/10/2010 22:00h] http://es.wikipedia.org/wiki/Winsock.

25. **MySQL.** 15.10.2. InnoDB y AUTOCOMMIT. *Documentación MySQL - Manual de Referencia.* [En línea. Último acceso: 11/10/2010 22:00h] http://dev.mysql.com/doc/refman/5.0/es/innodb-and-autocommit.html.
26. —. 15.6.4. Restricciones (constraints) FOREIGN KEY. *Documentación MySQL - Manual de Referencia.* [En línea. Último acceso: 11/10/2010 22:00h] http://dev.mysql.com/doc/refman/5.0/es/innodb-foreign-key-constraints.html.
27. **Wikipedia.** UUID. *Wikipedia, la enciclopedia libre.* [En línea. Último acceso: 11/10/2010 22:00h] http://es.wikipedia.org/wiki/UUID.
30. **García García, Carlos.** Universidad Carlos III de Madrid - Departamento de Ingeniería Telemática. *Artículos publicados - Tutorial.* [En línea. Último acceso: 11/10/2010 22:00h] http://www.it.uc3m.es/cgarcia/articulos/tutorials/bluetooth-cgarcia.pdf.
31. **Forum Nokia.** Ejemplo de detección y lanzamiento de PushRegistry NFC. *Códigos de ejemplo.* [En línea. Último acceso: 11/10/2010 22:00h] http://wiki.forum.nokia.com/index.php/NFC_PushRegistry_launch_and_detect_example.
32. **MySQL.** 5.5. Cuestiones de seguridad general. *Documentación MySQL - Manual de Referencia.* [En línea. Último acceso: 11/10/2010 22:00h] http://dev.mysql.com/doc/refman/5.0/es/security.html.
33. —. 5.8. Prevención de desastres y recuperaciones. *Documentación MySQL - Manual de Referencia.* [En línea. Último acceso: 11/10/2010 22:00h] http://dev.mysql.com/doc/refman/5.0/es/disaster-prevention.html.
34. **Java Sun Microsystem - Oracle.** Requisitos de hardware y software. *Notas de la version de Sun GlassFish Enterprise Server v 2.1.1.* [En línea. Último acceso: 11/10/2010 22:00h] http://docs.sun.com/app/docs/doc/821-1040/abpaj?l=en&n=1&a=view.
35. **MySQL.** 2.3.5. Utilización del asistente de configuración. *Documentación MySQL - Manual de Referencia.* [En línea. Último acceso: 11/10/2010 22:00h] http://dev.mysql.com/doc/refman/5.0/es/windows-config-wizard.html.
36. —. 11.3.1. Los tipos de datos DATETIME, DATE y TIMESTAMP. *Documentación MySQL - Manual de Referencia.* [En línea. Último acceso: 11/10/2010 22:00h] http://dev.mysql.com/doc/refman/5.0/es/datetime.html.
37. —. 12.5. Funciones de fecha y hora. *Documentación MySQL - Manual de Referencia.* [En línea. Último acceso: 11/10/2010 22:00h] http://dev.mysql.com/doc/refman/5.0/es/date-and-time-functions.html.
38. —. 12.2. Funciones de control de flujo. *Documentación MySQL - Manual de Referencia.* [En línea. Último acceso: 11/10/2010 22:00h] http://dev.mysql.com/doc/refman/5.0/es/control-flow-functions.html.
39. —. 13.2.7.1. Sintaxis de JOIN. *Documentación MySQL - Manual de Referencia.* [En línea. Último acceso: 11/10/2010 22:00h] http://dev.mysql.com/doc/refman/5.0/es/join.html.
40. **COITT.** Acciones del grupo de Gestión del Conocimiento del Colegio para defender los intereses de nuestros profesionales y la calidad de los servicios a los usuarios. *Noticias.* [En línea. Último acceso: 11/10/2010 22:00h] http://www.coitt.es/index.php?page=noticias_coitt_reg&ireg=10&icod=181.
41. **Forum Nokia.** Carbide.j 1.5. *Herramientas para el desarrollo en dispositivos móviles.* [En línea. Último acceso: 11/10/2010 22:00h] http://www.forum.nokia.com/info/sw.nokia.com/id/bc2785aa-bda0-436a-80d6-e6cf4157416a.html.

Ref.2 Libros

18. **Alur, Deepak.** *Core J2EE patterns: best practices and design strategies.* s.l. : Prentice Hall PTR, 2001. 0130648841.
28. **Ballou, Ronald H.** *Logística empresarial : control y planificación.* s.l. : Díaz de Santos, 1991. 8487189687.
29. **Gemeil, Torres, Daduna y Cabrera, Mederos.** *Fundamentos Generales de la Logística.* 2007.

Anexo A

Implantación del Sistema

A.1 Requisitos del Sistema

En este proceso, tendremos en cuenta la parte del entorno de desarrollo únicamente necesaria para la ejecución del sistema en lo que al equipo servidor se refiere, el cual supone la mayor parte del proceso de implantación pues desde este instauraremos también las aplicaciones móviles en los correspondientes dispositivos. De este modo nos quedamos, de la segunda etapa, con los siguientes requisitos software:

- *Java SE JDK 6 Update 20* (12).
- *Java EE 5 SDK Update 8*, con *GlassFish Enterprise Server v2.1.1* (6).
- *MySQL 5.1.45* (8).

En principio no existe ningún invonveniente en ampliar cada herramienta en versiones, si acaso la aplicación web es la única que está orientada para la versión de GlassFish 2.1[15].

A partir de lo anterior, sólo queda determinar los requisitos hardware del servidor como los más restrictivos de los impuestos por estas herramientas. Así, es *J2EE 5* el que

[15] En la opción de Eclipse para la exportación del proyecto web a un fichero WAR (*2.2.6 WAR* para mayor información) se ha elegido un *runtime* específico donde se presupone que se empleará la aplicación, gracias a la opción de *optimizar la exportación para el runtime del servidor específico GlassFish v2.1 Java EE 5*.

determina sobre los demás, necesitando de *J2SE 5.0* ó *Java SE 6* y de *MySQL 5.0* mínimo. (Todo el detalle de requisitos en (34).)

En conclusión, tenemos los siguientes requisitos para el sistema PFCJMAG, tanto hardware como software:

❖ **Hardware**

- **Dispositivos móviles** con tecnologías bluetooth y NFC, y compatible con *MIDP 2.0* de Java.
- **Equipo servidor**, con las siguientes especificaciones:
  - Dispositivo bluetooth, integrado o externo (USB), compatible con los controladores específico de BlueCove (detallados en *2.3.3.5 BlueCove* y enumerados en el siguiente punto de requisitos software).
  - Mínimo (con SO Windows):
    - Memoria (RAM): 1 GB.
    - Espacio en disco: 500 MB de espacio libre.
  - Recomendado (con SO Windows):
    - Memoria (RAM): 2 GB.
    - Espacio en disco: 1 GB de espacio libre.

❖ **Software**

- Sistema Operativo del equipo servidor:
  - Windows.
  - Solaris.
  - Red Hat Enterprise Linux.
  - SUSE Linux.
  - Ubuntu Linux.
  - AIX.
  - Macintosh OS.
- El software adjunto en el CD de entrega e implantación, en el que se incluye el citado al comienzo del apartado y el software producto del desarrollo.
- Controlador del dispositivo bluetooth que empleemos en el equipo servidor correspondiente a:
  - Mac OS X.
  - WIDCOMM.
  - BlueSoleil.
  - Pila Bluetooth de Microsoft (Winsock)[16] (24), a partir de Windows XP SP2.

[16] Anotamos de nuevo que este es el controlador compatible empleado por el dispositivo bluetooth del portátil para el desarrollo.

