INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS MONTERREY

PROGRAMA DE GRADUADOS EN ELECTRÓNICA,

COMPUTACIÓN, INFORMACIÓNY COMUNICACIONES

SERVICIOS DE BIBLIOTECAS DIGITALES A TRAVÉS DE UNA ARQUITECTURA PUNTO A PUNTO EN DISPOSITIVOS MÓVILES.

TESIS

PRESENTADA COMO REQUISITO PARCIAL PARA OBTENER EL GRADO ACADEMICO DE:

MAESTRO EN CIENCIAS EN
TECNOLOGÍA INFORMÁTICA

POR:

RODRIGO RUIZ BACA

MONTERREY, N.L. **DICIEMBRE 2005**

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

DIVISIÓN DE ELECTRÓNICA, COMPUTACIÓN, INFORMACIÓN Y COMUNICACIONES

PROGRAMAS DE GRADUADOS EN ELECTRÓNICA, COMPUTACIÓN, INFORMACIÓN Y COMUNICACIONES

Los miembros del comité de tesis recomendamos que la presente tesis del Lic. Rodrigo Ruiz Baca sea aceptada como requisito parcial para obtener el grado académico de Maestro en Ciencias en Tecnología Informática.

Comité de tesis:

_____

Dra. Lorena Gómez Martínez
Asesor Principal

_____

Dr. Juan Carlos Lavariega Jarquin
Sinodal

_____

Dr. David Garza Salazar
Sinodal

_____

Dr. David Garza Salazar
Director del Programa de Graduados en Electrónica, Computación, Información y Comunicaciones.

Diciembre de 2005

SERVICIOS DE BIBLIOTECAS DIGITALES A TRAVÉS DE UNA ARQUITECTURA PUNTO A PUNTO EN DISPOSITIVOS MÓVILES.

POR:

RODRIGO RUIZ BACA

TESIS

Presentada al Programa de Graduados en Electrónica, Computación, Información y Comunicaciones.

Este trabajo es requisito parcial para obtener el grado de Maestro
En Ciencias en Tecnología Informática.

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS

SUPERIORES DE MONTERREY

Diciembre de 2005

Abstract

La presente tesis presenta el diseño, desarrollo y evaluación de un ambiente punto a punto para dispositivos móviles ofreciendo servicios de bibliotecas digitales llamado Peer To Peer Digital Library (P2PDLib). P2PDLib nace como la solución a algunas de las necesidades de un proyecto anterior de bibliotecas digitales en dispositivos móviles llamado PDLib [17]. PDLib es una biblioteca digital creada tanto para dispositivos móviles como para computadoras de escritorio o laptops que utiliza una arquitectura cliente/servidor, es decir que todos los clientes dependen de un servidor central para poder operar. P2PDLib ofrece servicios tales como envío y recepción de documentos, envío y recepción de metadatos de documentos, chat, entre otros. Para que P2PDLib funcione, es necesario que dentro del dispositivo móvil exista una copia del esqueleto de la biblioteca digital que se encuentra en el servidor de datos de PDLib, es decir las colecciones y los metadatos de documentos. P2PDLib está formado básicamente por dos entidades: la aplicación que se tendrá en los dispositivos móviles (cliente) y un punto privilegiado (PP) que es un punto con más recursos de cómputo que un dispositivo móvil y que sirve para proveer a los clientes de la red punto a punto de algunos servicios.

P2PDLib resuelve algunos de los problemas típicos de tener un servidor centralizado, tales como ¿Qué pasa si el servidor se cae?, ¿Qué pasa si quiero que otros usuario tengan mis documentos o conozcan parte de mi biblioteca digital?. Para resolver este tipo de cuestiones, se crea un ambiente punto a punto, donde todos los usuarios de PDLib estén conectados entre sí de tal manera que todos lo puntos puedan compartir documentos y “chatear”, entre otras cosas.

Esta tesis aporta a la comunidad informática una investigación y experimentación de la tecnología punto a punto en dispositivos móviles ofreciendo servicios de bibliotecas digitales. Ofrece también una investigación del uso de J2ME, XML y tecnología punto a punto, la cual es prácticamente nueva en dispositivos móviles, y debido a la implementación de la misma a través de un prototipo se prueba que lo anterior es posible, así como también se demuestra el poder y eficiencia de J2ME como lenguaje de programación y XML para transferencia y almacenamiento de datos.

Capítulo 1

1.1 Introducción

Una de las más recientes tendencias en el campo del intercambio de información son los sistemas punto a punto (P2P). En un modelo P2P, los sistemas están compuestos de una red de puntos dinámica, los cuales comparten datos, recursos computacionales, etcétera, los puntos son por lo regular autónomos o semiautónomos y trabajan en conjunto en la ejecución de trabajos o en el almacenamiento y consulta de recursos.

Los sistemas P2P son muy populares debido a que permiten el intercambio de información de una manera ad-hoc y sin una entidad central de supervisión. Este tipo de características son benéficas para las comunidades de usuarios móviles que tienen la necesidad de intercambiar información de interés común. Una red pura punto a punto tiene por lo regular usuarios móviles que están conectados a través de una red inalámbrica, debido a esto los puntos o pares de la red están en constante movimiento. Las ventajas principales de una red punto a punto son su escalabilidad debido a la capacidad de compartir recursos entre los puntos, tolerancia a fallos debido a la simetría natural de los pares y su robustez debido a su autoorganización. Existen dos tipos de redes punto a punto, las estructuradas en donde todos los puntos mantienen en una tabla la dirección de sus vecinos más cercanos. Para un ruteo eficiente cada nodo mantiene O (log n) (donde n es el número de nodos en la red) apuntadores a los nodos vecinos, y las redes no estructuradas donde los nodos no tienen la capacidad de conocer a sus vecinos debido a entre otras cosas, su movilidad. Para los segundos tipos de redes es necesario contar con alguna entidad que pueda conocer de alguna forma la dirección de todos lo nodos y que así, pueda mantener a todos los puntos en contacto.

