Escola Politècnica Superior
de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: Accès via mòbils a Serveis Web publicats en una Xarxa P2P DHT.

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Telemàtica

AUTOR: Xavier del Valle Gombau

DIRECTOR: Dolors Royo Vallès

DATA: 20 de Desembre de 2007

TÍTOL: Accès via mòbils a Serveis Web publicats en una Xarxa P2P DHT.

AUTOR: Xavier del Valle Gombau

DIRECTOR: Dolors Royo Vallès

DATA: 20 de Desembre de 2007

Resumen

En este proyecto, se desea desarrollar una aplicación sobre un dispositivo móvil (ya sea PDA, SmartPhone o cualquier otro con capacidad de conectarse a la red) que sea capaz de consumir Servicios Web publicados en una red P2P DHT.

La aplicación, se implementará mediante el lenguaje JAVA por lo que será necesario estudiar las APIs que nos proporciona SUN para el desarrollo en dispositivos móviles, J2ME. También habrá que tener en cuenta las limitaciones que tiene frente a JAVA estándar ya que un dispositivo móvil no tiene la potencia que puede tener un PC.

La red P2P sobre la que se trabajará es la red Pastry y utilizaremos su implementación de libre distribución FreePastry, ya que ofrece una buena implementación de las DHT y nos proporciona un conjunto de protocolos y servicios que nos permiten crear aplicaciones distribuidas de una forma fácil y eficiente.

Finalmente, para comprobar el funcionamiento del conjunto, se ha diseñado, también mediante lenguaje JAVA, un Servicio Web XML. Para alojar los Servicios Web se ha decidido utilizar el servidor de aplicaciones Apache Tomcat añadiéndole el módulo Axis, ambos son software de libre distribución.

TÍTOL: Accès via mòbils a Serveis Web publicats en una Xarxa P2P DHT.

AUTOR: Xavier del Valle Gombau

DIRECTOR: Dolors Royo Vallès

DATA: 20 de Desembre de 2007

Overview

In this Project, I aim to develop an application for a mobile device - such as PDA, SmartPhone or another device with capability to connect to a network - that is able to interact with a Web Service published on a P2P DHT network.

The application will be implemented in JAVA language, so it will be necessary to study the SUN's API provided for develop in mobile devices: J2ME. Also, will it be necessary to consider the limitations of J2ME API as opposed to the standard JAVA runtime environment, given that the mobile devices don't have the same features and capabilities of conventional PCs.

The chosen P2P network is Pastry and we will use the open source's implementation for such network called FreePastry, which given it's very nice implementation of the DHT technology provides us with a set of protocols and services that permit a rapid and efficient application development.

Finally, an XML Web Service has been developed to verify the correct operation of the project. In order to host the Web Services, an Apache Tomcat application server with the Axis module has been chosen, both of which are Open Source software.

Me gustaría dedicar este proyecto a Elena,
sin ella no sería quien soy, a mi familia
y a mi directora Dolors.

A todos ellos, gracias por todo.

ÍNDICE

INTRODUCCIÓN

En el tiempo en que vivimos actualmente, cada vez se precisa más el tener acceso a información en cualquier momento, en cualquier lugar y de una forma eficiente. Atrás va quedando el tiempo en que era preciso un PC y un cable cada vez que se quería una conexión a la red. Hoy en día, dada esa necesidad y al gran avance por parte de los fabricantes de Hardware, surgen muchas aplicaciones destinadas a poder tener acceso a información desde dispositivos móviles (teléfonos móviles, PDA's, pocketPC's etc.).

Cuando hablamos de conexión a la red, actualmente también vamos dejando atrás la estructura típica de *cliente-servidor*, en la que *n* clientes se conectan a un servidor y éste les da servicio, para dejar paso a la arquitectura Peer-to-Peer, en la que no existen clientes ni servidores, si no que todos son nodos que hacen de clientes y a la vez de servidores de otros nodos formando así, una red *Overlay*.

El objetivo de este proyecto, es diseñar una aplicación sobre un dispositivo móvil capaz de consumir Servicios Web publicados en una red Peer-to-Peer.

Los Servicios Web son un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones, la ventaja de estos servicios, es que los datos se pueden intercambiar independientemente del lenguaje de programación y de la plataforma desde donde se ejecutan. Esta interoperabilidad es posible gracias a la utilización de estándares abiertos.

Este proyecto está claramente diferenciado en tres partes que son las que comentaré a continuación, éstas son, Redes Peer-to-Peer, Servicios Web y Programación de aplicaciones sobre dispositivos móviles. Después entraremos en lo que ha sido el desarrollo del proyecto, con su correspondiente planificación y evaluación de los costes, una valoración del impacto medioambiental y por último, las conclusiones.

CAPÍTULO 1: REDES Peer-to-Peer

En este capítulo veremos el funcionamiento y las características de las redes Peer-to-Peer, las diferencias que tienen frente a las redes centralizadas convencionales, sus ventajas y las APIs que utilizaremos en este proyecto.

1.1 Arquitectura centralizada vs Peer-to-Peer.

1.1.1 Arquitectura centralizada.

Actualmente, existen multitud de servicios y aplicaciones que siguen esta arquitectura, por ejemplo, cualquier portal Web en el cual, después de autenticarte como usuario, vamos haciendo peticiones al servidor y éste nos va dando sus respuestas.

Este tipo de arquitectura tiene varias desventajas, ya que dependen del servidor central que les proporciona el servicio. En el momento que ese servidor falla, automáticamente, los usuarios caen por efecto barrido.

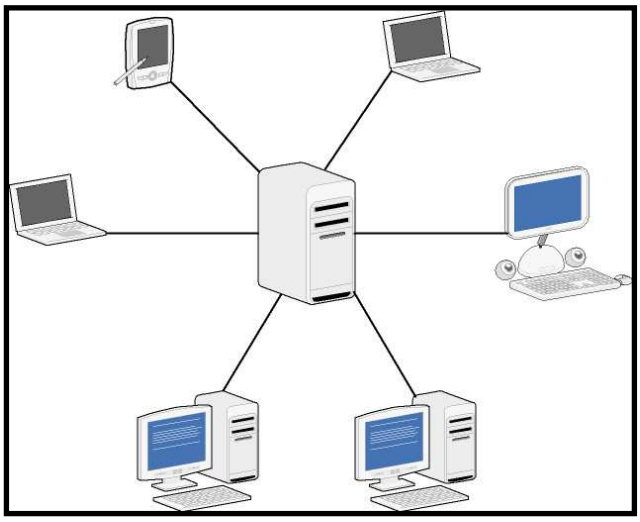


Fig. 1 - Arquitectura centralizada

Otro problema de esta arquitectura, es la seguridad. Con la arquitectura centralizada los usuarios han de guardar sus datos, passwords etc. en dicho servidor central, lo que en caso de ataque al servidor, los datos de los usuarios podrían verse comprometidos.

Por último, el mantenimiento de los servidores centrales supone un gasto, por lo que muchos sistemas tienen tarifas, a demás, de que el usuario debe adaptarse a los requerimientos, tanto de software como de hardware, que exija el servidor para poder utilizar sus servicios.

Estos problemas se pueden solucionar gracias a las redes Peer-to-peer (P2P).

1.1.2 Arquitectura Peer-to-Peer.

La arquitectura Peer-to-Peer *ref. [1]* (también conocido con el acrónimo P2P que utilizaremos a partir de ahora) define una red de máquinas que siguen una arquitectura de red totalmente descentralizada. En dicha red las máquinas presentes (en adelante nodos), comparten una serie de recursos y servicios distribuidos a través de toda la red. Estos recursos pueden ser de cualquier tipo, desde algo tan sencillo como una impresora compartida hasta la realización de cálculos de manera conjunta o también, en nuestro caso, el consumo de Servicios Web que han sido publicados en ella.

Las máquinas que conforman una red P2P no necesitan encontrarse bajo una arquitectura física de red determinada, ya que P2P es una arquitectura de tipo lógica, únicamente necesitan poder "verse" las unas a las otras. Este tipo de redes se llaman redes *Overlay*. Por tanto en una misma red P2P pueden estar máquinas situadas en puntos muy distantes físicamente y que pertenezcan a redes y dominios totalmente distintos.

Esto hace que P2P sea una arquitectura perfecta para ser utilizada tanto en entornos de redes LAN privadas como de Internet, pudiendo existir en ambas al mismo tiempo.

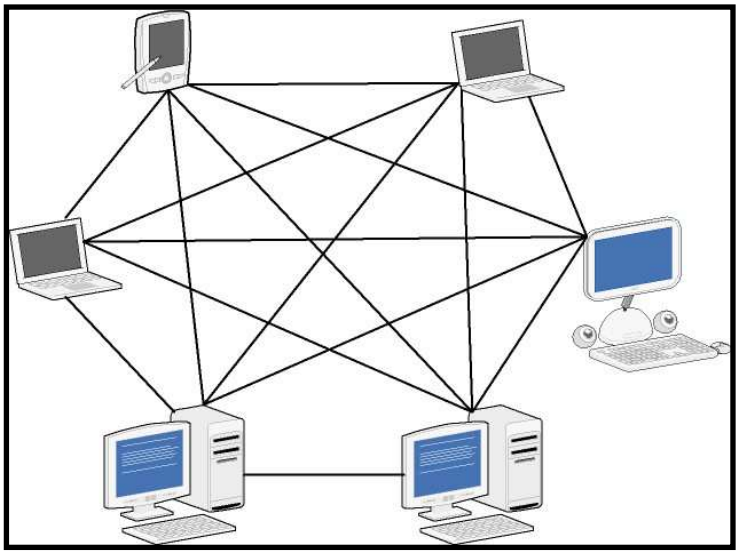


Fig. 2 - Arquitectura P2P

Como podemos ver en la figura anterior, la caída de un nodo en una estructura P2P, no afecta en absoluto al resto de nodos, éstos pueden seguir realizando sus funciones ya que están conectados entre sí. Además como los servicios que se utilizan no se encuentran centralizados en el nodo principal, si no que se encuentran en todos los nodos, el sistema continúa en perfecto funcionamiento.

1.2 Características de las redes P2P

Las redes P2P tienen la filosofía de que todos los usuarios deben compartir y mientras más usuarios haya en la red, mejor será su funcionamiento. Estas son algunas de las características deseables en toda red P2P. *ref. [2]*.

- Escalabilidad.
- Robustez.
- Descentralización.
- Balanceo de carga.
- Privacidad.
- Seguridad.

En un sistema P2P no existe un nodo central, por lo que no hay ningún problema si algún nodo cae, ya que existen otros nodos que pueden ofrecer los mismos servicios. Como todos los nodos del sistema son a la vez clientes y servidores de los servicios y que cada nuevo nodo proporciona nuevos recursos a la red, el sistema no se resiente si el número de nodos presentes es muy grande. Esto da una gran escalabilidad y robustez al sistema, ya que la caída de un nodo no implica la desaparición de los servicios que ofrecía.

En las redes P2P todos los nodos son iguales ante el resto, por tanto los usuarios que entran al sistema lo hacen de una forma totalmente anónima no proporcionando ningún tipo de información sobre ellos mismos. No obstante si que es posible identificar las máquinas mediante algún tipo de identificador. Estos datos, pueden cambiar y no ser siempre los mismos si no existe un sistema de autenticación, cosa que es totalmente opcional pero no incompatible.

1.3 La red P2P Pastry.

Pastry es un sistema de redes P2P, que funciona como un sistema independiente de posición distribuido a través de muchos terminales individuales, que hacen de nodos dentro de la red.

Es un sistema distribuido en el que todos los nodos tienen capacidades idénticas y donde las comunicaciones son simétricas. Pastry apunta para proporcionar una disponibilidad, adaptabilidad y seguridad altas, gracias a su arquitectura descentralizada.

Pastry es, por tanto, un sistema genérico de localización de objetos (nodos) y encaminamiento P2P basado en una red *overlay* auto-organizada, en la que los nodos se conectan los unos a los otros mediante Internet.