A.2 Proceso de Implantación

A.2.1 Entorno Web

La guía para la instalación y configuración del servidor web y de la Base de Datos del sistema en el equipo correspondiente consta de los siguientes pasos:

1. Instalación del JDK 6 (Update 20), plataforma de J2SE, mediante su instalador (12).

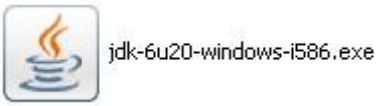


Figura 44 – Instalador JDK 6

2. Instalación de Java EE 5 SDK (Update 8), plataforma de J2EE y que incluye el servidor de aplicaciones *GlassFish Enterprise Server v2.1.1*, mediante su instalador (6). En la solicitud de contraseña para el administrador del servidor (*admin*) indicamos *pfcjmag2*, la cual emplearemos al acceder en su consola de administración más adelante.

Figura 45 – Instalador J2EE 5

3. Instalación de MySQL 5.1.45, gestor de BBDD, mediante su instalador (8).

Figura 46 – Instalador MySQL

Tras esta instalación, procederemos de continuo con la configuración del gestor de BBDD mediante el *Asistente para la Configuración del Servidor MySQL*, indicando las siguientes opciones en el mismo orden en el que aparecen a través de las ventanas del asistente (35):

- *Detailed Configuration*.
- *Server Machine*.
- *Transactional Database Only*, dado que el sistema de almacenamiento por defecto del gestor será *InnoDB*.
- En principio, dejamos la configuración por defecto: *Installation Path*.
- En principio, *Decision Support (DSS)/OLAP* sería adecuado, que promedia en 20 las conexiones simultáneas y establece un máximo de 100.

- En principio, dejamos la configuración por defecto, con ambas opciones activadas.
- En principio, *Standard Character Set* resulta apropiado.
- Dejamos las opciones por defecto, activando también la segunda opción de *Incluir el Directorio de Ejecutables en la variable PATH de Windows*, así podremos ejecutar más fácilmente los comandos del siguiente paso desde el directorio que contenga los scripts SQL.
- Para el usuario por defecto, *root*, indicamos la contraseña: *pfc* (importante para la ejecución de scripts SQL en el siguiente paso y para la configuración posterior del conjunto de conexiones).
- Por último, ejecutamos la configuración y finalizamos.

4. Creación de la Base de Datos del sistema. En este paso, vamos a dar dos opciones en las que se debe ejecutar el script[17] tal y como se muestra, ubicándonos en el intérprete de comandos en OS/2 (cmd.exe) en el directorio donde se encuentre, e indicando la contraseña configurada para el usuario *root*, *pfc*:

a. Para el inicio del SI en blanco, es decir, sin ningún dato inicial:
   - `mysql -u root -p < create_EnBlanco.sql`
     `Enter password: ***`

b. Para el inicio del SI con los datos de ejemplo del *3.4 Caso de Uso*:
   - `mysql -u root -p < create_CasoDeUso.sql`
     `Enter password: ***`

5. Configuración del conjunto de conexiones en el servidor web de aplicaciones. Este paso de la implantación lo vamos a mostrar detalladamente mediante capturas de pantalla.

a. Para este paso es necesario activar o poner en marcha el servidor GlassFish, para lo cual ejecutamos *Start Default Server* desde el menú *Inicio*.

[17] Ambos scripts fueron generados mediante la herramienta *mysqldump* para generar copias de seguridad de las BBDD de MySQL. Como tales, se pueden ejecutar para reiniciar el estado de la BD del sistema o, lo que es lo mismo, eliminar toda información vigente y restablecerla en la inicial, sin ninguna otra ayuda. No obstante, el script resultante del diseño del modelo de datos es *create.sql*, el cual tiene un efecto idéntico a *create_EnBlanco.sql* pero sólo es efectivo si no existe, o se ha eliminado previamente, la Base de Datos *pfcjmag_bd* y, por consiguiente, todo su contenido.

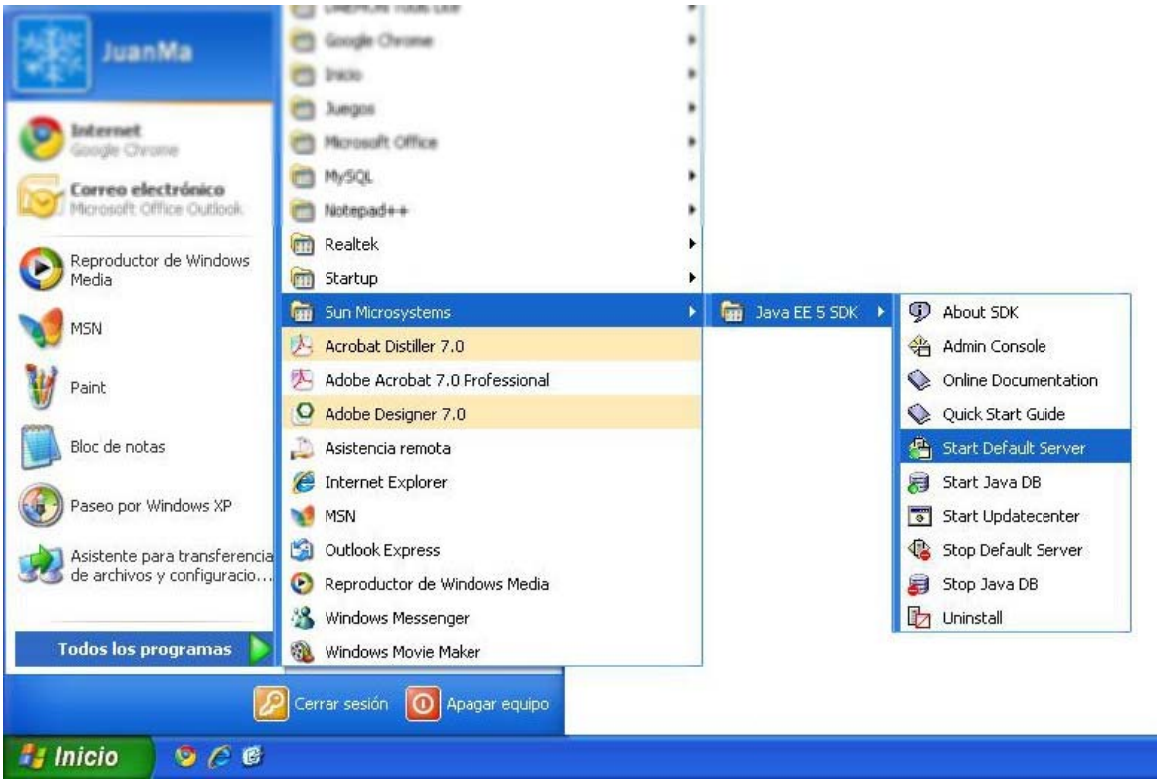


Figura 47 – Arranque servidor web

b. Cuando haya finalizado la activación del servidor, lo siguiente es acceder a la *Consola de Administración de GlassFish*.