Una arquitectura punto a punto en dispositivos móviles puede presentar algunos problemas tales como desconexiones constantes, bajo ancho de banda para la transferencia de recursos de gran tamaño, dificultad para la búsqueda de recursos en los puntos, difícil identificación de los puntos, dificultad para sincronizar información que cambia en un nodo, seguridad.

Los dispositivos móviles se han vuelto cada vez más populares, hoy en día cada vez hay más usuarios de esta tecnología y no es de extrañarse que las aplicaciones que por lo regular se utilizan cotidianamente en ambiente “fijos” se deseen utilizar también en ambientes móviles, las bibliotecas digitales son un ejemplo de este tipo de aplicaciones, que por lo regular se utilizan para consultar información a través de una red fija haciendo uso de una computadora de escritorio, sin embargo no es de extrañarse que haya usuarios que deseen tener estas capacidades en sus PDA’s o celulares.

Esta tesis trata sobre la creación de un sistema punto a punto en dispositivos móviles ofreciendo servicios de bibliotecas digitales basándose en el proyecto PDLib [17] del tecnológico de Monterrey. PDLib es una biblioteca digital universalmente disponible para dispositivos móviles. La investigación aquí realizada describe la investigación de los servicios de bibliotecas digitales que pueden ofrecerse a través del uso de la tecnología punto a punto en dispositivos móviles, analiza los problemas que se tienen y presenta a las soluciones que pueden ser implementadas y para esto, se apoya de un prototipo funcional que demuestra la factibilidad del ambiente punto a punto. Aunque en el capítulo cuatro se hablará a detalle del prototipo, es necesario mencionar a grandes rasgos los servicios que ofrecerá. Los servicios son: mostrar el esqueleto de la biblioteca digital previamente obtenidos del servidor de datos de PDLib (las colecciones, subcolecciones y metadatos de documentos), transferencia de documentos entre puntos de la red, búsqueda de recursos en la red punto a punto, indexamiento de los esqueletos de los puntos, creación y transferencias de vistas parciales del esqueleto de la biblioteca.

1.2 Objetivo General

Elaborar y validar la factibilidad de ofrecer servicios de bibliotecas digitales a través de dispositivos móviles utilizando la tecnología punto a punto.

1.3 Objetivos Específicos

1. Identificar los servicios de PDLib que pudieran ofrecerse a través de tecnología punto a punto.
2. Identificar la información de PDLib que debe mantenerse en el dispositivo móvil para ofrecer servicios punto a punto.

3. Diseñar una estructura apropiada de un dispositivo móvil para almacenar los datos requeridos en el dispositivo a fin de satisfacer servicios punto a punto.
4. Identificar los metadatos de documentos de una biblioteca personal contenidos en el dispositivo móvil.
5. Construir un prototipo funcional usando tecnología punto a punto que ofrezca los servicios de PDLIB identificados en el punto 1.
6. Evaluar la eficiencia y viabilidad de los servicios PDLib punto a punto.
7. Investigar tecnologías punto a punto para dispositivos móviles.
8. Diseñar un mecanismo que permita la comunicación punto a punto.

1.4 Problemas a atacar

1. Reducir la dependencia al servidor de datos, de tal forma que cuando un usuario no pueda conectarse, pueda por lo menos contar con los servicios de bibliotecas digitales implementados en el prototipo aquí presentado. El servidor de datos es el encargado de almacenar las bibliotecas digitales de los usuarios de PDLib.
2. Compartir documentos entre dispositivos, proveyendo así a cada usuario con la posibilidad de enviar y recibir documentos contenidos dentro del dispositivo.
3. Compartir metadatos entre dispositivos con la finalidad de proveer a los usuarios de P2PDLIb con la funcionalidad de conocer de forma parcial, que colecciones, subcolecciones o documentos existen en las bibliotecas digitales de otros usuarios de PDLib, ya que hasta ahora esto no es posible.
4. Facilitarle al usuario la búsqueda de documentos en el servidor de PDLib. Cada vez que un usuario desea realizar alguna búsqueda a través de metadatos, es necesario conectarse al servidor de datos de PDLib, por lo que con este servicio se puede eliminar el acceso al servidor de datos.
5. Poder contar respaldos parciales de una biblioteca digital. Si un usuario por alguna razón desea recuperar o simplemente ver su biblioteca digital cuando el servidor de datos tiene algún problema o su biblioteca sufrió algún daño, el esqueleto es almacenado en el dispositivo móvil y puede ser consultado en cualquier momento.

1.5 Servicios a Ofrecer

1. Creación y envío de vistas parciales. Cada usuario de P2PDLib cuenta con la posibilidad de crear vistas parciales de su esqueleto de la biblioteca digital (colecciones y metadatos de documentos). Una vista parcial es una parte del esqueleto de la biblioteca digital contenida en el dispositivo móvil, es decir en un subconjunto de ella. Una vista parcial puede contener una o varias colecciones (con todo y sus metadatos) del esqueleto de la biblioteca digital.
2. Envío y recepción de las vistas previamente descritas. Una vez que el usuario creó un vista, esta puede ser enviada o recibida para dar a conocer y/o conocer parte de la biblioteca en cuestión
3. Envío y recepción de documentos de otros usuarios. Si un documento fue obtenido del servidor de datos de PDLib y se encuentra físicamente en el dispositivo móvil, este puede ser enviado y/o recibido por otros usuarios a través de una comunicación punto a punto.
4. Chat. Cada usuario puede establecer una conversación tipo Chat con uno o más usuarios conectados a la red punto a punto.
5. Navegación en la biblioteca digital local. Debido a que cada usuario cuenta con el esqueleto de su biblioteca digital en el dispositivo móvil, es posible navegar a través de ella siguiendo una estructura tipo árbol, en donde las colecciones contienen otras colecciones y documentos.
6. Búsqueda de recursos (documentos principalmente) a través de un manejador de base de datos y haciendo uso del punto privilegiado.

1.6 Organización de la tesis

La presente tesis esta dividida en 5 capítulos, el segundo capítulo describe los antecedentes, tecnologías de trabajo y el trabajo previo realizado en otras investigaciones, en el tercer capítulo se habla de la arquitectura propuesta o ideal para resolver el problema principal de la tesis, en el capítulo cuatro se presenta el prototipo utilizado para la experimentación y obtención de resultados y por último en el capítulo cinco se presentan las conclusiones y el trabajo futuro.