Una vez montada la red y añadidos los nodos que hacen de usuario, sobre Pastry pueden ir diferentes aplicaciones, en nuestro caso, Servicios Web.

1.4 Tablas de Hash Distribuidas (DHT)

1.4.1 Funcionamiento de las DHT

Para poder almacenar toda la información necesaria para poder encaminar mensajes, necesitamos almacenar información de los nodos, donde se encuentran, que identificadores tienen, que key, etc. por ello hacemos uso de las tablas DHT, que es una tabla virtual donde se almacena la información de los nodos que han ingresado en la red Pastry.

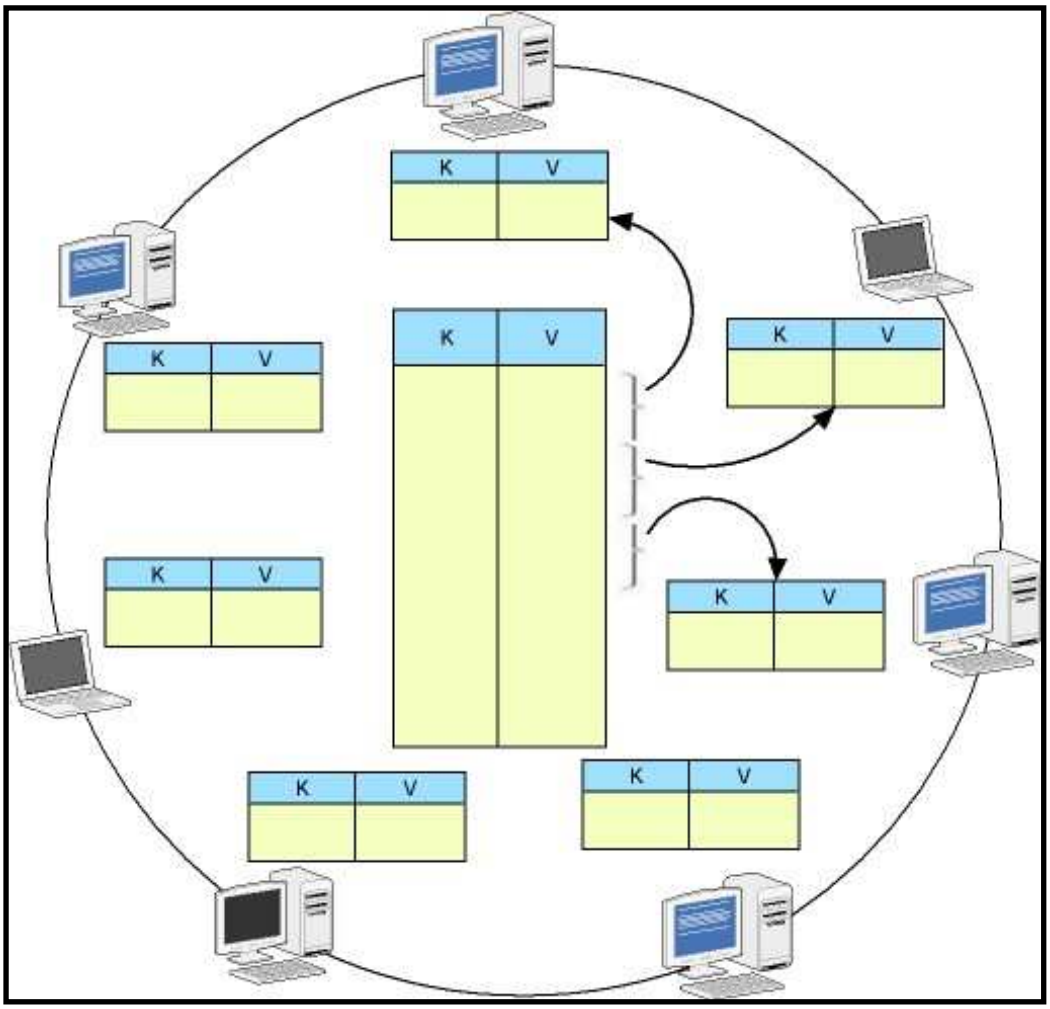


Fig. 3 – Estructura DHT en Pastry.

Decimos que es una tabla virtual porque no existe físicamente una tabla DHT con la información de todos los nodos de la red, si no que cada nodo tiene pequeñas tablas que juntas forman el total.

1.4.2 Características de las DHT

Las principales características de una DHT son:

- Descentralización: La información está compartida entre todos los nodos que forman el sistema, no se requiere ningún nodo central que almacene toda la información.
- Escalabilidad: El sistema mantiene un funcionamiento eficiente independientemente de la cantidad de elementos en el sistema, ya sean unos pocos o millones de ellos.
- Tolerancia a los fallos: El sistema debe funcionar correctamente aún cuando los nodos entren y salgan del sistema de manera continua. La información no debe perderse si un nodo abandona el sistema.

La información almacenada, se encuentra totalmente distribuida entre nodos independientes situados de manera aleatoria en el mundo físico, lo que nos garantiza que la información se encuentra descentralizada.

Es muy escalable ya que al no existir ningún nodo central que deba coordinar el acceso a los datos hace que el sistema no se resienta en situaciones con muchos nodos dentro del sistema.

Del mismo modo también se implementa un funcionamiento tolerante a fallos en lo que respecta a la entrada y salida de nuevos nodos al sistema y la reestructuración de como se almacena la información en dichos nodos.

1.5 FreePastry

FreePastry es una implementación en Java destinada al desarrollo de aplicaciones P2P siguiendo la estructura de una DHT. Cumple con todas las características de una red P2P que hemos comentado anteriormente aunque no sea una red P2P pura. Esta es la que hemos elegido para el desarrollo en este proyecto.

1.5.1 Estructura de FreePastry

Según explican en la página de los diseñadores de FreePastry, *ref.[3]*, FreePastry, tiene un conjunto de protocolos propios que se encargan de mantener una red P2P activa y procesar los cambios realizados por la entrada o salida de nodos de una forma totalmente transparente al usuario. Dicha red P2P se sitúa por encima de redes ya existentes, ya sean LAN privadas o la propia Internet, realizando conexiones directas entre los nodos del sistema (*red Overlay*).

Para ello define una arquitectura en anillo donde los nodos se conectan entre sí, por lo que no sigue una estructura P2P pura. En el momento que un nuevo nodo entra al sistema, FreePastry le da una posición dentro del Anillo y reestructura a los nodos que se encuentran a ambos lados del nuevo nodo. En el caso de que un nodo abandone el sistema se encarga de poner en comunicación a los nodos que se encontraban a ambos lados del nodo que ha

abandonado el anillo manteniendo así la conectividad entre todos los elementos del sistema.

La estructura de dichos nodos se realiza de una manera totalmente aleatoria. Al iniciar la aplicación, FreePastry genera un identificador Hexadecimal de manera aleatoria. Dicho identificador es el responsable de identificar al nodo dentro de la red. Cuando un nodo desea entrar al sistema, lo notifica y el sistema le indica cuales son los nodos con identificadores directamente "mayor" y "menor", entre estos 2 nodos será donde el nuevo nodo se insertará dentro del anillo. Los identificadores de los nodos se encuentran distribuidos de manera uniforme, ya que son generados de manera totalmente aleatoria, por lo que se entiende que nodos adyacentes, de manera lógica, pueden estar separados cientos de kilómetros. Esto garantiza la heterogeneidad del anillo, haciendo que si por ejemplo 5 nodos de una misma red abandonan el servicio a la vez, por la caída de un enlace, las reestructuraciones sean en distintas partes del anillo corrigiendo únicamente la ausencia de un nodo en cada caso.

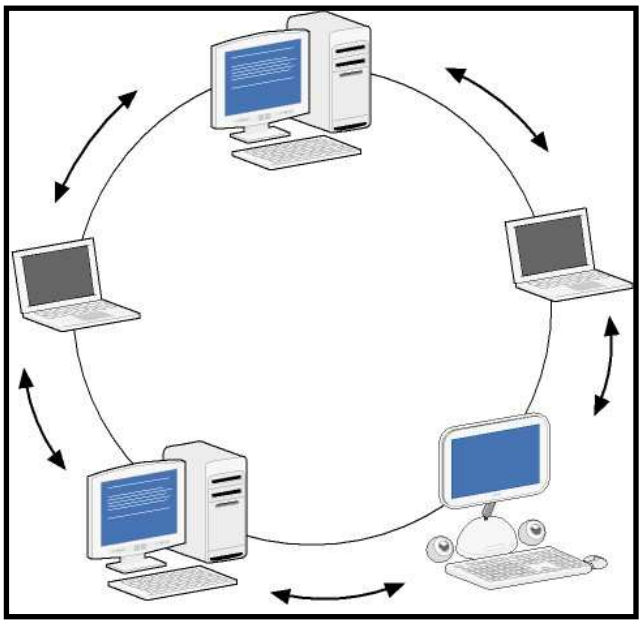


Fig. 4 – Anillo de FreePastry

Cuando un nodo quiere entrar a formar parte del anillo, primero lo notifica y se le asigna un identificador de nodo (NodeID), el cual es un identificador de 40 caracteres en formato Hexadecimal. También es posible obtener una versión reducida, de menos caracteres.

1.5.2 Comunicación entre nodos.

Como se ha explicado en apartados anteriores, FreePastry no es una red P2P pura, ya que no existe conectividad directa entre todos los nodos de la red.

Siendo así, la comunicación entre nodos adyacentes, es directa, el problema viene cuando queremos comunicarnos con un nodo que tiene 2 o más saltos. Para ello, FreePastry tiene un algoritmo basado en los identificadores de nodo.

Cuando necesita enviar un mensaje a un nodo al que no tiene ruta directa, lo envía al nodo cuyo identificador está, numéricamente, más cercano al del nodo destino, este lo procesa e intenta direccionarlo siguiendo el mismo proceso.

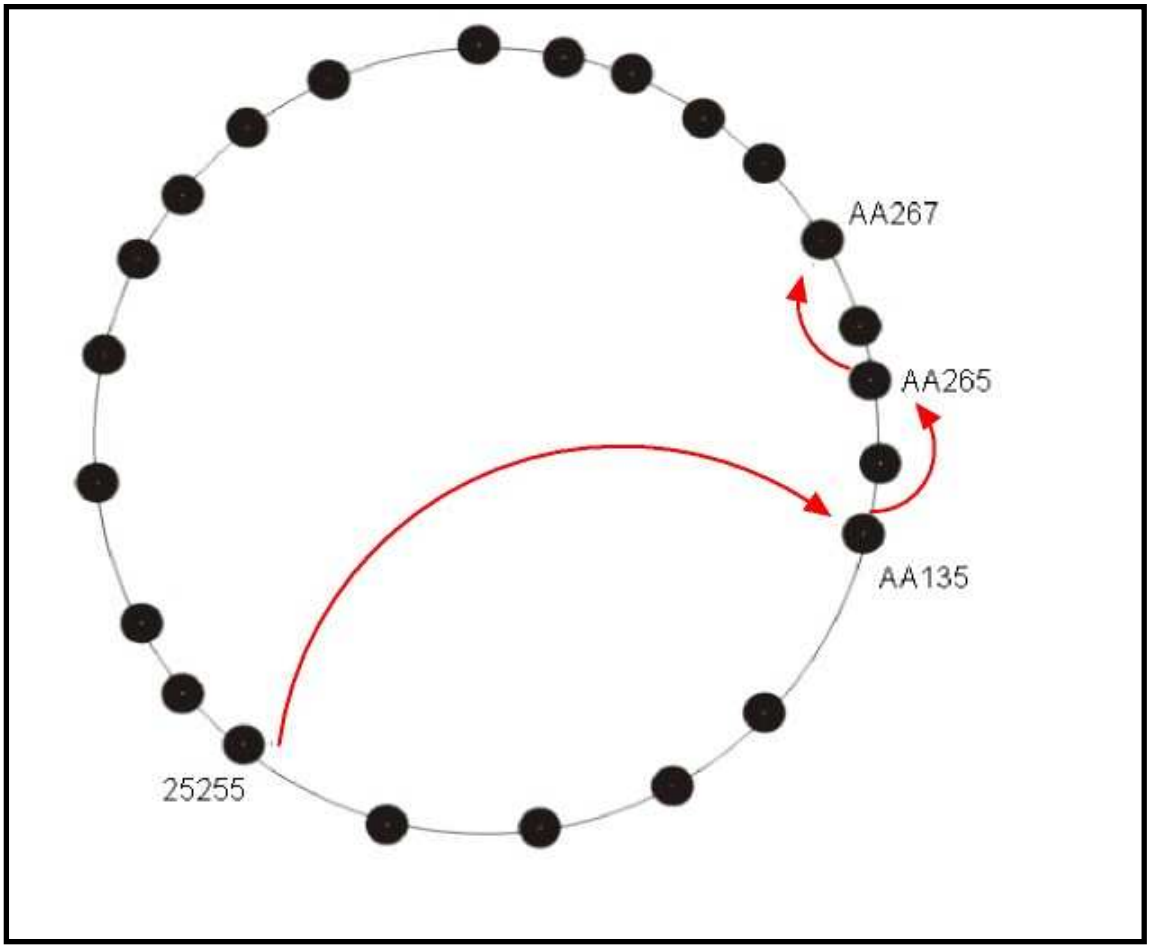


Fig. 5 – Enrutamiento en FreePastry.