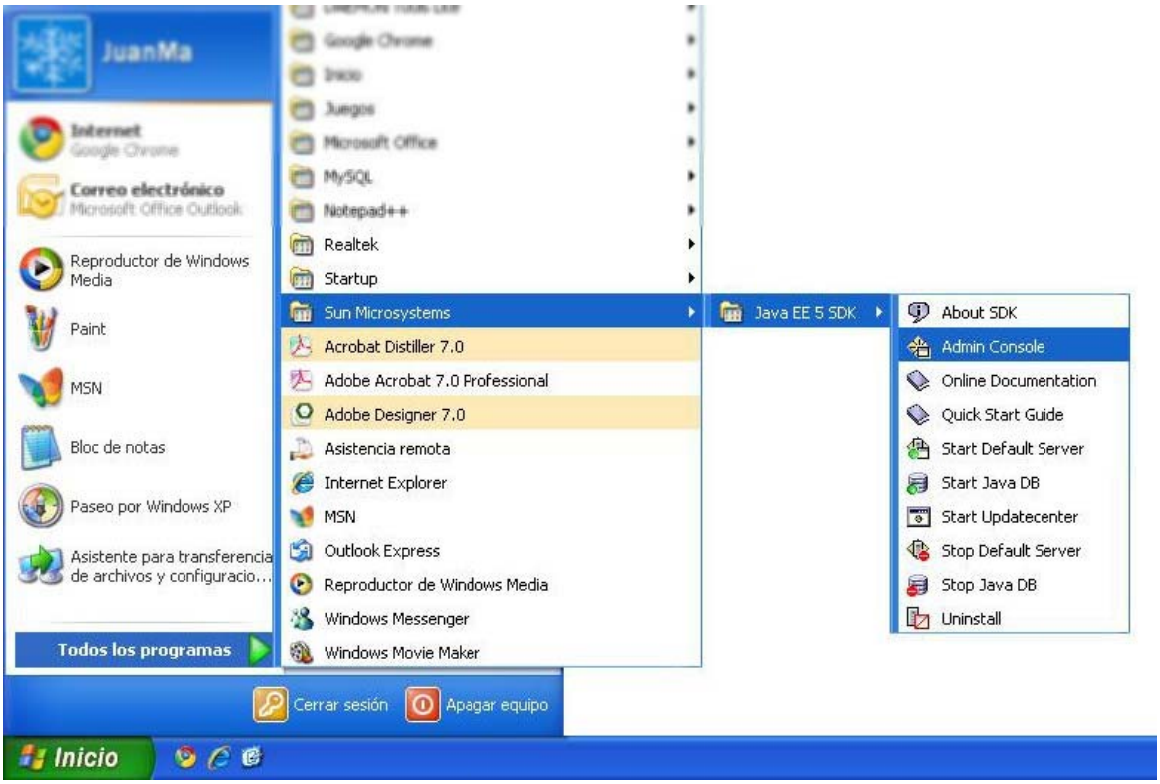


Figura 48 – Ejecución de la consola de administración del servidor web

c. Dicha consola se visualiza mediante el navegador web y lo primero en aparecer es la página de *Login*, en la cual nos autenticamos con los credenciales indicados en la instalación del servidor (*admin* / *pfcjmag2*).

Figura 49 – Acceso a la consola de administración del servidor web

d. A continuación se accederá a la página de inicio de la consola de administración. A lo largo de toda la navegación por esta consola, figurará un panel en forma de árbol a la izquierda con los apartados de los que se disponen. Nuestro siguiente paso será acceder mediante este a *Resources* → *JDBC* → *Connection Pools*.

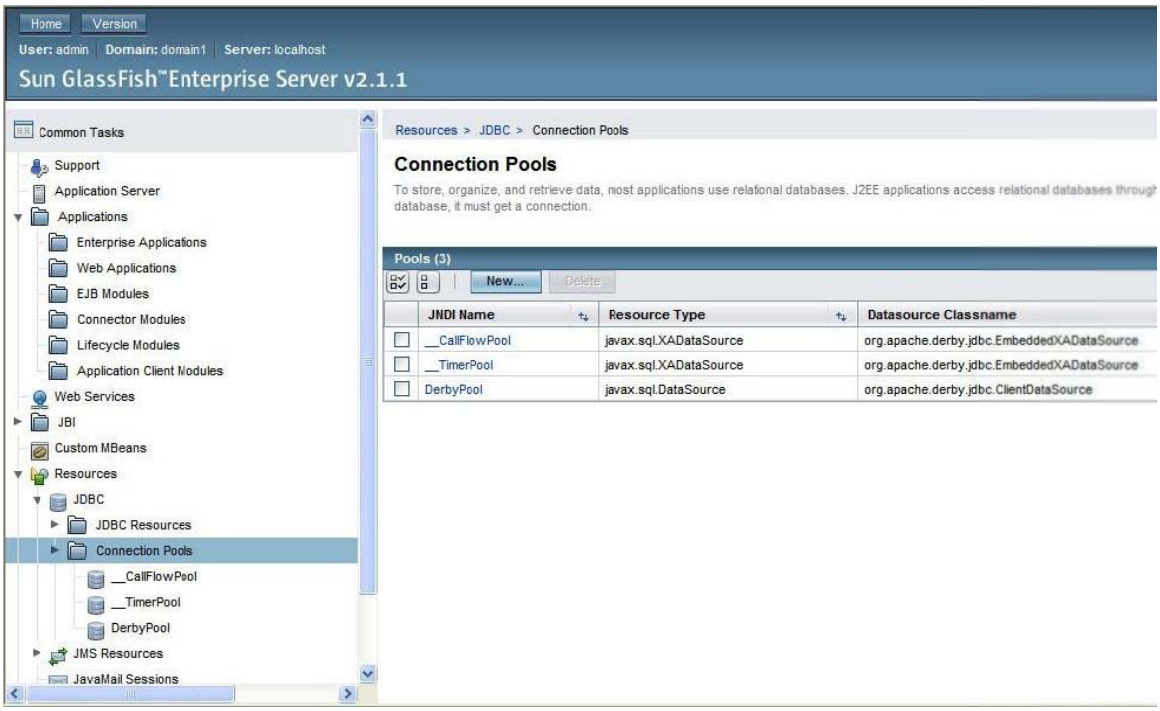


Figura 50 – Vista de *Connection Pools* en la consola de administración de GlassFish

e. Allí crearemos un nuevo *Connection Pool* mediante el botón *New*. En el primer paso, indicaremos los campos requeridos tal y como se muestra:
   - *Name*: PFCJMAGPool
   - *Resource Type*: javax.sql.DataSource
   - *Database Vendor*: MySQL

Figura 51 – Creacion del Conjunto de Conexiones (1 de 2)

f. En el siguiente paso de la creación únicamente deberemos indicar al final de la página, en la tabla *Additional Properties*, los siguientes valores, tras lo cual Finalizamos.
   - *databaseName*: pfcjmag_bd
   - *password*: pfc
   - *portNumber*: 3306.
   - *serverName*: localhost

- *user*: root (indicado por defecto en el punto 3 de este apartado, al igual que *portNumber*).