Dedicatorias

A mis papás, por ayudarme y apoyarme en la realización de todos los proyectos importantes en mi vida.

A mi esposa, por estar siempre a mi lado apoyándome en todos mis proyectos.

A mis abuelos, por velar por mí en donde quiera que estén.

A mis abuelas, por su amor.

Gracias.

Agradecimientos

A Dios, por nunca abandonarme, por apoyarme y por estar siempre a mi lado en todos los momentos de mi vida.

A mi familia, por alentarme para el logro de mis metas.

A mi tía Laura, por que sin su ayuda tal vez no hubiese podido lograr esta meta.

A mi asesora, por guiarme en la realización de esta tesis.

A mis amigos, por su apoyo incondicional.

Gracias.

Índice General

Índice de Figuras

2 Fundamento Teórico

En éste capítulo se introducirá el tema de bibliotecas digitales en la sección 2.1, qué son y una breve descripción de cómo funcionan. Se hablará asimismo, de lo que es el proyecto PDLib[17] dentro de la sección 2.2, parte fundamental de esta tesis, debido a que la solución presentada será precisamente parte del proyecto y fungirá como un servicio más del mismo. En la sección 2.3 se presentará la tecnología punto a punto que fue utilizada, describiendo sus funcionalidades básicas, ventajas y desventajas sobre otras tecnologías, así como los servicios que ofrece, así como también se hablará de lo que es la comunicación P2P y sus características. Dentro de la sección 2.4 se presenta JXTA debido a que es un conjunto de protocolos para el desarrollo de aplicaciones punto a punto. En la sección 2.5 se hablará también sobre el lenguaje de programación Java para ambientes móviles, sobre el cual estará desarrollado el prototipo. Dentro de la sección 2.6 se hablará sobre lo que son los sockets y los tipos que existen, ya que la comunicación punto a punto aquí implementada hace uso de ellos. Se hablará también de que es y para que sirve el lenguaje de marcación XML en la sección 2.7, debido a que la transferencia de metadatos y su almacenamiento en el dispositivo móvil serán a través de este lenguaje. Dentro de la sección 2.8 se introduce MYSQL, debido a que será utilizado dentro del prototipo aquí presentado y finalmente en la sección 2.8 se describen brevemente los trabajos existentes enfocados a la comunicación punto a punto.

2.1 Bibliotecas Digitales

Hoy en día existe en el mundo una enorme cantidad de información y conocimiento. El gran reto es poder hacer accesible dicha información de una manera eficiente y rápida para así poder obtener información relevante y aplicarla a la solución de un problema. Desde hace mucho tiempo las bibliotecas han sido tradicionalmente los grandes repositorios de información. El día de hoy las bibliotecas se han ayudado de computadoras para poder hacer más rápida la búsqueda y recuperación de información. Algunas han ido más allá y cuentan con una cantidad considerable de material digital. Las bibliotecas

digitales proveen hoy en día acceso en línea a un número muy grande de recursos de información en texto y multimedia de una manera integrada [12].

Los recursos de una biblioteca digital incluyen textos, figuras, fotografías, sonido, video, filminas, diapositivas, etcétera. El tamaño de los repositorios de información y datos disponibles es enorme. Las fuentes de datos son distribuidas y heterogéneas y los servicios de una biblioteca digital deben proveer una interfase uniforme para el acceso a la información transparente [13].

Una biblioteca digital es una colección de documentos almacenados en un formato electrónico [7]. Una biblioteca digital ofrece servicios tales como, envío, clasificación, indexación, búsquedas y obtención de documentos digitales.

2.1.1 Bibliotecas digitales en ambientes móviles

Los usuarios de bibliotecas digitales no solo son aquellos que cuentan con una red local fija (LAN) o con una red amplia (WAN), sino que también hay usuarios móviles con redes inalámbricas, sin embargo las conexiones móviles en general son caracterizadas por sus desconexiones frecuentes, limitado poder de cómputo y memoria, el tamaño de la pantallas por lo regular es muy pequeño, así como diferentes tipos de conexiones: Conexión completa, conexión parcial y conexión intermitente por ahorro de energía. Muchos retos pueden surgir cuando se intenta unir a las tecnologías de cómputo móvil y a las bibliotecas digitales en un solo proyecto, por un lado una biblioteca digital cuenta con grandes volúmenes de recursos y datos, y que además cada uno de estos recursos son frecuentemente grandes en tamaño, y por el otro lado los dispositivos móviles son muy limitados en poder de cómputo, por tal motivo el poder ofrecer servicios de bibliotecas digitales en dispositivos móviles involucra modificar la naturaleza de dichos servicios para adaptarlos a este tipo de dispositivos.

2.2 PDLib

El proyecto de software PDLib [17] propone una arquitectura para bibliotecas digitales con soporte para usuarios móviles. Un usuario móvil es alguien que desea conectarse a través de su teléfono celular, palm, pocket pc, table pc, o cualquier otro dispositivo a la biblioteca digital. En la figura 2.1 se pueden observar de manera resumida las características de la arquitectura de PDLib.

El proyecto PDLib propone un sistema de biblioteca digital personal, universalmente disponible. Es personal en el sentido de que cada usuario es proveído de un repositorio de documentos de propósito general. Es universalmente disponible en el sentido que permite a cada usuario acceder a su biblioteca digital desde casi cualquier dispositivo de cómputo conectado a la red, incluyendo teléfonos móviles, PDA'S y laptops.

Las bibliotecas digitales personales proveen los servicios básicos de las bibliotecas digitales tradicionales, tales como agregar documentos, indexación de texto completo y metadatos y búsquedas y obtención de documentos. Los metadatos son datos sobre los datos, esto es, información sobre la información misma. En esencia, intentan responder a las preguntas quién, qué, cuándo, cómo, dónde y porqué, sobre cada una de las facetas relativas a los datos que se documentan. Para este contexto los metadatos se refieren a información descriptiva sobre los documentos y colecciones de la biblioteca digital.