De esta forma y siempre que no haya habido una reestructuración del anillo muy grande durante este proceso se consigue llegar al nodo destino en menos de Log N saltos, siendo N el número total de nodos en el sistema.

1.5.3 Bootstrap Node

Como hemos mencionado anteriormente, no existe ningún servidor central que sea el encargado de gestionar la red, pero si que es cierto que en FreePastry, se necesita un punto de partida para podernos unir al anillo. Ese punto es el Bootstrap node que es el encargado de darnos la bienvenida al sistema y lo debemos conocer a priori, a partir de ahí nuestro nodo se posicionará dentro del anillo y crearemos la tabla de rutas para los demás nodos del anillo.

CAPÍTULO 2: Servicios Web

2.1 Introducción.

El término Servicio Web es un concepto del que actualmente se abusa con cierta frecuencia y más aun en la actualidad ya que al tratarse de una tecnología relativamente reciente, no se tiene una idea clara y concisa acerca de los requisitos que debe cumplir un sistema software para que sea realmente un SW. Son muchas y muy variadas las definiciones de SW que actualmente se pueden leer en las distintas publicaciones que abordan este tema. A continuación veremos algunas de ellas.

2.1.1 Definiciones de Servicio Web.

Una definición precisa de SW sería la emitida por la W3C, *ref. [4]:*

"Una aplicación software identificada por una URI, cuyas interfaces y vinculaciones son capaces de ser definidas, descritas y descubiertas como artefactos XML. Un SW soporta la interacción con otros agentes software mediante el intercambio de mensajes basado en XML a través de protocolos basados en Internet".

Por otro lado, según un informe de la universidad de Castilla la Mancha, *ref.[5],* otra definición sería: "Los SW son interfaces Web genéricas a servicios componente. A fin de soportar la interoperabilidad entre todas las arquitecturas, los SW utilizan protocolos del W3C como pueden ser XML, WSDL y SOAP".

Nosotros nos quedaremos con que los SW son una forma estandarizada de integrar aplicaciones basadas en Web mediante estándares abiertos, como XML, SOAP, WSDL, y UDDI.

2.1.2 Elementos de los Servicios Web.

Los SW, constituyen un avance tecnológico que no está ligado a ninguna tecnología, ya que una de sus premisas es la interoperabilidad multiplataforma.

Un SW es más una idea, un concepto cuya implementación debe respetar una serie de reglas e interfaces para poder ser accedido por cualquier sistema conectado a la red, siempre que disponga de permiso para ello.

Fuera de toda implementación, como hemos dicho, existen una serie de partes bien diferenciadas que tenemos que tener en cuenta a la hora de implementar y utilizar un SW. Estas son:

Los agentes y los servicios, el cliente y el proveedor, la descripción del servicio, la semántica y por último las personas, la semántica y los agentes.

A continuación explicaré brevemente la funcionalidad de cada uno de ellos, ref. [6].

2.1.2.1 Agentes y servicios.

Un SW debe ser implementado por algún agente. El agente es un trozo de software que implementa la funcionalidad que realiza el SW.

Este agente tiene la peculiaridad de estar capacitado para enviar y recibir mensajes. De esto se desprende, que el agente es el encargado de recibir las peticiones y enviar las respuestas.

2.1.2.2 Cliente y proveedor.

El SW pertenece a un propietario que puede ser bien una persona o una organización.

El *proveedor,* será la persona u organización que desarrolle un agente capaz de soportar un determinado servicio.

La *entidad Cliente,* puede ser también una persona u organización que desea hacer uso del servicio que expone la entidad proveedora.

Para que la comunicación se lleve a cabo de manera correcta, las entidades participantes deben ponerse de acuerdo, tanto en la semántica como en los mecanismos utilizados en el intercambio de mensajes.

2.1.2.3 Descripción del servicio.

El intercambio de mensajes entre las entidades está guiado por un protocolo que éstas deben seguir para que las transacciones lleguen a buen fin.

Este protocolo está documentado en la descripción del SW (WSD, Web Service Description). La WSD es una descripción completa del SW, escrito en un lenguaje denominado WSDL.

2.1.2.4 Semántica.

La semántica de un SW, es el comportamiento que se espera que tenga el mismo. Este comportamiento, se refiere a qué esperamos del SW cuando le enviamos un mensaje de petición.

De la misma forma que la sintaxis de un objeto es su estructura, la semántica de un objeto se refleja en las relaciones que éste establece con otros objetos.

Esta semántica también puede verse como un contrato entre la entidad que realiza la consulta y la entidad a la que va dirigida la consulta. Este contrato supone un total acuerdo entre las dos entidades, y trata sobre cómo y porqué los agentes que intervienen en la comunicación deberán interactuar. Este acuerdo puede ser explícito o implícito, oral o escrito, inteligible por la máquina o por las personas.

2.1.2.5 *Personas, semántica y agentes.*

El papel que tienen las personas en el uso y en la arquitectura de los SW se puede se puede ver en dos aspectos:

1. Los usuarios de los SW deberán estar de acuerdo con la semántica y la descripción del servicio al cual dirigen las peticiones.

2. Los creadores de los agentes deben asegurar que estos cumplen la semántica y la descripción del servicio.

2.1.2.6 *Resumen.*

Para que todos estos conceptos antes mencionados queden más claros, los resumiremos en la siguiente figura.

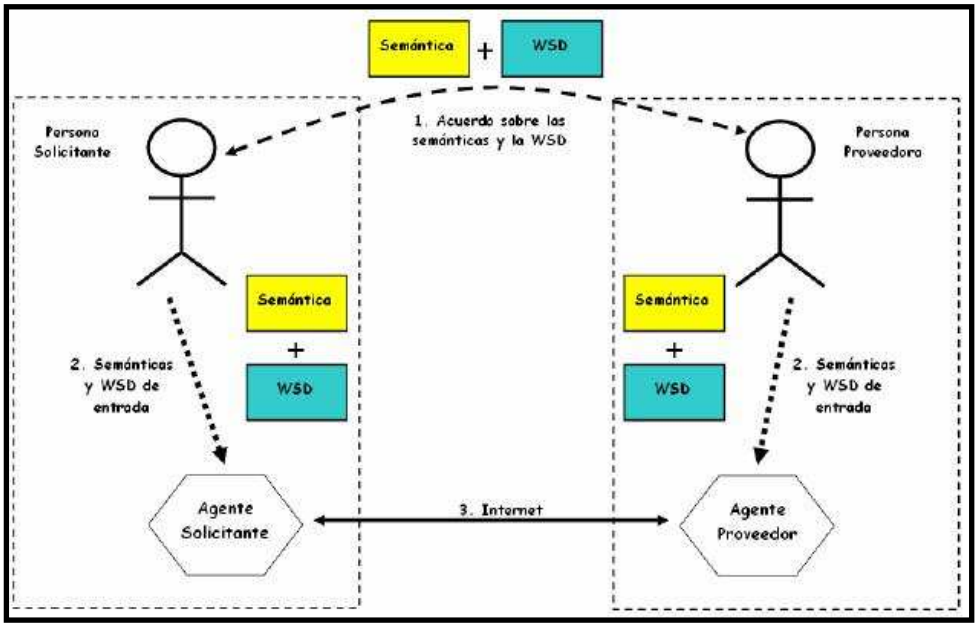


Fig. 6 – Elementos de los Servicios Web

Como se puede ver, proveedor y cliente, se ponen de acuerdo respecto a la descripción del servicio (mediante un documento en WSDL) y la semántica que guiarán la interacción entre los agentes.

Ambos agentes (el solicitante y el proveedor) intercambian mensajes SOAP en nombre de los respectivos propietarios.

2.2 Estándares básicos.

Los estándares básicos más importantes y extendidos son: SOAP (Simple Object Access Protocol, Protocolo de acceso a objetos sencillos), WSDL (Web Services Description Language, Lenguaje de descripción de servicios Web), UDDI (Universal Description, Discovery, and Integration, Descripción, detección e integración universales). A continuación se explicará brevemente cada uno de ellos, *ref.[7]*.

2.2.1 SOAP (Simple Object Acces Protocol).

SOAP es un protocolo estándar creado por Microsoft, IBM y otros que define como comunicar dos objetos en diferentes procesos mediante el intercambio de mensajes XML.

Su funcionamiento se ilustra en la siguiente figura:

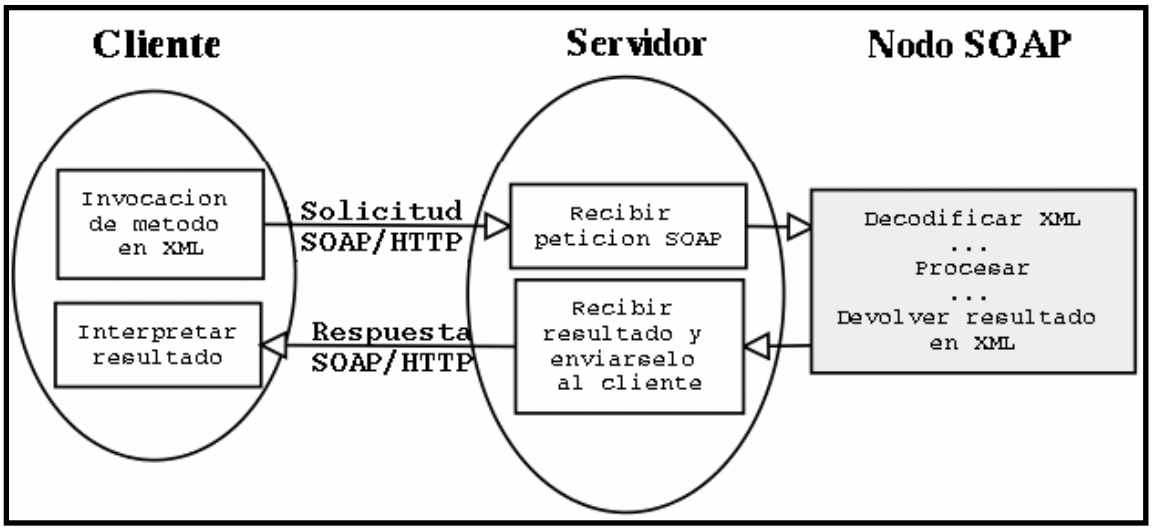


Fig. 7 Funcionamiento de SOAP.

Como podemos ver en la figura anterior, el cliente invoca una solicitud al servidor, éste recibe la petición, la procesa y le devuelve el resultado al cliente.