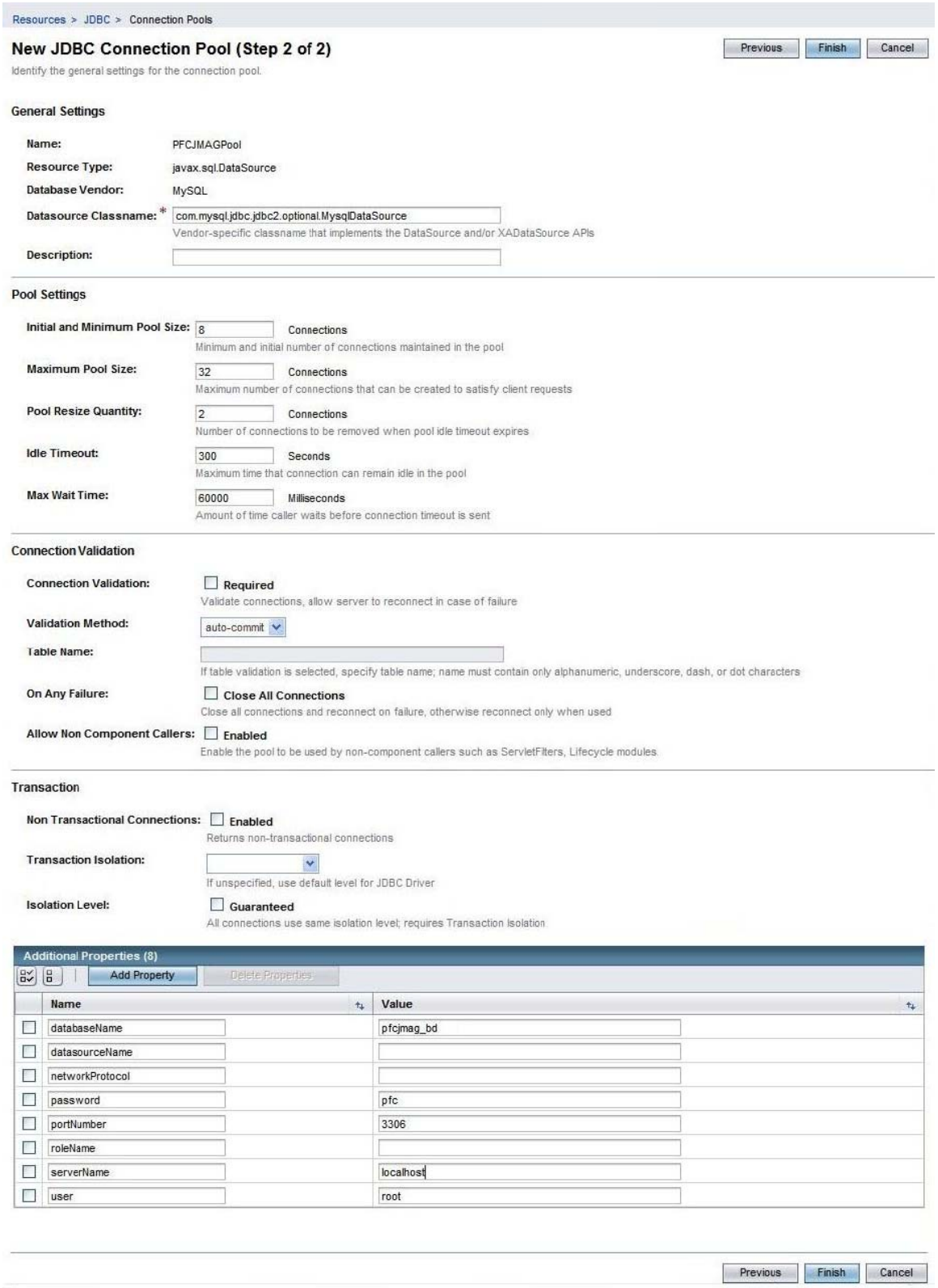


Figura 52 – Creacion del Conjunto de Conexiones (2 de 2)

g. En la lista de *Conection Pools*, donde nos trae de vuelta la aplicación, podemos observar nuestro nuevo conjunto de conexiones. Ahora debemos irnos a la sección *Application Server* → *JVM Settings* → *Path Settings*, para indicar en el campo *System Classpath* la ruta o *path* de la librería JDBC accesible por el equipo servidor donde se encuentre alojada permanentemente, por ejemplo, D:\lib_PFCJMAG\mysql-connector-java-5.1.12-bin.jar. No nos olvidamos de *Salvar*.

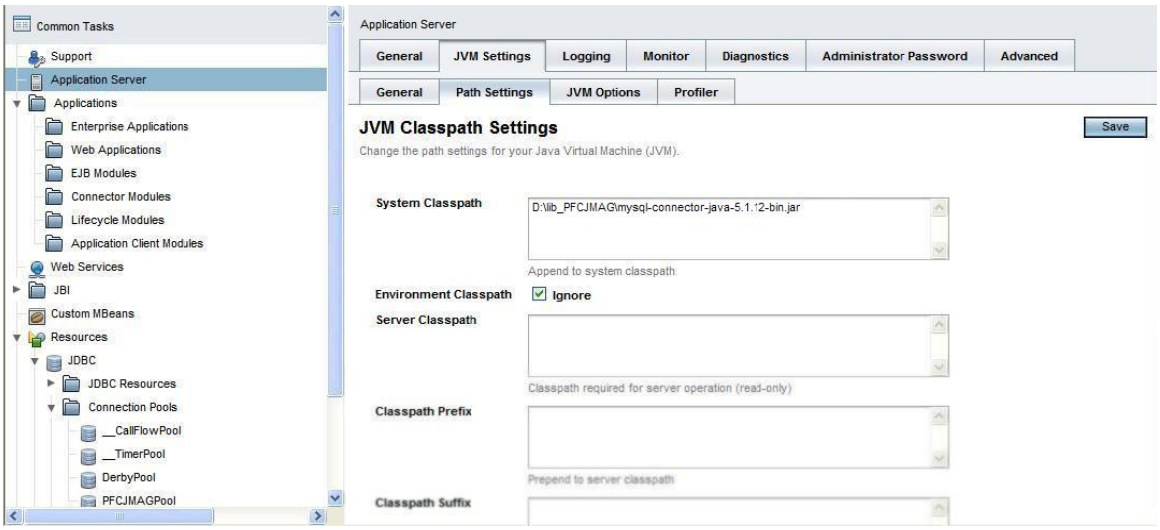


Figura 53 – Configuración del *Classpath* en GlassFish

h. Para que toda la configuración llevada a cabo hasta ahora surta efecto se debe reiniciar el servidor, es decir, ir a la opción *Stop Default Server* y, seguidamente, a *Start Default Server*, ambas en el mismo lugar del menú *Inicio*. A pesar de no haber finalizado aún la configuración por completo, es conveniente reiniciar el servidor para comprobar la correcta configuración hasta el momento. Con este propósito, otra vez en la consola de administración, nos adentramos en la página de edición de nuestro conjunto de conexiones creado anteriormente, pulsando sobre el nombre *PFCJMAGPool*, para finalmente pulsar sobre el botón *Ping*. Si el mensaje resultante de esta acción es *Ping Succeeded*, la configuración transcurre según lo previsto y podemos continuar; si, por el contrario, el mensaje es *An error has occurred*, ha existido algún problema, por lo que no debemos continuar hasta que revisemos todo lo configurado anteriormente y esta prueba sea exitosa.

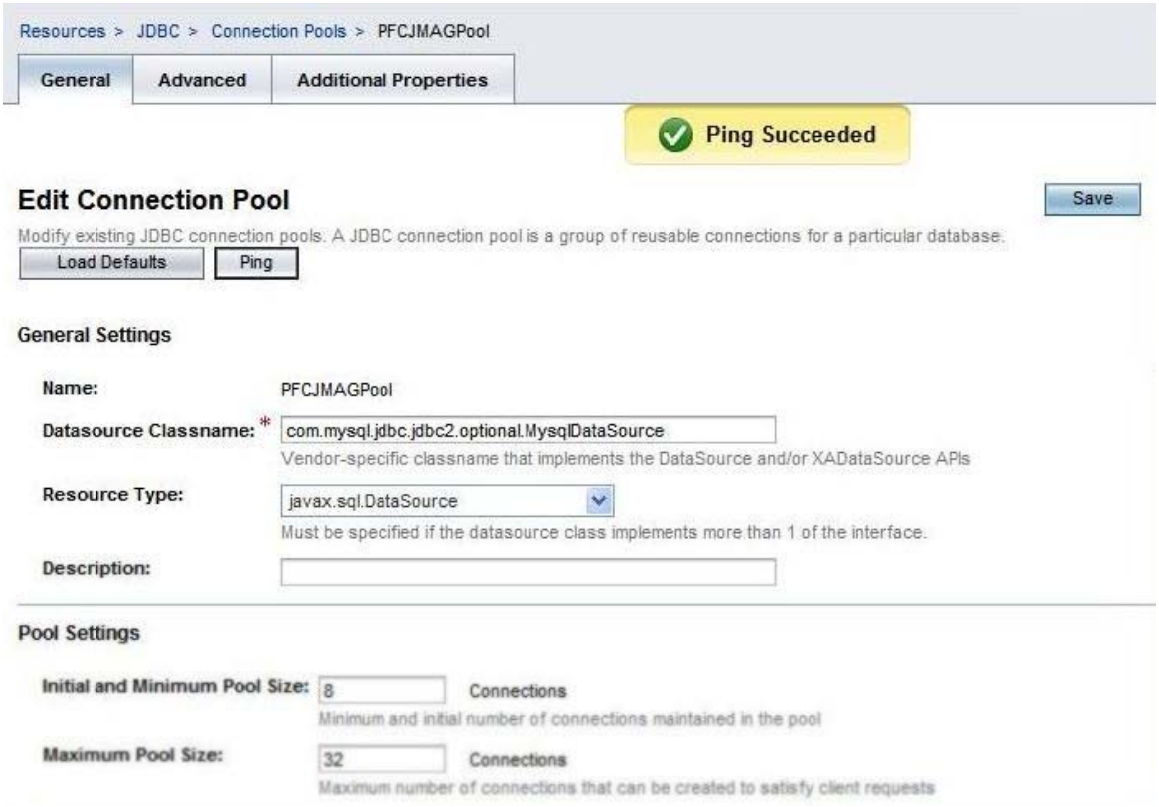


Figura 54 – Mensaje exitoso del *Ping* realizado mediante nuestro Conjunto de Conexiones

i. Por último, debemos crear un recurso JDBC accesible mediante JNDI para poder emplear nuestro conjunto de conexiones desde nuestra aplicación Java. Para ello nos dirigimos a *Resources* → *JDBC* →*JDBC Resources*.