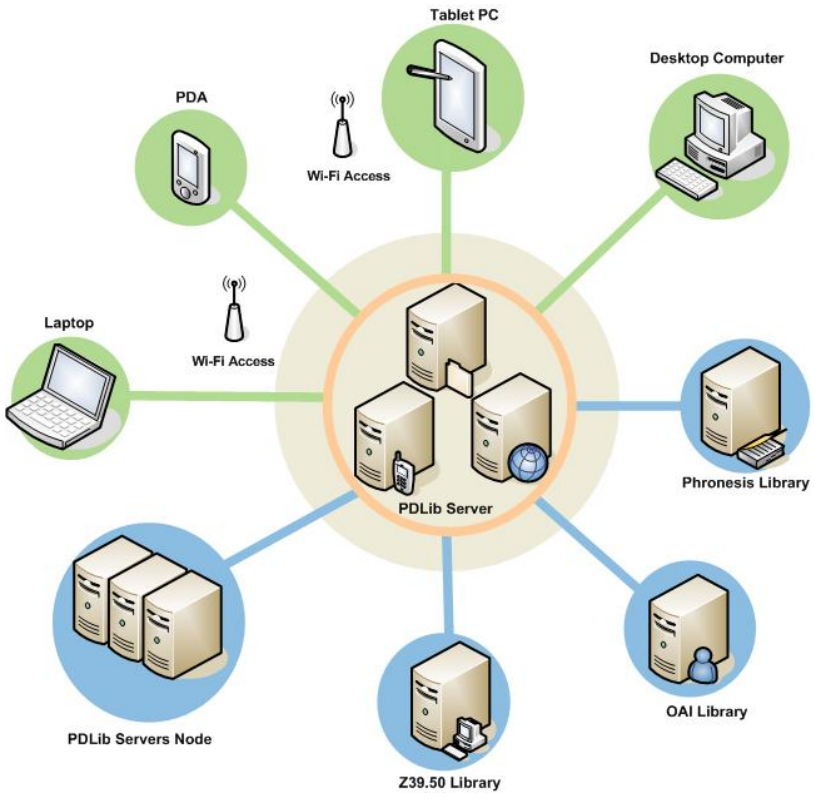


Figura 2.1 Arquitectura general de PDLib.

2.2.1.1 Bibliotecas digitales universalmente disponibles.

Los sistemas de bibliotecas digitales tradicionales conceden permisos a un grupo de usuarios para tener acceso a una biblioteca digital. Los sistemas de bibliotecas digitales personales proveen a cada usuario de una biblioteca digital. PDLib es un sistema de biblioteca personal digital que permite al usuario formar y acceder a su biblioteca digital desde cualquier lugar en cualquier momento. Para lo anterior se definieron los siguientes requerimientos:

- *Colección flexible y administración de metadata:* Las colecciones deben ser proveídas como un mecanismo de clasificación de documentos. Los usuarios deben tener la posibilidad de definir los conjuntos de metadatos que serán usados para describir su colección.
- *Presentación de documentos digitales:* Los usuarios deben tener la capacidad de subir documentos digitales desde cualquier dispositivo móvil.
- *Búsquedas y obtención:* Los mecanismos de búsquedas y obtención deben ser adaptados al esquema de clasificación personalizado definido por el usuario del sistema.
- *Acceso universal:* Acceso universal se refiere que un usuario puede acceder a su biblioteca desde cualquier parte del mundo y utilizando cualquier dispositivo de cómputo conectado a internet. Para ofrecer un acceso universal a documentos y colecciones, se utilizan diferentes tipos de aplicaciones cliente. Los programas cliente pueden ser clasificados de acuerdo a su arquitectura en: clientes ligeros y clientes densos y de acuerdo a su movilidad en: clientes fijos y clientes móviles. En los clientes ligeros, la aplicación es presentada en un navegador o micro navegador, mientras que en los clientes densos el código y los datos residen en el dispositivo. Los clientes móviles son aquellos que cuentan con una conexión a la red inalámbrica y los clientes fijos son aquellos que cuentan con un acceso alámbrico a la red.
- *Administración y control de acceso:* El dueño de una biblioteca digital tiene acceso ilimitado a sus contenidos y, además la posibilidad de conceder a otros usuarios acceso (a otros usuarios) a dichos contenidos.

- *Interoperabilidad:* Con otros usuarios de la biblioteca personal y con otros sistemas de bibliotecas digitales haciendo uso de protocolos bien conocidos. Interoperabilidad se refiere a la capacidad que tienen dos o más sistemas de interactuar entre sí de forma transparente.

Las bibliotecas digitales personales están compuestas por colecciones, las cuales a su vez contienen otras colecciones y documentos.

Los servicios proveídos por una biblioteca personal digital son:

- *Operaciones CRUD (por sus siglas en inglés):* Operaciones de creación, obtención, actualización y borrado para la administración de de librerías, colecciones, documentos, formatos de documentos y conjuntos de objetos de metadatos.
- *Copiar/Mover documentos o colecciones:* Los procesos de copiar y mover documentos (son proveídos) se ofrecen para duplicar o relocalizar documentos y colecciones de objetos.
- *Búsqueda de documentos:* Búsquedas boleanas y por ranking de documentos según su texto completo o metadatos (son proveídos).
- *Envío de documentos:* Transferencia de documentos a una biblioteca personal de otro usuario.
- *Envío de documentos a través de correo electrónico:* Envío de documentos y metadata a través del correo electrónico.
- *Conversión de formatos de documentos:* Permitir la conversión de formatos en documentos, por ejemplo: texto plano a formato pdf.
- *Obtener/Establecer conjuntos de metadatos de colecciones:* Obtener el conjunto metadatos de una colección específica o asignarle a una colección un conjunto de metadatos.
- *Actualizar metadatos de documentos:*
- *Obtener/Establecer Obtención/Establecer permisos de colecciones:* Permitirle al usuario dar o quitar permisos a otros usuarios sobre sus colecciones y/o documentos.