Las ventajas de utilizar esta tecnología son:

- No está asociada a ningún lenguaje de programación ni a ninguna plataforma.
- No se encuentra asociado a ningún protocolo de transporte, no es necesario utilizar http, se puede enviar en cualquier protocolo capaz de transportar texto.
- Utiliza estándares existentes, es decir, SOAP aprovecha XML para la codificación de los mensajes, y los mensajes se pueden enviar mediante http.

2.2.2 WSDL (Web Services Description Language).

Para crear una aplicación cliente que llame a un servicio Web, se necesita saber dónde se encuentra ubicado dicho servicio y como acceder a él.

El Lenguaje de descripción de servicios Web, WSDL, ofrece esta información. Un documento WSDL, escrito en XML, describe un servicio Web y el proceso que hay que seguir para llamarlo.

Un programa cliente que se conecta a un Servicio Web, puede leer el WSDL para determinar que funciones están disponibles en el servidor. El cliente puede usar SOAP para hacer la llamada a una de las funciones listadas en el WSDL. *Ref.[6].*

2.2.3 UDDI (Universal Description Discovery and Integration).

UDDI consiste en definir cómo se dará a conocer un Servicio Web para que los clientes interesados puedan descubrirlo fácilmente y poder utilizarlo en sus aplicaciones.

UDDI está diseñado para almacenar de forma estructurada información sobre empresas y los servicios que éstas ofrecen. A través de UDDI, se puede publicar y descubrir información de una empresa y de sus servicios.

Para aclarar un poco el concepto de UDDI, se puede pensar que es como unas “páginas amarillas” donde nosotros buscamos un servicio por una determinada categoría y encontramos las empresas que lo ofrecen.

CAPÍTULO 3: PROGRAMACIÓN EN DISPOSITIVOS MÓVILES

3.1 Introducción a los dispositivos móviles.

Cuando pensamos en un dispositivo móvil (DM en adelante), lo primero que nos viene a la cabeza es un teléfono móvil, pero en la actualidad hay una gran variedad de dispositivos que no tienen porqué ser teléfonos, como por ejemplo, PDA's, PocketPC's, SmartPhones entre otros.

Esto da lugar a una problemática para los programadores de estos dispositivos, ya que cada uno de ellos tiene unas características particulares, disponen de una memoria determinada o tienen que soportar lenguajes de programación específicos. Esta problemática reside a la hora de estandarizar todos los dispositivos, ya que están implementados en distintos lenguajes y tienen también diferentes sistemas operativos.

3.1.1 Características de los dispositivos móviles.

La principal característica de un DM, es su gran capacidad de comunicación, la cual nos permite tener acceso a información y a servicios independientemente del lugar y del momento en el que nos encontremos. Es decir, es una fuente de información fácil de transportar.

La movilidad de un DM está condicionada por la necesidad de utilizar una batería. Esto representa un inconveniente debido a que necesitan recargas periódicas lo que dificultan la portabilidad.

Otra característica, es su reducido tamaño lo cual favorece la movilidad, pero tiene como inconvenientes, el uso de procesadores más simples y memorias de menor capacidad. Además la pantalla y el teclado son también reducidos y el reconocimiento de voz es muy limitado.

Para intentar solucionar toda esta problemática y así conseguir una mayor funcionalidad, se tiene que aprovechar la gran capacidad de comunicación que tienen programando la "lógica pesada" en las aplicaciones ubicadas en los servidores y intentar que las interfaces del DM sean lo más simples posibles.

3.1.2 Tipos de dispositivos móviles.

A continuación veremos brevemente los tipos de dispositivos móviles que existen hoy en día en el mercado.

3.1.2.1 Teléfono móvil.

Un teléfono móvil es un teléfono sin hilos conectado a una red celular. Esta clase de DM permite a los usuarios mantener conversaciones telefónicas en tiempo real desde cualquier lugar cubierto por la red.

3.1.2.2 PDA (Personal Digital Assistant).

Las PDA son agendas personales electrónicas que tienen capacidad para almacenar datos y que poseen mayor memoria que los teléfonos móviles. También disponen de una pantalla más grande y tienen mayor capacidad de proceso.

Existe gran variedad de PDA's en el mercado y debido al éxito que están teniendo en los últimos años, hay diversas compañías dedicadas a hacer los sistemas operativos de éstas como por ejemplo: Palm Source, Linux, Symbian y Microsoft Windows CE.

3.1.2.3 Híbridos.

Existen dos clases de híbridos, PDA's con propiedades de teléfonos móviles y teléfonos móviles con propiedades de PDA. Los dos intentan coger las ventajas del otro, Teléfonos móviles con mayor capacidad de proceso y pantalla de mayores dimensiones y PDA's con capacidad de comunicación con otros usuarios.

Los dos tipos de híbridos más comunes son;

- *Communicators*: Son pequeños ordenadores con posibilidad de uso de telefonía.
- *SmartPhones*: Son teléfonos con ciertas capacidades que se asemejan a los ordenadores.

3.1.3 Sistemas Operativos.

A continuación veremos rápidamente algunos de los más importantes sistemas operativos que podemos encontrar hoy en día en los DM disponibles en el mercado.

3.1.3.1 Palm OS.

Las principales características de Palm OS, son su gran rapidez y la poca memoria que ocupa. También posee un gran volumen de aplicaciones destinadas a cubrir las necesidades del usuario. Otro aspecto a destacar, es que el software incluido en esta plataforma, es compatible con documentos de Word, Excel, registros e-book, e-mail y navegadores WAP y Web.

El gran punto débil de Palm OS es la escasa importancia que le da al aspecto multimedia.

3.1.3.2 Windows CE.

Este sistema operativo es altamente modular y permite a los desarrolladores crear de forma flexible aplicaciones para DM de 32 bits que se integran fácilmente con Windows y con Internet.

3.1.3.3 Windows Mobile 2003.

Es la nueva versión que mejorada que ofrece Microsoft para sustituir a su predecesor Windows CE, las mejoras que ofrece este versión son las siguientes:

- Permite una fácil conexión a multitud de redes inalámbricas. Detecta automáticamente redes WI-FI y se conecta a ellas de una forma sencilla, también a redes de área personal mediante bluetooth.
- Amplía sus funcionalidades de mensajería y e-mail integrándose con Microsoft Exchange Server 2003.
- Incluye soporte integrado para Microsoft .net Compact Framework lo que permite desarrollar aplicaciones que aprovechen las ventajas de los Servicios Web y XML.

3.1.3.4 Linux.

Linux, es un sistema operativo compatible con UNIX, las dos características más importantes de este sistema operativo es que es libre, no tiene ningún coste de licencia, y la segunda es que el sistema viene acompañado del código fuente.

3.1.3.5 Symbian.

Este sistema operativo, tiene características que influyen de una manera determinante en el desarrollo de aplicaciones:

- Ha sido diseñado para el ahorro de batería.
- Cada aplicación corre sus propios procesos y sólo tiene acceso a su espacio de memoria.
- El sistema tiene componentes que permiten el diseño de aplicaciones multiplataforma.

Symbian se ha dedicado sobretodo al desarrollo de sistemas operativos para SmartPhones aunque ahora se ha extendido su uso a los teléfonos móviles de última generación.

3.2 Plataformas de Desarrollo.

Aunque hay varias plataformas para el desarrollo de aplicaciones para dispositivos móviles, nos vamos a centrar en la que finalmente se ha utilizado para el desarrollo de este proyecto, J2ME de Sun, pero haremos un repaso rápido a la tecnología Microsoft .net Compact Framework de Microsoft.

3.2.1 Microsoft .net Compact Framework.

Proviene directamente de .NET Framework de Microsoft ref. [8] incluyendo las librerías de clase base y otras adicionales utilizadas específicamente para interactuar con dispositivos móviles. El CLR (Common Language Runtime) del Compact Framework es diferente al del .NET Framework. Esta hecho para manipular eficientemente los recursos de los dispositivos móviles ya que estos tienen características muy particulares que requieren de un cuidado especial.

Esta plataforma, fue creada sobretodo para la programación de PDA's u otros dispositivos que tuvieran instalado el sistema operativo Microsoft Windows CE.

Si se elige esta plataforma para desarrollar aplicaciones sobre dispositivos móviles, se tiene la ventaja que podemos utilizar la herramienta *Microsoft Visual Studio 2005* en el cual viene todo muy bien integrado, facilitando así el diseño no sólo de la interfaz gráfica sino de la funcionalidad de la aplicación.

A continuación, se muestra el resultado un ejemplo programado con esta tecnología que funcionaría en un PocketPC con Windows CE como sistema operativo.

Fig. 8 – Aplicación .net Compact Framework para PocketPC

Existen dos grandes desventajas a la hora de utilizar esta plataforma, la primera, es que no es multiplataforma, no pudiendo utilizarse bajo ningún otro sistema operativo y la otra desventaja es que no es gratuita, al contrario de la plataforma Java J2ME que a continuación veremos más detalladamente.

3.2.2 J2ME (Java 2 Micro Edition)

J2ME es la versión de Java orientada a los dispositivos móviles. Debido a que los dispositivos móviles tienen una potencia de cálculo baja e interfaces de usuario pobres, es necesaria una versión específica de Java destinada a estos dispositivos, ya que el resto de versiones de Java, J2SE o J2EE, no encajan dentro de este esquema. J2ME es por tanto, una versión "reducida" de J2SE.

3.2.2.1 Plataforma.

En la siguiente figura se muestra qué lugar ocupa la plataforma J2ME en el mundo Java. Dicha plataforma se divide en configuraciones y en perfiles por encima de ellas.

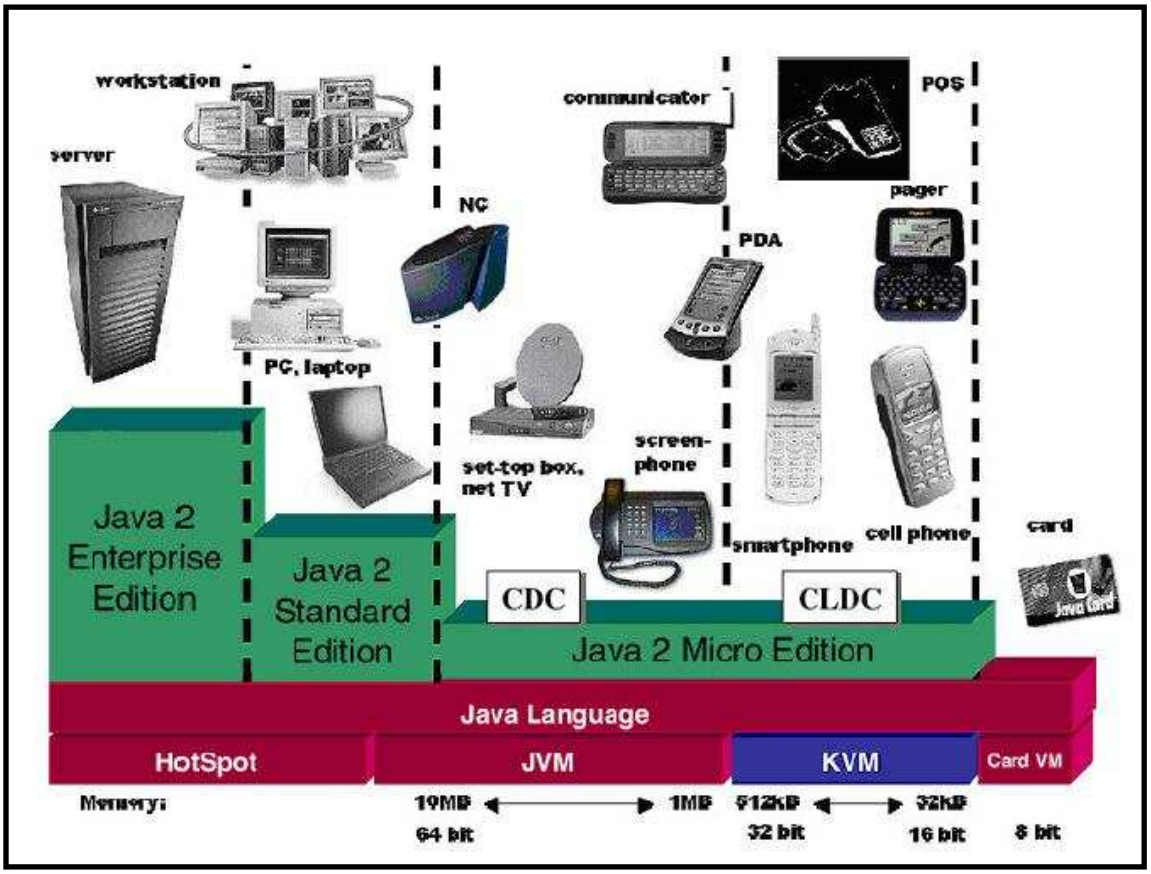


Fig. 9 - Plataforma Java

En función de las características del dispositivo para el cual desarrollemos nuestras aplicaciones, nos acogeremos a las normas impuestas por alguna de las especificaciones que aparecen en el gráfico. En nuestro caso, al orientarnos al desarrollo para dispositivos móviles de capacidades muy restringidas en gráfica, procesamiento y memoria, trabajaremos bajo la combinación siguiente:

- Configuración Connected Limited Device Configuration, CLDC (versión 1.1), a la cual nos obliga la reducida Kilo Virtual Machine (KVM) que utilizan estos dispositivos para interpretar los bytecodes JAVA que generemos.