Figura 55 – Vista de *JDBC Resources* en la consola de administración de GlassFish

j. Allí pulsamos sobre *New*, obteniendo el debido formulario a rellenar con la siguiente información:
   - *JNDI Name*: jdbc/PFCJMAGPoolResource
   - *Pool Name*: PFCJMAGPool

Figura 56 – Creacion del recurso JDBC

k. Finalmente, en la lista de *JDBC Resources*, donde nos trae de vuelta la aplicación, podemos observar nuestro nuevo Recurso JDBC vinculado a nuestro conjunto de conexiones. Con esto se da por concluida la configuración del servidor GlassFish para el correcto funcionamiento de la aplicación web del proyecto.

6. Por último, debemos desplegar la aplicación web en el servidor GlassFish. Para ello, desde la misma consola de administración, tendremos que acceder a *Applications* → *Web Applications* para pulsar sobre *Deploy*.

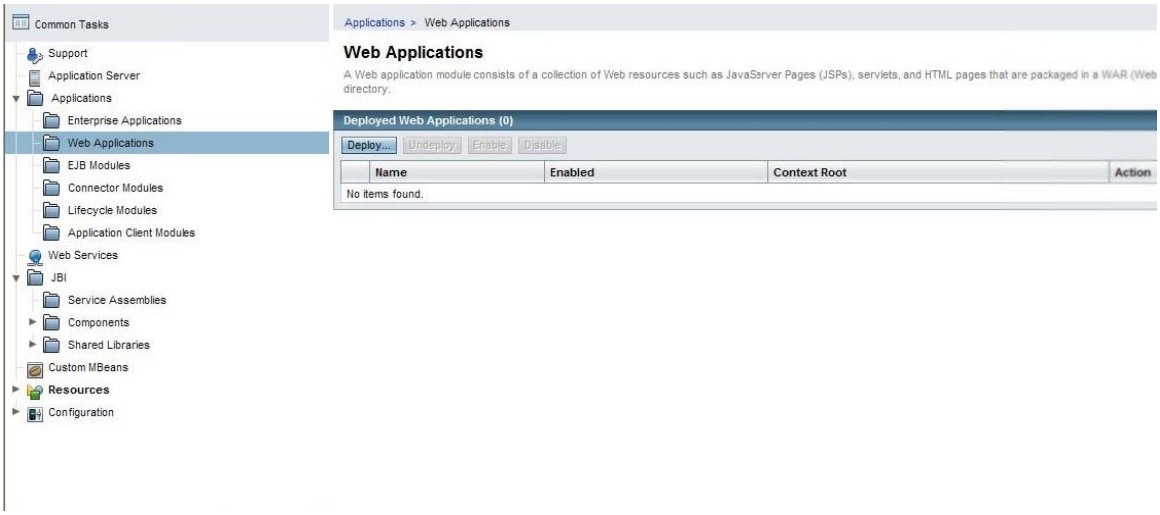


Figura 57 – Vista de *Web Applications* en la consola de administración de GlassFish

Entonces se nos presentará un formulario donde únicamente tendremos que facilitar el fichero .WAR de nuestra aplicación web, para lo que tendremos dos opciones:

- Indicar la ubicación temporal de dicho fichero, de modo que el servidor GlassFish lo conserve.
- Señalar la ubicación del fichero guardado en una unidad de almacenamiento accesible por nuestro equipo servidor (junto al resto de archivos para la implantación y funcionamiento del sistema, copiados desde el CD de entrega e implantación).

Escogemos la primera opción, *Packaged file to be uploaded to the server*, para evitar posibles problemas futuros en caso de cambiar o eliminar el fichero .WAR de implantación, y confirmaremos finalmente.

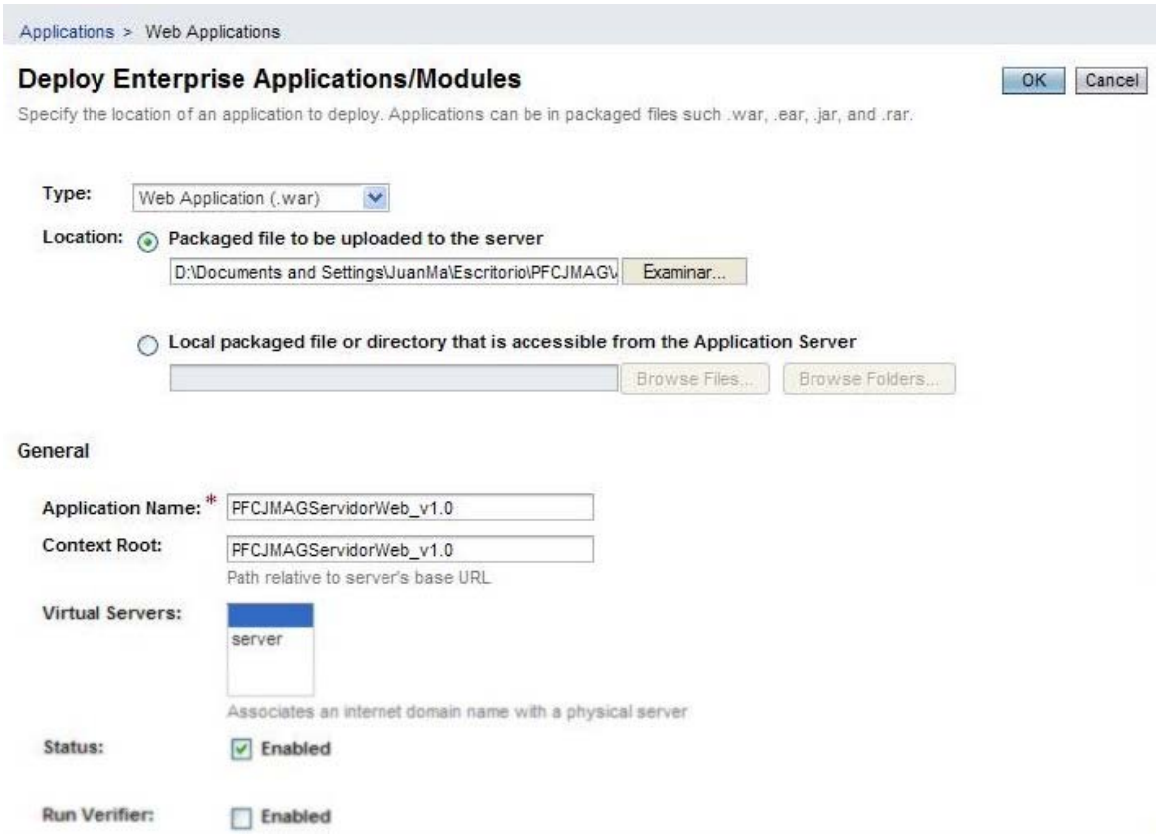


Figura 58 – Despliegue de la aplicación web del sistema en el servidor GlassFish

Si queremos comprobar la correcta implantación de la aplicación web, simplemente con pulsar sobre *Launch* en la misma línea donde ha aparecido nuestra aplicación web recién alojada en el servidor y que se aprecia en la página de *Web Applications* a la que se nos trae de vuelta, o introduciendo en la barra de direcciones del navegador del equipo servidor la dirección http://localhost:8080/PFCJMAGServidorWeb_v1.0/, podremos acceder y navegar por la aplicación web del proyecto.