2.2.1.2 Arquitectura del sistema

El principal componente de PDLib es el servidor de datos; éste es el encargado de almacenar los objetos de las bibliotecas personales, además de proveer a las aplicaciones cliente con un módulo de comunicación que permite el acceso remoto al contenido de una biblioteca digital.

La arquitectura de PDLib soporta, además, la iteración con otros sistemas de bibliotecas digitales a través de OAI-PMH [26].

El sistema se compone de tres capas:

- *Capa cliente:* Incluye toda la variedad de dispositivos con los que el usuario debe interactuar a través de PDLib.
- *Capa servidor:* Es la infraestructura del sistema servidor que deberá proveer servicios a los Clientes: Servidor de datos, MCM (middleware de adaptación para la conexión de dispositivos móviles con el servidor de datos), interfase web.
- *Capa de interoperabilidad:* Incluye otros servidores de datos y otros sistemas de bibliotecas digitales compatibles con OAI-PMH.

En la figura 2.2 se puede apreciar de forma gráfica las capas antes mencionadas y la relación entre ellas.

La arquitectura de PDLib refleja los siguientes tipos de acceso:

- *Acceso a través del middleware:* Soporta dispositivos móviles, especialmente aquellos con recursos limitados.
- *Acceso Web:* Provee acceso HTTP a cualquier dispositivo que cuente con un navegador de internet.
- *Acceso directo:* Aplicaciones con requerimientos muy específicos pueden acceder directamente al servidor de datos.

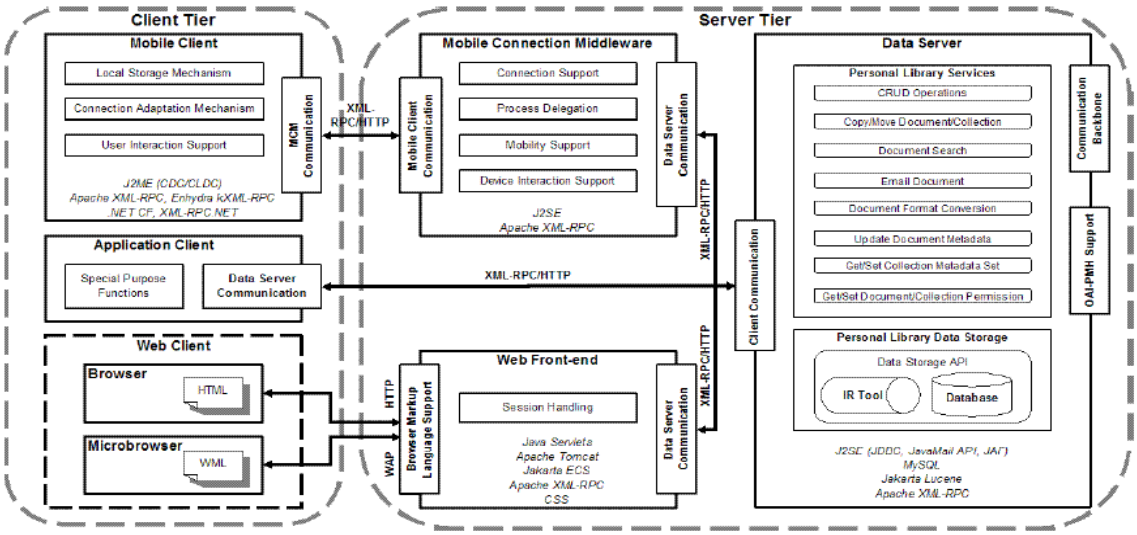


Figura 2.2 Arquitectura de PDLib por componentes

2.2.1.3 Clientes

El software cliente del ambiente móvil puede ser clasificado de acuerdo a "la arquitectura del lado del cliente" y a su movilidad.

De acuerdo a la arquitectura del lado del cliente: Las aplicaciones ligeras se conectan al servidor de datos a través de una interfaz Web, mientras que los clientes densos se comunican con el servidor de datos a través del MCM.

De acuerdo a su movilidad: Las aplicaciones móviles se conectan con la capa servidor a través de conexiones inalámbricas no confiables mientras que los clientes fijos se conectan a través de conexiones fiables alámbricas.

Se definieron los siguientes tipos de clientes:

- *Clientes móviles:* Los clientes móviles fueron designados para hacer frente a las limitaciones de los ambientes móviles. Se requiere de una capa intermedia (middleware) para ayudar a los clientes móviles a alcanzar sus objetivos de conexión móvil.
- *Clientes Web:* Esta categoría incluye todos aquellos dispositivos con un navegador de internet o micro navegador capaz de desplegar o interpretar HTML.
- *Aplicaciones cliente (clientes fijos):* Las aplicaciones cliente son diseñadas para correr en una computadora de escritorio o en una computadora personal. Además de

que estas aplicaciones pueden comunicarse directamente con el servidor de datos (directamente).

En la figura 2.2 se puede apreciar de manera gráfica los tipos de clientes que existen y la forma en que se comunican con la capa de servicio.

Uno de los principales retos del proyecto reside en los clientes móviles, ya que éstos deben enfrentarse a los problemas de las limitaciones de un ambiente móvil. Para cubrir sus tareas, los clientes móviles deben tener la posibilidad de llevar a cabo las siguientes funciones:

- *Mecanismo de almacenamiento local:* Se refiere a poder almacenar los documentos de la biblioteca digital en el dispositivo móvil, para su consulta fuera de línea.
- *Mecanismo de adaptación a la conexión:* Este mecanismo provee un tiempo de respuesta constante defiendo las variaciones en las conexiones inalámbricas.
- *Soporte para interacción de usuario:* Una interfaz gráfica que le permita al usuario administrar su biblioteca digital desde su dispositivo móvil.

La presente tesis esta enfocada precisamente a la parte de los dispositivos móviles, tratando de resolver los problemas de comunicación con el servidor de datos eliminado de manera parcial la dependencia a él, para lo cual se propone un ambiente de comunicación punto a punto para el envío y recepción de documentos almacenados en los dispositivos, también se considera el tener del lado del dispositivo una copia de la estructura de la biblioteca digital (esqueleto de la biblioteca digital), para dos principales cosas: Tener un respaldo de por lo menos la estructura y algunos documentos y el poder compartir colecciones con otros usuarios. Sin embargo cabe señalar que en esta última modalidad lo que se comparte es solamente una copia de la estructura lógica de alguna colección, es decir, nombre y metadatos pero no los documentos contenidos en ellas, a menos que estos se encuentren físicamente en el dispositivo.