- Perfil Mobile Information Device Profile, MIDP, en su última y mejorada versión 2.0.

3.2.2.2 Aplicaciones J2ME. Midlets.

Una aplicación JAVA que cumpla las especificaciones CLDC y MIDP será denominada MIDlet, y a varias de ellas empaquetadas en un mismo elemento (JAR) se le denominará MIDlet SUITE. El dispositivo nos permitirá seleccionar una u otra aplicación de la suite como él considere apropiado.

Tal y como se muestra en la siguiente figura, los MIDlets siempre tendrán tres métodos básicos que marcan los estados de su ciclo de vida.

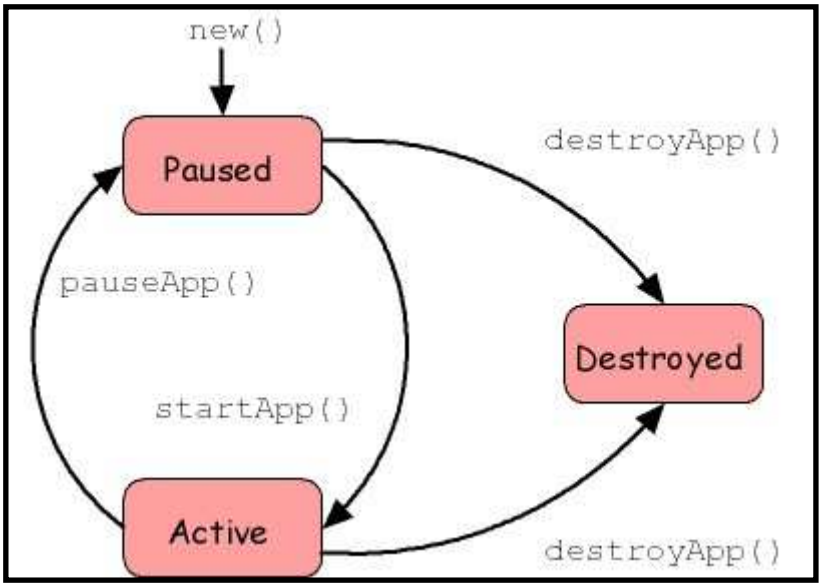


Fig. 10 – Ciclo de vida de un MIDlet.

- *Pausado*. Estado "en espera" en el que el MIDlet mantiene los mínimos recursos posibles, entrando en él al crearse, antes de ejecutarse su método startApp() o tras llamarse a su método pauseApp(). La plataforma puede pasar el MIDlet a este estado si así lo estima oportuno (por ejemplo, ante una llamada telefónica).

- *Activado*. Estado de ejecución del MIDlet al que pasa tras ejecutar su método startApp(), tanto inicialmente, como ante la recuperación de una pausa o ante casos especiales que ya estudiaremos en el siguiente capítulo. Desde este último podrá pasarse al anterior y viceversa.

- *Destruido*. Los dos estados anteriores pueden pasar a éste y de él ya no se podrá salir. Es el estado donde el MIDlet concluye su actividad, pasando a él por medio de la invocación de su método destroyApp() o, por ejemplo, ante excepción en el constructor del MIDlet.

3.2.2.3 Ejemplo MIDlet.

A continuación veremos el ejemplo más simple de MIDlet para ver lo que hemos explicado anteriormente.

```
package epsc.tfc;

import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

/**
 * HolaMundo
 * Ejemplo MIDlet
 *
 * @author Xavi
 */

public class HolaMundo extends MIDlet implements CommandListener {

	private Form form;
	private Display display;

	public HolaMundo(){
		form = new Form("Hola Mundo by: Xavi");
		form.append(new StringItem(null, "\nHola Mundo!!!"));
		form.addCommand(new Command("Salir", Command.EXIT, 1));
		form.setCommandListener(this);
	}
	protected void destroyApp(boolean arg0) throws
	MIDletStateChangeException {}

	protected void pauseApp() {}

	protected void startApp() throws MIDletStateChangeException {
		display.getDisplay(this).setCurrent(form);
	}
	public void commandAction(Command arg0, Displayable arg1) {
		notifyDestroyed();
	}
}
```

Fig. 11 – Código fuente del MIDlet HolaMundo

Este sencillo programa ilustra los métodos comentados anteriormente de su ciclo de vida. Véase también el método commandAction que es el que nos sirve para implementar los menús del programa (en este ejemplo, sólo se ha implementado el comando “Salir”).

De este modo, el resultado después de compilar el anterior ejemplo es el siguiente:

Fig. 12 – Resultado MIDlet.

Como podemos ver, la figura anterior es el resultado de la compilación del código anterior pero utilizando un emulador de dispositivo móvil. Si quisiéramos poner este programa en un dispositivo móvil real, habría que crear dos ficheros, uno con extensión .jar (propio de Java) y otro .jad (el propio MIDlet), se tienen que descargar los dos ficheros desde el móvil y ya podríamos utilizar nuestro programa en él.

CAPÍTULO 4: IMPLEMENTACIÓN DEL PROYECTO

En este apartado, entraremos en los detalles de implementación de este proyecto, tanto en el diseño del dispositivo móvil como en la modificación del nodo Pastry. También se comentarán las decisiones tomadas sobre la implementación y sobre las tecnologías utilizadas.

4.1 Consideraciones sobre la tecnología utilizada.

Como se ha comentado anteriormente, existen varias tecnologías para implementar un proyecto de estas características, pero las dos que consideré son .net Compact Framework y la de Java, J2ME (Java 2 Micro Edition).

Tanto una tecnología como otra, tienen sus ventajas y sus inconvenientes pero la que mejor se ajusta a nuestras necesidades es J2ME, las razones de esta decisión son las que a continuación detallo:

- En primer lugar, hay que tener en cuenta que FreePastry está escrito en Java, por lo tanto, existe mayor compatibilidad que si lo desarrollamos con .net.

- En segundo lugar, para desarrollar aplicaciones con .net, hay que tener obligatoriamente una máquina con el sistema operativo Windows instalado, por el contrario, utilizando Java podemos desarrollar en cualquier sistema operativo que tenga una máquina virtual (por ejemplo: Linux, Mac o BSD).

- Por último, al contrario que con la plataforma de Microsoft, Java es gratuito y también lo son la gran mayoría de sus IDEs (en este caso, Eclipse 3.2.2).

En el siguiente capítulo, se explicarán y se razonarán las decisiones tomadas en cuanto al diseño de la aplicación.

4.2 Consideraciones sobre el diseño.

A la hora de implementar la aplicación, hubo la necesidad de tomar una decisión referente a la arquitectura de red que se implementaría.

En la fase inicial del proyecto se barajaron dos opciones:

- La primera era implementar la aplicación y el nodo FreePastry dentro del dispositivo móvil.
- La segunda era modificar el nodo FreePastry para que aceptara dispositivos móviles.

En la siguiente figura se ilustra como quedaría la arquitectura con el dispositivo móvil formando parte de la red.

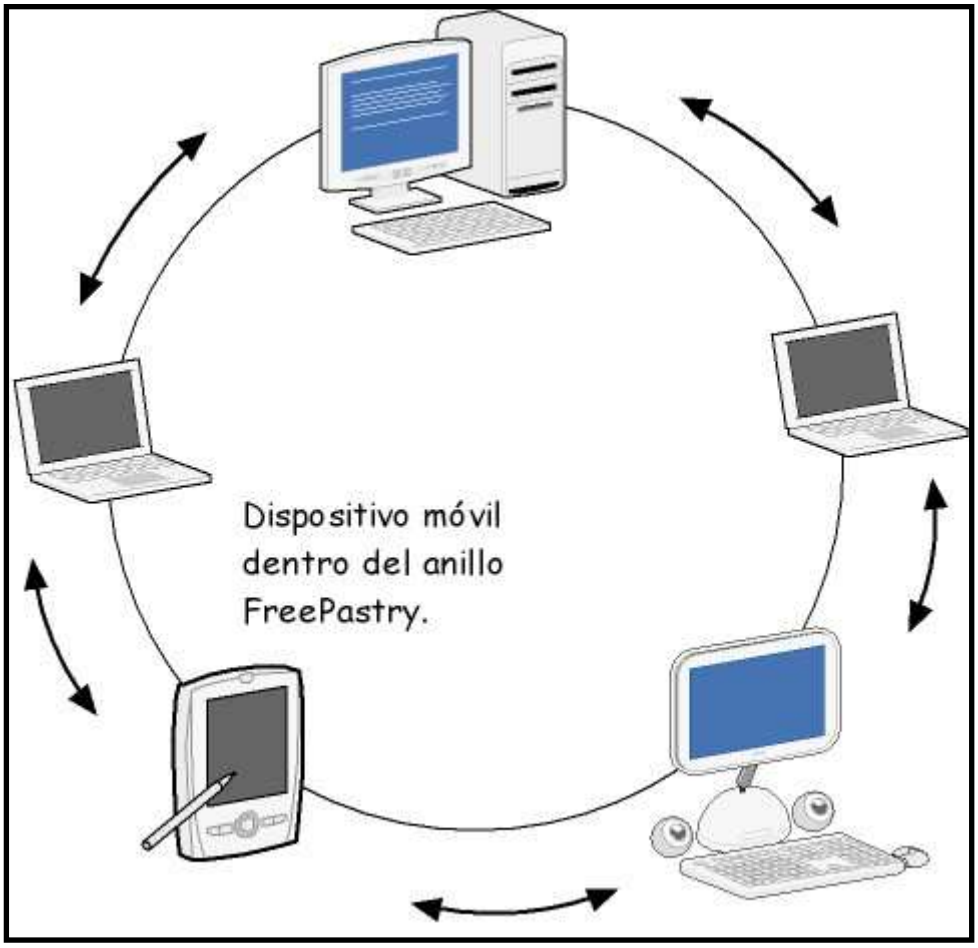


Fig. 13 - Dispositivo móvil dentro de la red FreePastry.

La ventaja que tenemos al implementarlo de esta manera, es que el dispositivo móvil tendría todas las características de un nodo cualquiera en una red DHT, por ejemplo, no le afectaría el hecho de las caídas de otros nodos gracias a la reconfiguración automática de los nodos de la red P2P.

Lamentablemente, nos encontramos con un gran inconveniente, FreePastry consume una considerable cantidad de recursos de modo que la gran mayoría de dispositivos móviles del mercado no pueden asumir esa carga, y los que pueden, no trabajarían de una manera óptima dentro de la red.

Es por ese motivo por el que se ha decidido implementar la segunda opción, ya que la aplicación implementada dentro del dispositivo móvil ocupe muy pocos recursos funcionando así muy rápidamente.