A.2.2 Entorno Móvil

La implantación del sistema en lo que respecta a los dispositivos móviles trata únicamente de la transferencia de los paquetes instalables de la aplicación móvil (*MIDlet Suite*) a estos. Una vez allí, la instalación y el acceso a las aplicaciones móvil es bastante intuitivo.

En nuestro caso, emplearemos para esta transferencia la propia comunicación bluetooth, entre ordenador y móvil, mediante la aplicación provista por Windows (en la instalación de su controlador) para el manejo del dispositivo bluetooth del ordenador portátil empleado para el desarrollo.

A.3 Activación del Servidor del Sistema

Para la activación del servidor del sistema son necesarios simplemente dos pasos, cada cual para los servicios en un entorno:

- Arranque del servidor web mediante la opción *Start Default Server* en el menú Inicio, tal y como ya se hizo en la implantación.
- Puesta en marcha del servidor bluetooth mediante la ejecución del fichero *ServidorBT_up.exe*, facilitado en el CD de entrega e implantación[18].

[18] Además del fichero .jar del intérprete Java para la aplicación del servidor bluetooth, se adjunta también un ejecutable exclusivamente para Windows. Se presupone muy probable la implantación sobre dicho SO, dada además la experiencia de BlueCove sobre Winsock, por lo que se incluye dicho ejecutable específico para Windows, sin descartar tampoco el uso del fichero .jar para otros SOs.

Anexo B

Operaciones con Base de Datos

B.1 Tratamiento de Fechas

Este es un tema en el que se ha de ser muy cautos. Se debe tomar alguna postura concreta en el manejo de los campos de tipo fecha para eliminar confusiones en la interpretación de dicho campo por parte del gestor MySQL y de los usuarios finales.

El formato elegido para los valores de fecha en toda el área de nuestra aplicación (web y móvil) es *dd/mm/aaaa* (definido como variable final o constante). Sin embargo, el gestor MySQL a la hora de mostrar un resultado de tipo fecha (*date*) lo hace con la forma *aaaa-mm-dd*, aparte del tratamiento interno que MySQL define en su manual para este tipo (36).

Con estas premisas, se ha decidido que el punto donde se controlará dicha interpretación de formato en un sentido u otro sea en las sentencias de transacción SQL. Es decir, el tratamiento de fechas se reduce a:

- Si la fecha la tomamos de la misma BD para mostrarla al usuario en cualquier parte del aplicativo, emplearemos en la sentencia SELECT como campo a obtener la siguiente expresión que aprovecha una función definida por el gestor MySQL (37):
  ```
  date_format (<campoDateEnBD>, get_format(DATE,'EUR'))
  ```
- Si, a la inversa, tomamos una fecha de alguna parte de nuestro aplicativo con el formato europeo, que como dijimos hemos escogido, para que la interprete el

gestor MySQL en alguna de las correspondientes sentencias INSERT o UPDATE, emplearemos la siguiente expresión que también se beneficia de una función definida por el gestor de BBDD (37):

```
str_to_date ('<fechaByUsuario>', '%d/%m/%Y')
```

B.2 Vista alternativa de campos *Nombre*

Antes de comenzar el repaso de operaciones con BD, vamos a detenernos en los campos *Nombre* consultados. Se debe tener en cuenta que dichos campos se han diseñado como posiblemente nulos en todos los casos y, por ello, cuando en una consulta se trata de obtener un nombre hay que contemplar el caso de que no tenga valor. Así, ideamos para cada elemento del modelo de datos que incorpora uno de estos campos *Nombre* una vista alternativa si no consta ningún valor al consultarlo.

Gracias a la función de MySQL *ifnull* (38) podemos obtener el propio campo en caso de no ser nulo u otro alternativo en caso de serlo. La vista alternativa de los campos *Nombre* empleada en cualquier parte del aplicativo (web y móvil) es en cada caso:

- **Usuario**. La vista principal de un usuario esta compuesta por nombre, apellidos y dni; si el nombre resulta nulo se excluye de esta vista, al igual que ocurre con los apellidos, pudiendo figurar la vista únicamente con el DNI si ambos valores son nulos.
- **Lugar**. Un lugar que prescinde de nombre se define mediante la composición de dirección, CP, localidad y provincia, mostrado este último entre paréntesis; a esta vista se le suprime el CP y la provincia si, en cada caso, resultan nulos.
- **Ruta**. Una ruta sin nombre se visualiza mediante la composición del literal “ID” seguido de su identificador o clave primaria *IDRuta*.

A continuación, se podrá apreciar todo lo descrito en estos primeros apartados del anexo.

B.3 Sentencias SQL en el Entorno Web

En esta parte, vamos a reproducir todas las transacciones SQL que se realizan en la BD desde la aplicación web. Se muestran las sentencias tal y como figuran en el código, con los caracteres de escape pertinentes en el caso de querer que aparezcan los signos de paréntesis como literales de un campo consultado, es decir, duplicados para “escapar” del intérprete Java y del intérprete del gestor de MySQL. Procedemos por apartados:

❖ **Usuarios**

- Consulta para la lista.

```
select IDusuario, Nombre, Apellidos, DNI from pfc_usuario
order by Apellidos, Nombre
```

- Consulta para la carga del contenido del formulario en la edición.
```
select IDusuario, Nombre, Apellidos, DNI from pfc_usuario
where IDusuario = _
```
- Consulta para la validación del campo Contraseña Actual en el cambio de contraseña.
```
select Password from pfc_usuario where IDusuario = _
```
- Modificación.
```
update pfc_usuario set DNI = '_', Nombre = '_'/null,
Apellidos = '_'/null[, Password = '_'] where IDusuario =
_
```
- Creación.
```
insert pfc_usuario (DNI, [Nombre,] [Apellidos,] Password)
value ('_', ['_',] ['_',] '_')
```
- Eliminación.
```
delete from pfc_usuario where IDusuario = _
```

❖ **Lugares**

- Consulta para la lista.
```
select * from pfc_lugar order by Localidad, Direccion
```
- Consulta para la carga del contenido del formulario en la edición.
```
select * from pfc_lugar where IDlugar = _
```
- Modificación.
```
update pfc_lugar set Direccion = '_', Nombre = '_'/null,
CP = '_'/null, Localidad = '_', Provincia = '_'/null
where IDlugar = _
```
- Creación.
```
insert pfc_lugar (Direccion, [Nombre,] [CP,] Localidad[,
Provincia]) value ('_', ['_',] ['_',] '_'[, '_'])
```
- Eliminación.
```
delete from pfc_lugar where IDlugar = _
```

❖ **Rutas**

- Consulta para la creación de la tabla que contiene la lista (debemos definir tantas columnas como el número de paradas de la ruta con más paradas).
```
select count(*) as maxNumParadas from pfc_parada group by
IDruta order by maxNumParadas desc limit 1
```
- Consulta para la lista. Aparecerán por orden alfabético del campo consultado ruta, que es uno de los casos que implementa una vista alternativa de las descritas en anterior apartado. Es de vital importancia que se ordenen también los resultados, para paradas de una misma ruta, por *IDparada* para mostrar la secuencia de la ruta correctamente.
```
select p.IDruta, ifnull(r.nombre, concat('ID ',
p.IDruta)) as ruta, ifnull (l.nombre, concat(direccion,
ifnull(concat(' CP.', cp),''), ' ', localidad,
ifnull(concat(' \\(', provincia, '\\)'),''))) as parada
from pfc_parada p, pfc_lugar l, pfc_ruta r where
p.idlugar = l.idlugar and p.idruta = r.idruta order by
ruta, idparada
```