2.2.1.4 Interfaz Web.

La interfaz Web transforma los servicios de una biblioteca digital personal en una aplicación web. Esta interfaz debe (de) proveer las siguientes funcionalidades:

- *Manejo de sesiones:* El manejo de sesiones es usado para mantener la interacción de un cliente delgado con la interfaz Web más allá de una simple petición HTTP.

- *Soporte para navegación en lenguajes de marcación de texto:* Verifica que el navegador utilizado soporte lenguajes de marcación.

2.2.1.5 Capa de conexión móvil (MCM).

Uno de los principales problemas a resolver para proveer los servicios del servidor de datos es el hecho de que éste fue diseñado para ser usado por dispositivos que hacen uso de redes de banda ancha y no dispositivos móviles.

Existen diferencias muy claras de recursos de computación entre dispositivos móviles (teléfonos celulares, PDA's etc.) y fijos (computadoras de escritorio, laptops, etc.). Esta diferencia hace muy difícil la adaptación del servidor de datos a dispositivos móviles, por lo que es necesario contar con una capa intermedia para mediar la interacción entre los dispositivos móviles y el servidor de datos.

Para solucionar este problema se diseñó lo que se llama MCM (Mobile Connection Middleware), que debe contar con las siguientes funcionalidades:

- *Soporte de conexión:* Es requerido por los dispositivos móviles para enfrentar y adaptarse a los grandes cambios de anchos de banda que existen en un ambiente móvil.
- *Proceso de delegación:* Ejecuta procesos que demandan muchos recursos de cómputo y que, por lo tanto, no podrían ser realizados por un dispositivo móvil.
- *Soporte para movilidad:* Se refiere a la capacidad de realizar operaciones tales como "prefetching" para aumentar la velocidad de obtención de documentos del servidor de datos, almacenándolos en servidores de cache que se encuentren más cerca del usuario.
- *Soporte de interacción de dispositivos:* Provee de adaptación al contenido, de acuerdo a las características del dispositivo en cuestión.

2.2.1.6 Servidor de datos.

Es la parte principal de PDLib. El servidor de datos provee los servicios de una biblioteca digital personal, almacena los datos de las librerías digitales y soporta interoperabilidad a través de OAI-PMH. El servidor de datos debe proveer las siguientes funcionalidades:

Servicios de bibliotecas personales digitales:

El servidor de datos ofrece operaciones de creación, eliminación, actualización y obtención, sobre los objetos almacenados en el espacio personal de almacenamiento de datos. Además, provee servicios para copiar y mover documentos y colecciones, así como búsquedas en la biblioteca personal, envío de documentos a una biblioteca digital de otro usuario, envío de documentos a través de correo electrónico a cualquier otro usuario, cambio de formato en documentos, edición y creación de metadatos y otorgar o quitar permisos sobre la biblioteca personal.

Almacenamiento personal de datos:

Provee de un almacenamiento estructurado con capacidades escalables de obtención de información para bibliotecas digitales.

Una herramienta de obtención de información es utilizada para indexar el contenido personal de la biblioteca digital. Una base de datos es utilizada para almacenar los objetos de las librerías personales.

El uso motor de búsquedas y una base de datos dentro del servidor de datos permite consultas basadas en texto y en SQL para soportar la clasificación jerárquica del contenido de una biblioteca digital.

Soporte para OAI-PMH:

El servidor de datos presenta los metadatos de una biblioteca personal digital a través de OAI-PMH [25].

El modelo de PDLib es, a grandes rasgos, el siguiente:

Una biblioteca contiene una o más colecciones. Una colección contiene documentos o más colecciones y, a su vez, debe estar asociada a un conjunto de metadatos. El conjunto de metadatos está compuesto por una o más definiciones de metadatos. Los campos que componen los metadatos de un documento están determinados por el conjunto de metadatos de una colección. El conjunto de metadatos de una colección es definido, por omisión, por el conjunto de metadatos de su colección padre y puede ser redefinida por el usuario.

Los permisos están diseñados con los siguientes niveles de acceso:

- *Persona (Personal):* El acceso es restringido al dueño de la librería.
- *Entrante (Incoming):* Otros usuarios cuentan con la capacidad para crear documentos y colecciones.

- *Saliente (Outgoing):* Otros usuarios cuentan con la capacidad para navegar por las colecciones y obtener documentos.
- *Actualización (Update):* Otros usuarios cuentan con la capacidad para actualizar documentos y sus respectivos metadatos asociados.
- *Compartido (Shared):* Otros usuarios cuentan con la capacidad total para acceder a documentos y colecciones.

2.3 Comunicación punto a punto

Es sabido que la comunicación punto a punto es un tipo de arquitectura para que dos o más computadoras establezcan intercambio de información sin la necesidad de hacer uso de un servidor o computadora central que proporcione algún o algunos servicios de utilidad para la red, de hecho, todos los nodos conectados son iguales o dicho de otra forma, tienen el o los mismos programas instalados. Sin embargo dudas como ¿Qué es punto a punto?, ¿Cómo esta definido?, ¿Cuáles son los problemas que tiene? Y ¿Qué aplicaciones existen para está tecnología? pueden surgir por lo que a continuación se presentan la respuesta a esta preguntas que ayudarán a entender un poco más sobre esta tecnología. La comunicación punto a punto es muy diferente a la tecnología basada en el servicio y entender estas diferencias es esencial.

El cómputo punto a punto no es un concepto nuevo, básicamente se puede decir que en el momento que dos computadoras se conecten por primera vez forman una red punto a punto. Servicios como servidores de correo y servidores de dominio operan en redes punto a punto. Por ejemplo un servidor de correo interactuar directamente con otro enviando, recibiendo y direccionando mensajes y a esto puede considerársele una comunicación punto a punto.