En la siguiente figura se puede ver la arquitectura resultante de ésta última arquitectura.

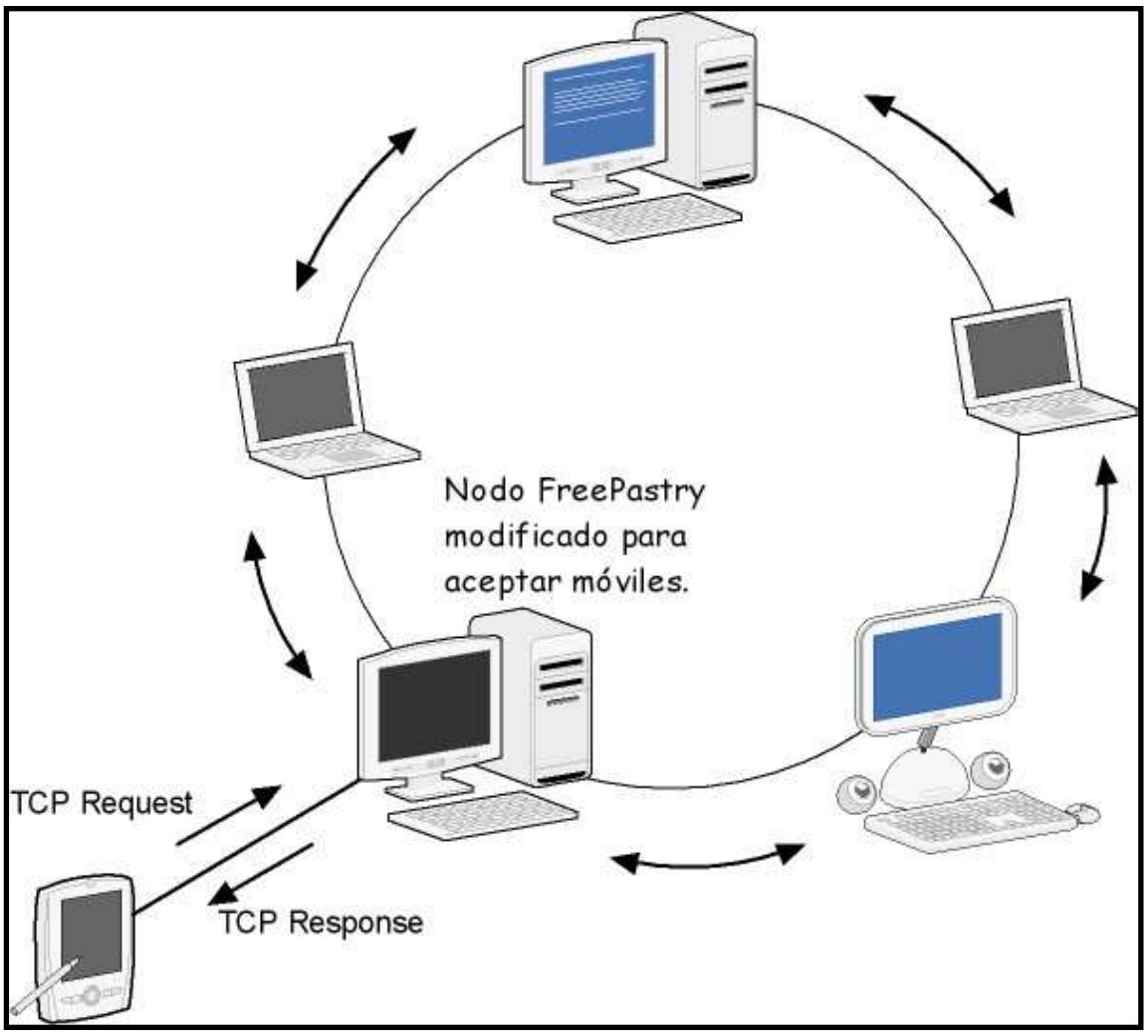


Fig. 14 – Nodo FreePastry modificado para aceptar DM.

Como hemos mencionado anteriormente, con esta arquitectura liberamos al dispositivo móvil de la carga que conlleva ser un nodo FreePastry pero tiene una desventaja, no cumple al 100% con la filosofía de las redes P2P DHT ya que si su Bootstrap node (nodo modificado al cual se conecta) cae, no será capaz de reconfigurarse automáticamente y perderá el servicio que le estuviera ofreciendo.

4.3 Diseño final.

En este apartado veremos los detalles de implementación tanto de la aplicación dentro del dispositivo móvil como de las modificaciones realizadas en el nodo FreePastry para soportar este tipo de servicio.

Para ayudar a entender el funcionamiento global de todo el sistema nos fijaremos en la siguiente figura que nos muestra la secuencia de la interacción del dispositivo móvil con el nodo FreePastry.

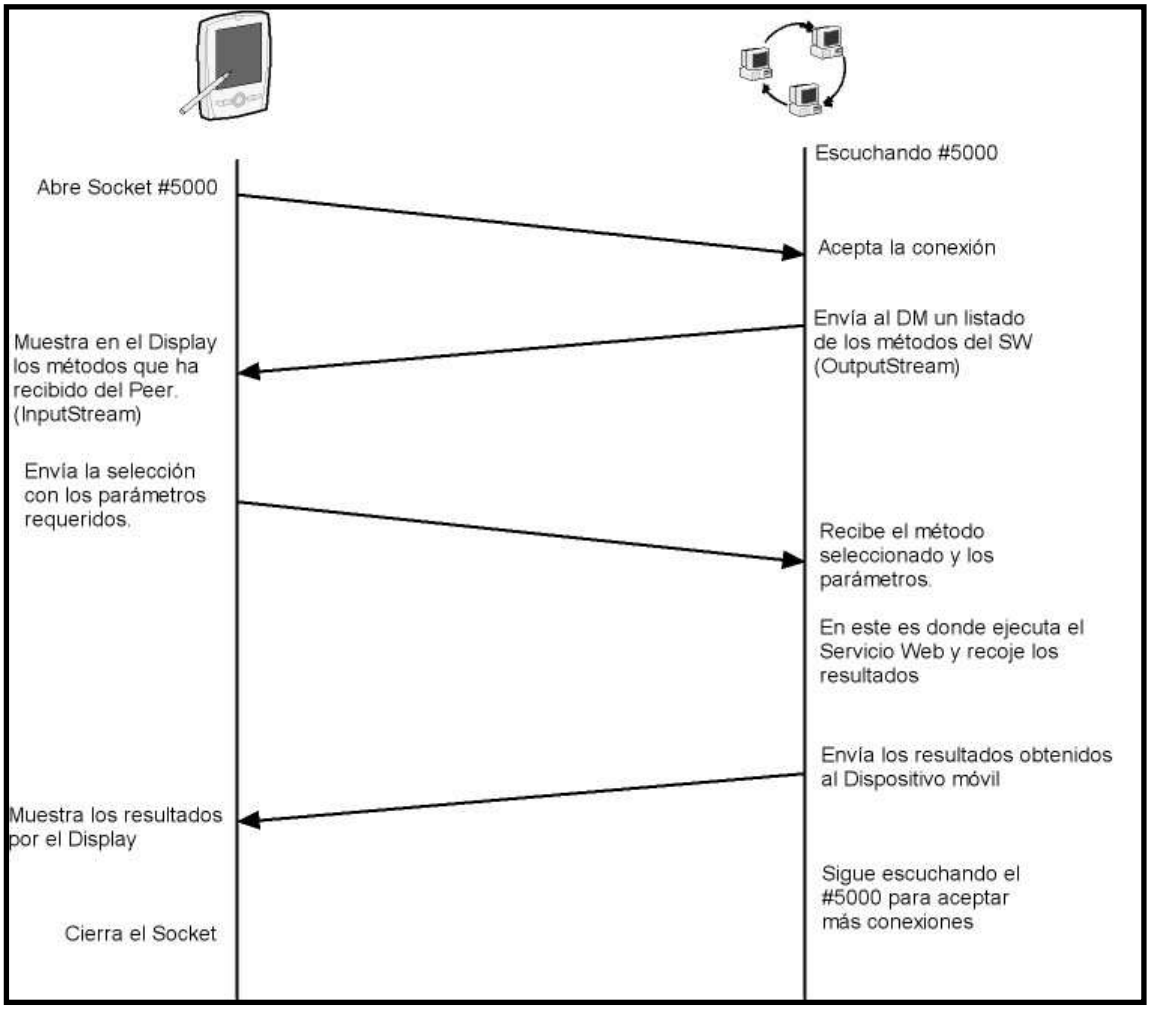


Fig. 15 – Secuencia de comunicación Peer - DM

Tal y como vemos en la figura anterior, el funcionamiento del sistema es el siguiente:

- El nodo FreePastry se ha configurado para aceptar múltiples conexiones de dispositivos móviles y continuamente está escuchando el puerto por defecto.
- El dispositivo móvil, abre un socket al puerto por defecto para iniciar la comunicación.
- El Peer, acepta la conexión y envía un listado con los métodos que tiene disponibles (dependiendo del Servicio Web).
- El DM selecciona el método que quiere que se ejecute y le pasa los parámetros que se le piden (en nuestro caso, le pasa dos números para que haga operaciones básicas).
- El nodo ejecuta el Servicio Web y le manda los resultados al DM.
- Finalmente el DM muestra los resultados por el display y cierra el socket.

El nodo FreePastry por su parte, se ha implementado de modo que soporte la conexión de varios dispositivos móviles a la vez, así que continuará escuchando el puerto por defecto hasta que reciba alguna otra petición de servicio de algún DM.

4.3.1 Diseño del Servicio Web.

Para hacer las pruebas de funcionamiento del sistema, se ha desarrollado, a modo de ejemplo, un sencillo servicio web que consiste en las cuatro operaciones básicas que realiza una calculadora.

Tanto el servicio web en sí, como su consumo, se ha implementado mediante las APIs que nos proporciona el lenguaje Java para este tipo de servicios. El servicio web funcionará bajo un módulo del servidor Apache-Tomcat llamado Axis ref. [13].

A continuación vemos el código fuente de lo que representaría el servicio web en sí.

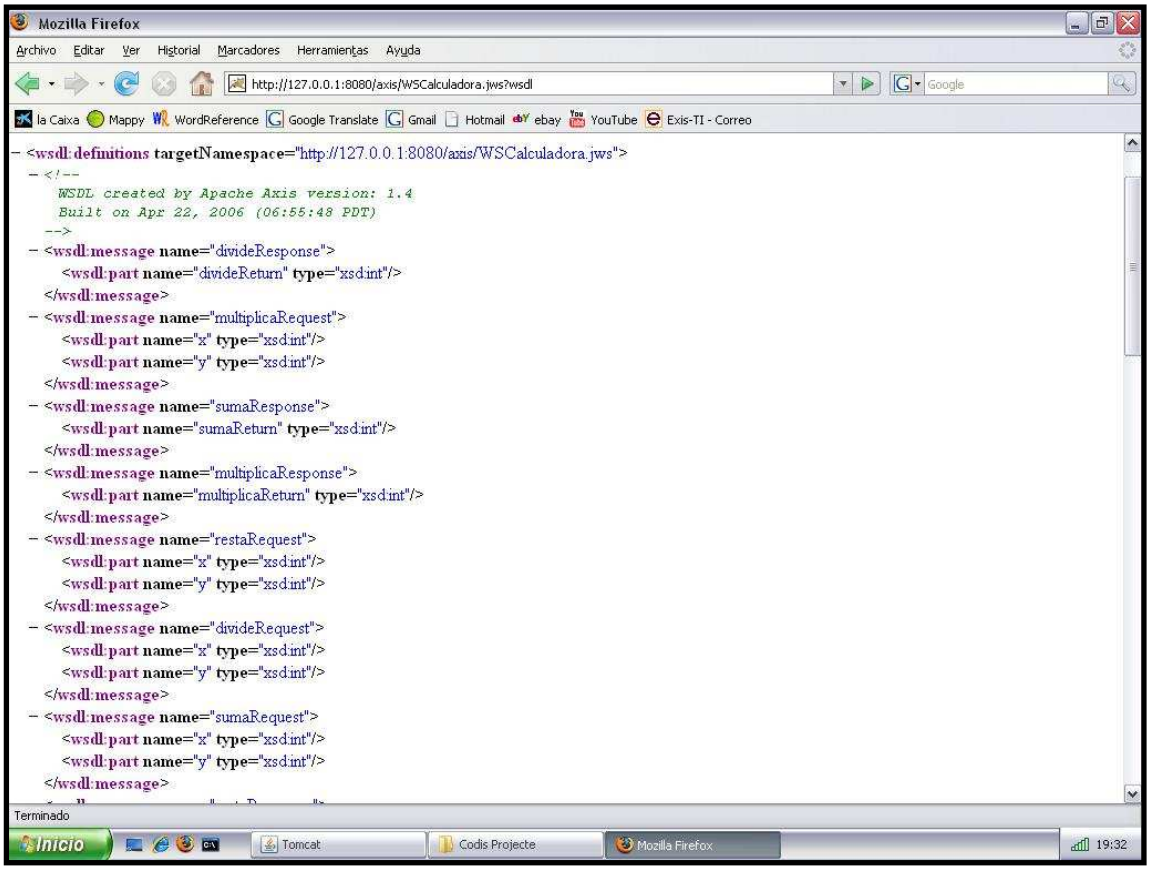
```
public class WSCalculadora {

    public int suma(int x, int y) {
            return x + y;
    }

    public int resta(int x, int y) {
            return x - y;
    }

    public int multiplica(int x, int y) {
            return x * y;
    }

    public int divide(int x, int y) {
        if(y==0)return -1;
        else return x / y;
    }
}
```

Fig. 16 – Servicio Web en Java.