- Consulta de lugares para la carga del contenido del control desplegable Parada en el formulario.

```
select IDlugar, ifnull(Nombre, concat(direccion,
ifnull(concat(' CP.', cp),''), ' ', localidad,
ifnull(concat(' \\(', provincia, '\\)'),''))) as lugar
from pfc_lugar order by lugar
```

- Consulta para la carga del contenido del formulario en la edición. De nuevo, es crucial la ordenación por *IDparada* para la construcción apropiada de la ruta.

```
select r.IDruta, r.nombre, IDParada, IDLugar from
pfc_parada p, pfc_ruta r where p.IDRuta = _ and p.IDRuta
= r.IDRuta order by IDParada
```

- Modificación. Esta operación se aborda mediante más de una posible transacción SQL:

  1º) Tratamos la modificación en el nombre.

```
update pfc_ruta set Nombre = '_'/null where IDRuta = _
```

  2º) Iremos recorriendo las paradas en orden incremental. Gracias a que incluimos el *IDparada* en el nombre del control de cada parada cargada con algún lugar, sabremos si:

  - Actualizar, si consta el *IDparada* en el nombre del control y figura algún lugar.

```
update pfc_parada set IDLugar = _ where IDParada = _
```

  - Eliminar, si consta el *IDparada* en el nombre del control y se ha dejado en blanco.

```
delete from pfc_parada where IDParada = _
```

  - Insertar, si no consta ningún *IDparada* en el nombre del control y se ha seleccionado un lugar como nueva parada.

```
insert pfc_parada (IDRuta, IDLugar) value (_, _)
```

  3º) Por último, concluirá la operación de Modificación con el primer campo de parada en cuyo nombre no conste ningún *IDparada* y tampoco se haya seleccionado ningún lugar (suponemos pasada la validación, *4.2.2 Validación de formularios*).

- Creación. Esta operación también consta de más de una transacción SQL:

  1º) Damos de alta la ruta, incluyendo el nombre si se ha especificado. Obtenemos el *IDruta* resultante de esta inserción para las transacciones del siguiente paso.

```
insert pfc_ruta ([Nombre]) value (['_'])
```

  2º) Y damos de alta tantas paradas como se hayan indicado.

```
insert pfc_parada (IDruta, IDlugar) value (_, _)
```

- Eliminación. En esta operación, dado su modo de funcionamiento que pasamos a exponer, haremos excepcionalmente una consulta para comprobar que no existen itinerarios para la ruta, pues de no ser así podríamos eliminar todas las paradas de la ruta para finalmente, en la eliminación definitiva de la ruta, obtener un error del gestor MySQL por violación de restricción por FK.

  1º) Consulta para la comprobación de restricción por FK en itinerarios de la ruta.

```
select IDRuta from pfc_itinerario where IDRuta = _
```

  2º) Si la anterior consulta devuelve algún resultado, se informa mediante mensaje de la imposibilidad de eliminar la ruta, sin haber modificado dato alguno; si por el contrario, devuelve un resultado nulo, podemos proceder a la eliminación de la ruta mediante las siguientes transacciones:

- Eliminación de las paradas correspondientes a la ruta.

```
delete from pfc_parada where IDRuta = _
```

- Eliminación definitiva de la ruta.

```
delete from pfc_ruta where IDRuta = _
```

❖ **Itinerarios**

- Consulta para la lista. Aparecerán por orden alfabético del campo consultado ruta, que es uno de los casos que implementa una vista alternativa; para los itinerarios de una misma ruta la fecha de salida será el siguiente criterio de aparición, comenzando por el más obsoleto en adelante. Notese en esta consulta el empleo de *LEFT JOIN*, en vez del condicionamiento en la cláusula *where*, dado el posible caso nulo de la FK *IDUsuario* (39).

```
select IDItinerario, ifnull(r.nombre, concat('ID ',
i.IDRuta)) as ruta, u.Nombre, Apellidos, DNI,
date_format(FechaSalida, get_format(DATE,'EUR')) as
FcSalida, date_format(FechaLlegada,
get_format(DATE,'EUR')) as FcLlegada, Observaciones,
Resuelto from pfc_ruta r, pfc_itinerario i LEFT JOIN
pfc_usuario u ON i.IDusuario = u.IDusuario where i.IDruta
= r.IDruta order by ruta, FechaSalida
```

- Consulta de usuarios para la carga del contenido del control desplegable *Usuario* en el formulario. Ordenamos alfabéticamente en función del campo consultado *Usuario*, que se trata de otro de los casos que implementa vista alternativa.

```
select IDUsuario, concat (ifnull(concat(nombre, ' '),
''), ifnull(concat(apellidos, ' '), ''), 'DNI ', DNI) as
Usuario from pfc_usuario order by Usuario
```

- Consulta de rutas para la carga del contenido del control desplegable *Ruta* en el formulario. Ordenamos alfabéticamente en función del campo consultado *ruta*, que de nuevo implementa la vista alternativa.

```
select IDRuta, ifnull(Nombre, concat('ID ', IDRuta)) as
ruta from pfc_ruta order by ruta
```

- Consulta para la carga del contenido del formulario en la edición.

```
select IDItinerario, IDUsuario, IDRuta,
date_format(FechaSalida, get_format(DATE,'EUR')) as
FcSalida, date_format(FechaLlegada,
get_format(DATE,'EUR')) as FcLlegada, Observaciones,
Resuelto from pfc_itinerario where IDItinerario = _
```

- Modificación.

```
update pfc_itinerario set IDRuta = _, [IDUsuario = _,]
FechaSalida = str_to_date('dd/mm/aaaa','%d/%m/%Y'),
FechaLlegada = str_to_date('dd/mm/aaaa','%d/%m/%Y')/null,
Observaciones = '_'/null , Resuelto = true/false where
IDitinerario = _
```

- Creación.

```
insert pfc_itinerario (IDRuta, [IDUsuario,] FechaSalida,
[FechaLlegada,] [Observaciones,] Resuelto) value (_, [_,]
str_to_date('dd/mm/aaaa','%d/%m/%Y'),
[str_to_date('dd/mm/aaaa','%d/%m/%Y'),] ['_',]
true/false)
```

- Eliminación
```
delete from pfc_itinerario where IDItinerario = _
```

B.4 Sentencias SQL en el Entorno Móvil

Por último, las operaciones del servidor bluetooth con la BD se constituyen de las siguientes transacciones SQL en cada caso:

- Recepción de Itinerarios (37).
```
select iditinerario, fechasalida, fechallegada,
observaciones, ifnull(r.nombre, concat('ID', r.idruta))
as ruta, idparada, l.* from pfc_itinerario i, pfc_parada
p, pfc_lugar l, pfc_ruta r where idusuario = _ and
resuelto = false and i.idruta = r.idruta and r.idruta =
p.idruta and p.idlugar = l.idlugar and fechasalida <=
current_date() order by fechasalida, ruta
```
- Entrega de Itinerarios. Se realiza esta transacción tantas veces como itinerarios se están entregando.
```
update pfc_itinerario set observaciones = '_'/null,
resuelto = true/false where IDitinerario = _
```
- Actualización de Login
```
select IDusuario, DNI, Password from pfc_usuario
```

Anexo C

Memoria Económica

C.1 Fases y subfases del proyecto

La realización de este proyecto se ha extendido a lo largo de seis meses. Vamos a tratar de estimar el número de horas aproximadas que se han dedicado a cada una de las tareas que componen las distintas fases por las que transcurre la realización del proyecto. Para demostrar la validez de dicha aproximación, debemos tener en cuenta que, durante la duración del proyecto, el número medio de horas empleadas cada día puede ser alrededor de cuatro.