Sin embargo a lo anterior el término punto a punto (P2P) es relativamente nuevo. Algunas personas le dan el crédito de esta tecnología a Gene Kan y a algunos otros pioneros de Gnutella [9]. La definición de punto a punto está tomando un crecimiento muy grande, sin embargo, no es simple. No solo existen problemas con el desarrollo de aplicaciones P2P, sino que también existen muchos protocolos P2P e implementaciones que operan en caminos distintos.

2.3.1 Definición

La mejor descripción de punto a punto no es acerca de eliminar servidores, no es una simple tecnología, aplicación o modelo de negocios. Tal vez lo más controversial es que no debe ser caracterizada estrechamente en cuanto al grado de centralización contra el grado de descentralización de la red.

La centralización en un ambiente punto a punto puede consistir en un catalogo central, tal y como lo hace Napster. Naptser [10] actúa como una arquitectura cliente servidor tradicional cuando los usuarios buscan algún recurso y actúa como una red punto a punto cuando los usuarios se transfieren archivos.

Gnutella [9] por otro lado, es una red P2P completamente descentralizada. Todos los puntos son exactamente iguales excepto por su contenido. En general P2P es más un estilo de cómputo que hace las interacciones de la red más simétricas. Aún cuando puede haber servicios descentralizados, la parte más importante de una red P2P está enfocada en los puntos. La tecnología P2P ha generado un gran interés dentro la comunidad informática, tanto en usuarios finales como en profesionales de la computación. Un punto es básicamente una abstracción de una persona que intenta compartir información con otras. Bajo este contexto los conceptos de punto, nodo y usuario serán asumidos como sinónimos.

2.3.2 Herramientas para desarrollo de ambientes P2P

En el mercado existen dos herramientas para desarrollo P2P con java: JRRA [4] y JXTA [3, 18], ambas son gratuitas y de código abierto, es decir, se pueden descargar los fuentes sin ningún problema y sin necesidad de realizar algún pago a los autores, así mismo, es posible implementarlas para crear un ambiente P2P en su estado original o bien modificarlas para un fin en particular. En la siguiente sección se describirán brevemente cada una de ellas y posteriormente en el capítulo dos, la que será usada para hacer la prueba de concepto de esta tesis y por qué.

2.3.2.1.1 Rocky Road (JRRA)

En el mercado existen o existieron muchas aplicaciones punto a punto para intercambio de información. Tal es el caso de Gnuttella [9], Morpheus [8] y Napster [10], entre otros; sin embargo, ninguno de ellos fue diseñado para trabajar en dispositivos

móviles, y ahí existe un área de oportunidad grande para explotar. Rocky Road es, de entre todos los protocolos punto a punto existentes en la actualidad, uno de los pocos que funcionan tanto en pc's y servidores, como en PDA's y teléfonos celulares [4]. Existen dos versiones de JRRA: la versión estándar y la versión micro. Cada una de ellas es de código abierto. Una extensa gama de protocolos para redes IP y no IP, así como una flexible política de proceso basado en paquetes, convierten a JRRA en una herramienta muy poderosa y efectiva para crear ambientes punto a punto. Rocky Road es un conjunto de API'S que aceleran el proceso de creación de aplicaciones punto a punto. Es un mecanismo de transporte para sistemas distribuidos que fue diseñado para trabajar en ambientes heterogéneos. Además del protocolo central existe una serie de servicios que Rocky Road provee. Un servicio es un módulo que no es compatible con la visión de punto a punto puro y tampoco es obligatorio para todas las aplicaciones.

2.3.2.1.2 Servicios de JRRA

Búsqueda e indexación.

La tarea principal de este servicio es la de buscar puntos y/o recursos en la red, indexarlos y crear una jerarquía basada en información personal de cada entidad de la red en cuestión, a lo que se le llama índice.

Existen dos tipos de sistemas punto a punto, las orientadas a puntos y las orientadas a recursos.

Un ejemplo clásico del primer tipo de sistemas punto a punto son las aplicaciones de mensajes instantáneos, en donde los usuarios no están interesados en los recursos del punto que desean contactar, sino del punto en sí.

La transferencia de archivos es, por su parte, un ejemplo clásico del segundo tipo de sistemas punto a punto, en el cual a los usuarios no les interesa de quién es el recurso que buscan o dónde esta, sino obtenerlo.

Organizar las búsquedas en el primer tipo de sistemas es relativamente fácil; la solución típica es mantener un punto privilegiado que contenga información sobre todos los puntos de la red, tal como estatus y dirección actual, entre otros.

En el segundo tipo de sistemas punto a punto la búsqueda y navegación resultan más complicadas, ya que no sólo es necesario conocer dónde esta el recurso que se desea sino

que también se necesita conocer cuál es la mejor ruta para ese recurso, en el caso de que se encuentre en más de un lugar.

Servicio de seguridad.

Como sabemos, todos los sistemas en los cuales se involucra a la internet están expuestos a ataques de hackers (persona que entra a algún sistema de cómputo sin autorización), y los sistemas punto a punto no son la excepción. Pensemos que un punto desea enviar un mensaje o recurso a otro punto a través de Internet; al hacerlo, dicho recurso es vulnerable a que alguien en la red lo intercepte, lo deseche, lo envíe a otro lugar o, peor aún, lo vea y se entere de información clasificada.

El hecho de tener una red punto a punto pura hace muy difícil la unión de nuevos puntos al ciclo seguro de transmisión de datos; lo mismo que proveerle una llave pública y privada para poder conectarse. Así, una solución factible es tener un conjunto de puntos privilegiados, en donde cada punto es identificado por una dirección y autenticado por una llave.

La gran ventaja del punto privilegiado consiste en que su llave es conocida en toda la red y, a su vez, conoce todas las llaves de los puntos pertenecientes al círculo seguro. Un círculo seguro es un grupo de puntos que se comunican de manera indubitable y auténtica. Un miembro del círculo seguro debe ser identificado por su dirección y por su llave.