Como podemos ver, este código es simplemente una clase de java normal y corriente. Una forma sencilla de convertirla en un servicio web entendible por Axis, es renombrar el fichero con la extensión “.java” a otro fichero con extensión “.jws” y colocarlo en el directorio de Axis. Éste directamente lo reconocerá y lo tratará como un servicio web construyendo su correspondiente fichero “.wsdl”.

Para comprobar el correcto funcionamiento del servicio web, nos basta con abrir un navegador y poner la siguiente URL:

http://127.0.0.1:8080/axis/WSCalculadora.jws?wsdl




```
- <wsdl:definitions targetNamespace="http://127.0.0.1:8080/axis/WSCalculadora.jws">
  - <!--
     WSDL created by Apache Axis version: 1.4
     Built on Apr 22, 2006 (06:55:48 PDT)
    -->
  - <wsdl:message name="divideResponse">
      <wsdl:part name="divideReturn" type="xsd:int"/>
    </wsdl:message>
  - <wsdl:message name="multiplicaRequest">
      <wsdl:part name="x" type="xsd:int"/>
      <wsdl:part name="y" type="xsd:int"/>
    </wsdl:message>
  - <wsdl:message name="sumaResponse">
      <wsdl:part name="sumaReturn" type="xsd:int"/>
    </wsdl:message>
  - <wsdl:message name="multiplicaResponse">
      <wsdl:part name="multiplicaReturn" type="xsd:int"/>
    </wsdl:message>
  - <wsdl:message name="restaRequest">
      <wsdl:part name="x" type="xsd:int"/>
      <wsdl:part name="y" type="xsd:int"/>
    </wsdl:message>
  - <wsdl:message name="divideRequest">
      <wsdl:part name="x" type="xsd:int"/>
      <wsdl:part name="y" type="xsd:int"/>
    </wsdl:message>
  - <wsdl:message name="sumaRequest">
      <wsdl:part name="x" type="xsd:int"/>
      <wsdl:part name="y" type="xsd:int"/>
    </wsdl:message>
```

Fig. 17 – WSDL del servicio web.

En la figura anterior se muestra una captura con el resultado que nos da el servicio web, se puede observar que se trata de su fichero .wsdl con los métodos implementados anteriormente.

4.3.2 Diseño del dispositivo móvil.

Para el desarrollo de aplicaciones en J2ME, existen varias herramientas que cumplen con este objetivo como por ejemplo, el "*J2ME Wireles Toolkit*" de Sun, el "*Nokia Developer's Suite for J2ME*" de Nokia o el "*Samsung Java SDK*" entre muchos otros.

La mayoría de estas herramientas son de libre distribución aunque podemos encontrar también varias soluciones propietarias.

Para el desarrollo de la parte del dispositivo móvil, se ha decidido utilizar el IDE de libre distribución Eclipse (ref. [11]) con su correspondiente plugin para el desarrollo en J2ME (ref. [12]) EclipseME.

A continuación se detallará paso a paso y con porciones de código la implementación del dispositivo móvil

El primer paso, es abrir un Socket TCP y un flujo de entrada para que el nodo le envíe el listado con los métodos del servicio web.

```
socket = (SocketConnection) Connector.open("socket://"+IP_ADDRESS+":"+PUERTO);
InputStream IS = socket.openInputStream();
flujo = new DataInputStream(IS);
```

Una vez el nodo FreePastry nos manda el listado de métodos del que dispone el servicio web, se procede a mostrarlo en el display como un menú seleccionable. Se muestran también los campos sobre los que queremos hacer la consulta.

```
lista = new ChoiceGroup("Menu de Servicios Web\n",List.EXCLUSIVE,menu,null);

num1 = new TextField ("Número 1:", "", 5, TextField.ANY);
num2 = new TextField ("Número 2:", "", 5, TextField.ANY);

form.append(lista);

form.append(num1);
form.append(num2);
```

Se implementan dos *commandAction*, uno para salir de la aplicación y el otro para enviar el método y los valores de los campos que hayamos seleccionado al peer.

En este último caso lo que hacemos es abrir un flujo de salida de datos por el socket que anteriormente habíamos abierto y enviamos todos los datos.

```
OutputStream os = socket.openOutputStream();
DataOutputStream flux = new DataOutputStream(os);

flux.writeUTF(menuSeleccionado);
flux.writeUTF(numero1);
flux.writeUTF(numero2);
```

Una vez enviado, es tarea del servidor llamar al servicio web, recoger el resultado y enviarlo de nuevo al dispositivo móvil. Éste mostrará en el display el resultado de la operación y cerrará el socket dando por terminada la comunicación.

```
TextField resul = new TextField("\nResultado:", flujo.readUTF(), 10, TextField.ANY);

form.append(resul);
socket.close();
```

En el siguiente apartado veremos en detalle la modificación del nodo FreePastry para llevar a cabo las tareas requeridas.

4.3.3 Modificación del nodo FreePastry

En primer lugar, hay que decir que la modificación está echa a base del código fuente de FreePastry en el apartado "*lesson3*" el cual el primer peer que se ejecuta crea el anillo y cuando se unen otros peers se intercambian mensajes entre ellos.

El nodo se ha implementado para que pueda soportar conexiones simultáneas de varios dispositivos móviles, esto se ha hecho mediante Threads.

Lo primero que hace el peer, es abrir un socket en modo escucha para esperar las conexiones de los dispositivos móviles.

```
ServerSocket socketServer = new ServerSocket(PUERTO);

Socket socket = socketServer.accept();
```

Una vez se ha conectado se procede a abrir un flujo de salida de datos y a enviar la lista de métodos implementados en el servicio web.

```
OutputStream aux = sock.getOutputStream();
DataOutputStream flujo= new DataOutputStream(aux);

flujo.writeUTF("suma");
flujo.writeUTF("resta");
flujo.writeUTF("multiplica");
flujo.writeUTF("divide");
```

Una vez enviado, esperaremos a que el dispositivo móvil nos devuelva la selección del método y los parámetros necesarios, es entonces cuando hacemos las peticiones al servicio web, pasándole la URL donde Axis lo tiene alojado.

```
Service  service = new Service();
Call call = (Call) service.createCall();

call.setTargetEndpointAddress( new java.net.URL(endpoint));
```

En este punto miramos los métodos que se van a invocar, establecemos los parámetros que necesita el método, información que se obtiene a partir del WSDL, especificamos también el tipo de datos que nos retornará el servicio web y por último invocamos al método.

```
call.getOperationName();

call.addParameter( "in0", XMLType.XSD_INT, ParameterMode.IN );
call.addParameter( "in1", XMLType.XSD_INT, ParameterMode.IN );

call.setReturnType( XMLType.XSD_INT );

result = (Integer) call.invoke( new Object [] { op1, op2 });
```

Una vez tiene el resultado sólo falta abrir otro flujo de datos hacia el dispositivo móvil y enviar el resultado obtenido.

```
OutputStream OS = sock.getOutputStream();
DataOutputStream res= new DataOutputStream(OS);

res.writeUTF(result+"");
```

4.3.4 FreePastry Launcher.

Por último, se ha diseñado un pequeño programa en Java Swing que lo que hace es arrancar automáticamente el servidor Tomcat y ejecutar el nodo FreePastry de manera que nos es más cómodo para no caer en el engorro de buscar los ficheros cada vez que queramos arrancar el sistema.

En La siguiente imagen se ilustra el resultado obtenido intuyéndose de este modo su funcionalidad.

Fig. 18 – FreePastry Launcher

En la siguiente imagen, observamos su funcionamiento, el servidor Tomcat arrancado y el peer en funcionamiento a la espera de la conexión de algún dispositivo móvil.

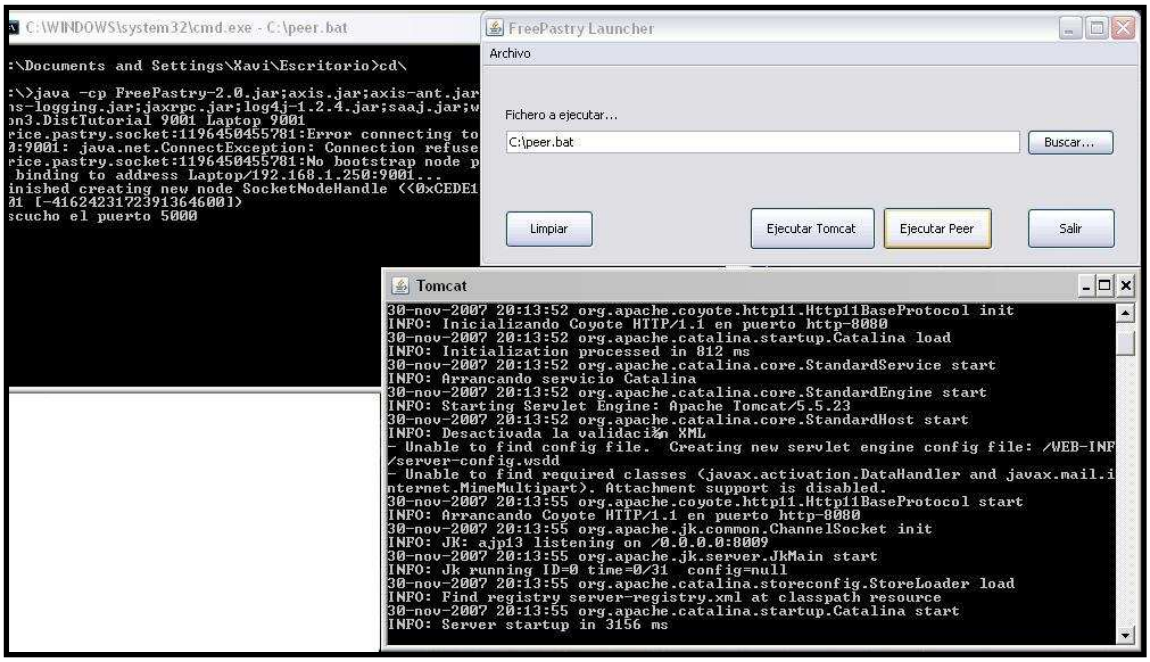


Fig. 19 – FreePastry Launcher en funcionamiento.

4.3.5 Funcionamiento global del sistema.

En este apartado se verá una demostración a base de capturas de pantalla del funcionamiento general del proyecto.