❖ **Análisis**

- Estudio de requisitos y formulación de objetivos: 10 horas.
- Recopilación de material de estudio: 15 horas.
- Estudio de tecnologías para la implementación del sistema.
    - Tecnologías inalámbricas (bluetooth y NFC): 35 horas.
    - Tecnología móvil (J2ME, MIDP): 5 horas.
    - Tecnología web (J2EE, servlets y JSPs,…): 40 horas.
- Establecimiento del entorno de desarrollo (Eclipse en las dos subfases del desarrollo).
    - Entorno móvil (Ambas SDKs de Nokia): 24 horas.
    - Entorno web (WTP y Eclipse – GlassFish): 16 horas.
- Familiarización con el entorno de trabajo y realización de demostraciones previas de las tecnologías elegidas.
    - Tecnologías inalámbricas: 40 horas.

- Tecnología móvil: 15 horas.
- Tecnología web (Conjunto de conexiones): 15 horas.

❖ **Diseño**

- Sistema de Información (Modelo de datos): 15 horas.
- Aplicación web (MVC, JavaBeans, herencia): 35 horas.
- Aplicaciones para móvil (MIDlets, almacenamiento): 15 horas.
- Comunicaciones inalámbricas (BlueCove y NFCIP): 30 horas.

❖ **Desarrollo**

- Base de datos (Scripts SQL): 8 horas.
- Aplicación web (Apariencia): 75 horas.
- Aplicaciones para móvil: 42 horas.
- Comunicaciones inalámbricas (Optimizaciones): 58 horas.
- Pruebas: 28 horas.

❖ **Memoria**

- Redacción de la Memoria: 200 horas.

Vamos a proceder ahora a una sencilla comprobación del tiempo total empleado por fases del proyecto, para corroborarlo con la duración aproximada de cada fase (análisis y diseño durante los dos primeros meses, los dos siguientes y un poco más para el desarrollo y algo menos de los dos últimos meses para la memoria) y, finalmente, con la duración total del proyecto:

- Análisis: 10 + 15 + 35 + 5 + 40 + 24 + 16 + 40 + 15 + 15 = 215 horas (53 días).
- Diseño: 15 + 35 + 15 + 30 = 95 horas (23 días).
- Desarrollo: 8 + 75 + 42 + 58 + 28 = 211 horas (53 días).
- Memoria: 200 horas (50 días).

❖ **Total:** 215 + 95 + 211 + 200 = 721 horas (180 días).

Se puede comprobar que el total del tiempo para la realización del proyecto concuerda con los seis meses de su duración y que los perídodos de las fases también se ajustan más o menos al tiempo desglosado.

C.2 Desglose de Costes

C.2.1 Costes de personal

Para el cálculo de los costes de personal se ha buscado la tarifa de honorarios que propone el Colegio Oficial de Ingenieros Técnicos de Telecomunicación (COITT), pero nos hemos encontrado con que "*Se suprime la función de los Colegios de fijar baremos orientativos de honorarios o cualquier otra recomendación sobre precios, […]*" (40).

Así que, de manera orientativa, se utilizará la tarifa de 2006. Cabe destacar que este proyecto es de fin de carrera y, por tanto, ha sido realizado por una persona que aún no está titulada, por lo que a efectos del presupuesto se considerará el coste de la hora trabajada como el 70% del valor de la hora tomada para un ingeniero ya titulado.

Costes de personal			
Concepto	**Tarifa**	**Tiempo**	**Importe**
Ingeniero Técnico no licenciado	42,11 €/h	721 h	**30.361,30 €**

Tabla 9 – Costes de personal

(Tanto en el cálculo de este importe como en los que siguen se redondean los céntimos de euro.)

C.2.2 Costes de materiales

Hay que tener en cuenta además los costes de materiales requeridos para la implantación del sistema, que se describen en la siguiente tabla:

Costes de materiales de implantación			
Concepto	**Precio unitario**	**Unidades**	**Importe**
Ordenador servidor[19]	900 €	1	900 €
Nokia 6131 NFC	190 €	5 (para el ejemplo de *3.4 Casos de uso*)	950 €
Total			**1.850 €**

Tabla 10 – Costes de materiales de implantación

Por otro lado, vamos a mostrar en una nueva tabla los costes de materiales empleados en el desarrollo del proyecto. Para ello, se ha supuesto que el periodo de vida media tanto de un ordenador personal como de un dispositivo móvil es de tres años, con lo que, si la duración del proyecto ha sido de seis meses, tenemos un coeficiente de amortización de 6/36. En el caso del software, no gratuito, suponemos un período de vida media de cuatro años, lo que resulta en un coeficiente de amortización de 6/48.

Costes de materiales de desarrollo				
Concepto	**Precio unitario**	**Coeficiente de amortización**	**Unidades**	**Importe**
Ordenador Portátil Toshiba Satellite	840 €	6/36	1	140 €
Nokia 6131 NFC	190 €	6/36	2	63,30 €
Software *MS*	379 €[20]	6/48	1	47,40 €

[19] Ver requisitos Hardware del Servidor en *A.1 Requisitos del Sistema*.

Office para la realización de la Memoria				
Software *Smartdraw* para la realización de la Memoria	146,20 €[21]	6/48	1	18,30 €
Total				**269 €**

Tabla 11 – Costes de materiales de desarrollo.

No se incluye en ninguna de las dos tablas anteriores el concepto de Sistema Operativo, pues en la adquisición del portátil viene incluido y se presupone que en la compra del equipo servidor también. En el caso del portátil empleado para el desarrollo el SO ha sido Windows, empleando WinSock como controlador compatible con BlueCove del dispositivo bluetooth integrado; en el caso del servidor puede ser cualquier otro SO mientras se cumpla con los *A.1 Requisitos del Sistema*.

El resto de software empleado en el desarrollo no se ha incluido por tratarse de software libre o gratuito.

Finalmente, vamos a incluir en una última tabla otros costes asociados al desarrollo:

Otros costes asociados al desarrollo				
Concepto	**Tarifa / Precio**	**Tiempo**	**Coeficiente de amortización**	**Importe**
Alquiler de estudio	500 €/mes	6 meses	-	3.000 €
Servicio de luz	25 €/mes	6 meses	-	150 €
Servicio ADSL	39,90 €/mes	6 meses	-	239,40 €
Comunidad y agua	14 €/mes	6 meses	-	84 €
Mobiliario (escritorio y silla)	150 € + 60 €	-	6/60	21 €
Material de oficina	40 €	-	1	40 €
Total				**3.534,40 €**

Tabla 12 – Otros costes asociados al desarrollo

Destacar de esta última tabla la esparanza de vida del mobiliario, de cinco años, y la del material de oficina, no superior a la duración del proyecto (6 meses).

[20] Precio obtenido de http://emea.microsoftstore.com/es/es-ES/Microsoft/Office-Hogar-y-Pequena-Empresa-2010 [Último acceso: 24/09/2010 20:00h].
[21] Precio obtenido de http://www.smartdraw.com/buy/ [Último acceso: 24/09/2010 20:00h]. Precio original $197, cambio empleado 1 € = $1,3474 de http://cambioeurodolar.com/ [Último acceso: 24/09/2010 20:00h].

C.2.3 Costes totales.

Por último, procedemos a recapitular todos los costes anteriores para resolver un precio de venta para el cliente, con el IVA actual del 18% aplicado.

Presupuesto total	
Concepto	**Importe**
Costes de personal	30.361,30 €
Costes de materiales de implantación	1.850 €
Costes de materiales de desarrollo	269 €
Otros costes asociados al desarrollo	3.534,40 €
Base imponible	**36.014,70 €**
IVA (18%)	6.482,60 €
Total	**42.497,30 €**

Tabla 13 – Presupuesto total del proyecto.

C.3 Presupuesto

El presupuesto total de este proyecto asciende a la cantidad de CUARENTA Y DOS MIL CUATROCIENTOS NOVENTA Y SIETE EUROS CON TREINTA CÉNTIMOS DE EURO.

Leganés a 13 de octubre de 2010

El ingeniero proyectista

Fdo. Juan Manuel Ardoy de Gracia.