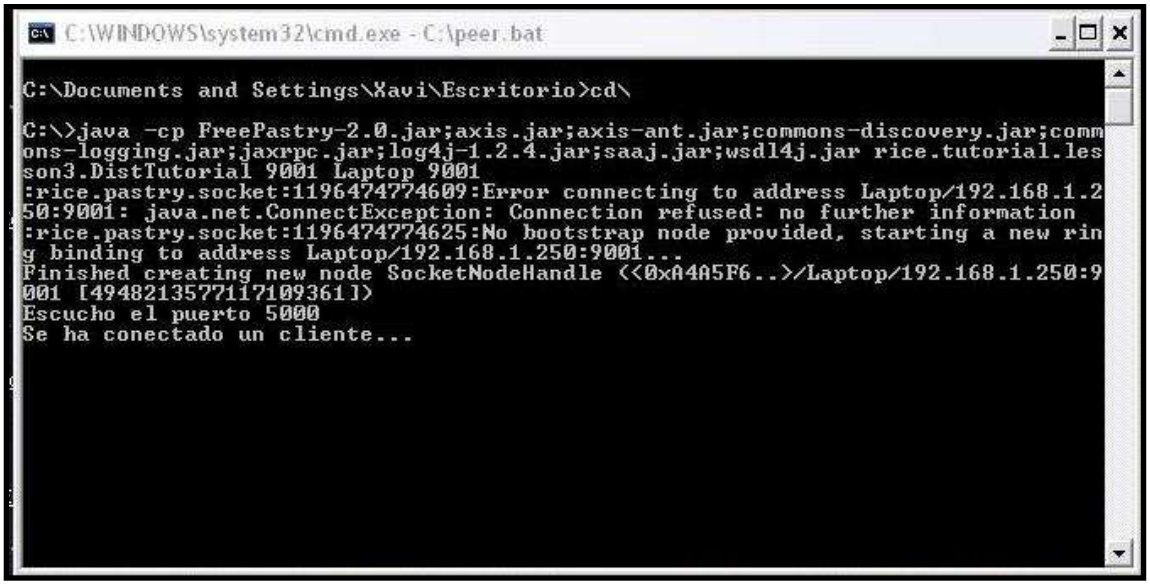


Fig. 20 – Conexión de un dispositivo móvil.

En la imagen anterior, se muestra la conexión al peer de un dispositivo móvil.

Fig. 21 -- DM con el menú recibido del Peer.

En esta captura, hemos visto como el peer le manda el listado con los métodos que tiene disponibles. Ahora se tiene que escoger uno de los métodos y pasarle los valores que necesita.

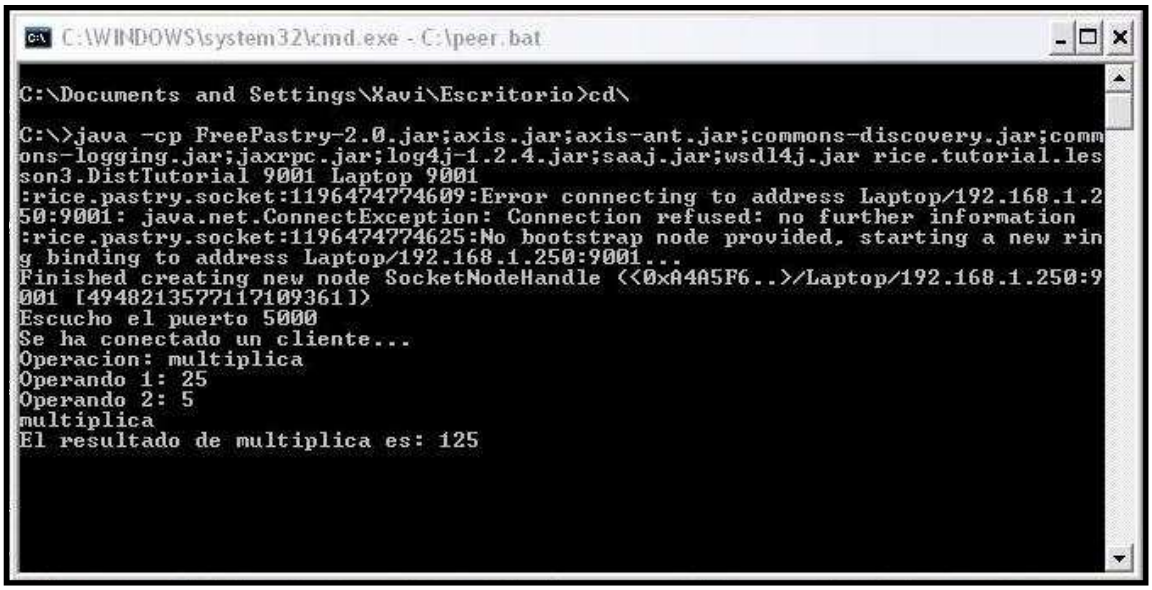


Fig. 21 – Peer accediendo al servicio web y enviando el resultado.

En este punto el peer invoca al servicio web con los valores que le hayamos pasado, recoge la respuesta y se la envía al dispositivo móvil el cual mostrará el resultado de la operación en el display.

Fig. 22 – Resultado final

CAPÍTULO 5: PLANIFICACIÓN Y COSTES

A continuación se detallará cual ha sido la planificación de este trabajo de fin de carrera. También se hará una estimación de los costes que suponen su realización tanto a nivel de personal como a nivel de material utilizado.

5.1 Planificación.

En el siguiente diagrama de Gantt se aprecia la planificación en tiempo de las fases de este proyecto, desde un estudio previo de requerimientos hasta la redacción de la memoria pasando por la implementación del mismo.

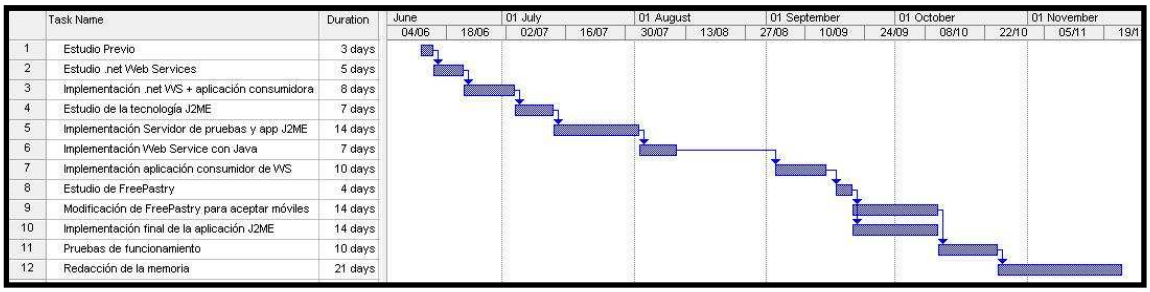


Fig. 20 – Planificación del proyecto

5.2 Costes de proyecto.

Para determinar los costes que han supuesto la realización de este proyecto, se han tenido en cuenta los siguientes puntos:

- *Tiempo de duración*: En este punto hago una estimación de las horas dedicadas al proyecto en estos 5 meses. TOTAL HORAS = 355.

- *Personal*: A nivel de personal, el coste de este trabajo supondrá el de un Ingeniero técnico de Telecomunicaciones a 24 Euros / Hora.

- *Herramientas y material*: Como todo el software utilizado es de libre distribución (eclipse, Axis, Freepastry, etc.) no ha supuesto ningún gasto, así que sólo contaré un PC y un dispositivo móvil.

PERSONAL	Nº Horas	€ / Hora	Nº Personas	Subtotal
Tiempo	355	24	1	**8520 €**

	TIPO	Cantidad	€ / unidad	Subtotal
EQUIPOS	PC Intel core2duo	1	1000	1000
	Dispositivo móvil	1	300	300
				1300

TOTAL	**9820 €**

Tabla 1. Costes del proyecto

CAPÍTULO 6: CONCLUSIONES

En este proyecto, he tenido la oportunidad de trabajar con tres de las tecnologías más punteras que existen actualmente en el marco de las comunicaciones. Antes de hacer la implementación definitiva, tuve que valorar varias tecnologías viendo así las ventajas e inconvenientes de cada una de ellas.

Por un lado las redes P2P, en concreto Pastry, me ha servido mucho para aprender muchos conceptos que desconocía y para aclarar algunos de los que ya había adquirido algún conocimiento a lo largo de la carrera.

También he tenido contacto con la tecnología de Servicios Web de las que no tenía ningún conocimiento y no sólo a nivel teórico, sino que he desarrollado uno sencillo para hacer las pruebas de funcionamiento del conjunto.

Por último y quizás lo más importante, la programación. He consolidado mis conocimientos de programación en Java cosa que me ha servido mucho en mi situación laboral actual. Además he tenido la ocasión de probar otra tecnología y aprender conceptos en otro lenguaje de programación, C# sobre la plataforma .net, con lo que he podido ver, las diferencias, similitudes, ventajas en inconvenientes de cada uno de los dos lenguajes.

6.1 Líneas futuras.

A la velocidad que avanzan los fabricantes de hardware en materia de dispositivos móviles, que cada vez son más potentes los que salen al mercado, no sería de extrañar que en un corto periodo de tiempo saliera algún DM con capacidad de proceso y memoria suficientes como para poder soportar la arquitectura detallada en un capítulo anterior, poner el DM como un nodo más dentro de la red FreePastry.

Otra posible mejora, sería en ir un paso más en al modificación del nodo para que descubriera automáticamente los Servicios Web y le pasara al DM un listado con todos los que ha encontrado y que este escogiera cual quiere consumir.

6.2 Impacto medioambiental.

Aunque a simple vista parece que no tiene mucho sentido reparar en el impacto medioambiental dadas las características de este proyecto, si nos paramos a reflexionar, nos damos cuenta que si que existe algún impacto.

Hay algunos aspectos negativos a tener en cuenta:

- Cada vez hay más demanda de nuevas tecnologías, esto ocasiona que cada vez se desarrollen mejores y más potentes aplicaciones que necesitan de medios físicos (tanto fijos como móviles) más potentes, esto provoca que el ciclo de vida de estos medios sea bastante corto y

necesiten renovarse de forma continua favoreciendo el aumento de los residuos.

- La mayor parte del soporte físico relacionado con las infraestructuras de las telecomunicaciones, suelen ser plásticos y otros materiales no biodegradables.

- Los cientos de millones de teléfonos móviles así como las antenas de las redes celulares repartidas por todo el mundo, emiten ondas electromagnéticas que favorecen al calentamiento global.

CAPÍTULO 7: REFERENCIAS

[1] *P2P Networking Overview, http://www.oreilly.com/catalog/p2presearch/index.html.*

[2] Wikipedia *http://es.wikipedia.org/wiki/Peer-to-peer#Caracter.C3.ADsticas.*

[3] FreePastry, *http://freepastry.rice.edu/*

[4] World Wide Web Consortium. *http://www.w3c.com/*

[5] Informe técnico Universidad de Castilla la Mancha
http://www.info-ab.uclm.es/descargas/thecnicalreports/DIAB-05-01-1/Servicios%20Web.pdf

[6] Eric Newcomer, *Understanding Web Services*.

[7] ebook – Guía del desarrollador de Servicios Web XML (*www.borland.com*)

[8] Microsoft .NET Compact Framework.
(http://msdn2.microsoft.com/es-es/library/f44bbwa1(VS.80).aspx)

[9] Sergio Gálvez Rojas J2ME (Java 2 Micro Edition)
(http://www.google.es/books?id=USaAQ0hHQWIC&printsec=frontcover&dq=j2me&sig=E69gZ5TZKPZtRfRY1gaHXvz0vZo)

[10] Kim Topley, O'Reilly -- J2ME in a Nutshell.

[11] Página oficial Eclipse -- www.eclipse.org

[12] Plugin eclipse para J2ME -- http://eclipseme.org/docs/installEclipseME.html

[13] Axis Web Services -- http://ws.apache.org/axis/