El algoritmo de intercambio seguro funciona de la siguiente forma:

- El punto A desea iniciar una conexión segura con el punto B. A no conoce la llave de de B y elige un punto privilegiado K, con una llave pública conocida.
- A hace un petición de la llave de B vía conexión segura con el servidor de llaves K a través de la llave pública de éste último.
- K le envía a A la llave de B, a través de una mensaje cifrado usando la llave pública de A.
- A le envía un mensaje a B usando la llave pública de B obtenida del servidor de llaves.
- B le pide al servidor de llaves la llave pública de A.
- B desencripta el mensaje de A.
- A y B establecen una comunicación segura e intercambian mensajes a través de sus llaves públicas.

Un punto que desea ingresar al círculo seguro debe realizar algunas tareas:

- El punto (B) genera un nuevo par de llaves.
- La llave pública de B es enviada al servidor de llaves vía conexión segura
- El punto privilegias opcionalmente confirma la el registro y opcionalmente envía un mensaje cifrado.

Si un punto desea cambiar sus llaves para evitar ataques de fuerza bruta, debe llevar a cabo las siguientes acciones:

- El punto (B) genera un nuevo par de llaves
- El punto le envía al servidor de llaves su nueva llave pública con la vieja y opcionalmente encriptada con la llave pública de K.
- El punto privilegiado publica la llave de B.

2.4 JXTA

JXTA fue creado como un esfuerzo de código abierto desde su concepción. Esta innovadora y controversial investigación fue basada en algunas de las creencias de sus autores en Sun Microsystems, estas creencias básicamente hablan de que el proyecto no debería ser desarrollado enteramente por una solo compañía, sino que desarrolladores de Sun y externos deberían colaborar, por lo que el desarrollo debería estar fuera de Sun, de tal forma que hubiera mucha gente involucrada y así poder tener un proyecto de código abierto accesible para cualquiera que deseara participar o utilizarlo.

JXTA puede definirse de la siguiente manera: Es un conjunto de seis protocolos que fueron específicamente diseñados para trabajar en una forma de cómputo punto a punto [18]. Haciendo uso de los protocolos de JXTA, los puntos pueden cooperar con los grupos de puntos en una forma auto-organizada y auto-configurable independientemente de su posición en la red y sin la necesidad de una infraestructura de administración centralizada. Todo esto quiere decir que JXTA es un framework con un conjunto de estándares que soportan aplicaciones punto a punto. JXTA no es una aplicación, y no define el tipo de aplicaciones que pueden ser desarrolladas. Los protocolos definidos en el estándar no están definidos de una forma rígida por lo que su funcionalidad puede ser extendida dependiendo de los requerimientos específicos [18].

JXTA fue hecho en tres capas distintas. La primera es la plataforma. La plataforma contiene la funcionalidad principal usada por los servicios, que constituyen la segunda capa. Los servicios proveen acceso a los protocolo de JXTA. Finalmente están las aplicaciones que utilizan a los servicios para acceder a la red JXTA y sus utilerías.

Los objetivos de JXTA son los siguientes:

1. Independencia del sistema operativo.
2. Independencia del lenguaje.
3. Proveer los servicios y la infraestructura necesaria para aplicaciones P2P.

Como puede observarse JXTA es una muy buena opción para desarrollar aplicaciones en ambientes punto a punto, esta tecnología cuenta con una versión para dispositivos móviles.

2.5 Java como lenguaje de programación para ambientes móviles.

Java es un lenguaje de programación creado por Sun Microsystems, que nació como un pequeño proyecto dentro de la compañía, muy separado de la empresa, con la idea de que se pudiesen crear cosas que no estuvieran basadas en las tecnologías que en aquel momento tenía la empresa.

Se pensó primero en una plataforma que permitiera controlar distintos dispositivos, una especie de mando a distancia universal muy sofisticado que pudiera configurarse con nueva funcionalidades. Pero aquello no llega a concretarse, seguramente porque se adelantó mucho a su tiempo.

Hubo un segundo intento enfocado a la televisión digital, pero fracasó también. En el tercer intento se encuentra en área muy grande de oportunidad en la internet y Java se presenta en mayo del 95 ya como un lenguaje de programación consolidado.

Dado al gran auge e impacto que Java ha tenido en los últimos años Sun le ha invertido mucho, tanto que no hace mucho tiempo empezó a incursionar en los dispositivos móviles, y dado que, Java es un lenguaje que se caracteriza principalmente por poder ejecutarse en cualquier ambiente, es bastante lógico pensar que los teléfonos celulares y PDA’s son dispositivos en los que se puede desarrollar y correr aplicaciones hechas en este lenguaje.

Precisamente para lo anterior Sun presentó la versión de Java para ambientes móviles a la cual le llamó Java 2 Mobile Enviroment (J2ME) [2].

La arquitectura J2ME define configuraciones, perfiles y paquetes opcionales como elementos que ayudan a que la completa construcción de “Java Runtime environment” pueda ser ejecutada en una amplia variedad de dispositivos. Cada combinación esta optimizada para ocupar el mínimo de memoria, poder de procesamiento y capacidades de I/O de dispositivos móviles [1].

Cada una de las configuraciones está compuesta por una máquina virtual y un conjunto mínimo de librerías de clases. Este conjunto provee una base funcional para un rango de dispositivos específicos que comparten las mismas características, como lo son la conectividad a la red y la capacidad de memoria. Actualmente existen dos configuraciones de J2ME: Connected Limited Device Configuration (CLDC), y Connected Device Configuration (CDC).

CLDC: Esta configuración es parte fundamental de la arquitectura J2ME. Esta configuración provee el más básico conjunto de librerías y características de la máquina virtual que deben ser presentadas en cada implementación del ambiente J2ME. Fue diseñada dispositivos muy pequeños, procesadores lentos, conectividad a la intermitente a la red y una cantidad de memoria limitada, tales como teléfonos celulares, PDA’S y pagers de dos vías.

CDC: Ésta configuración está diseñada para dispositivos que tienen mayor memoria, procesadores más rápido y un mucho mejor ancho de banda, como son TV set-up boxes, gateways residenciales, sistemas telemáticos instalados en automóviles y high-end PDA’S. CDC incluye una maquina virtual completa, y un conjunto mucho mayor de librerías de clases que CLDC. La mayoría de los dispositivos que soportan CDC tienen procesadores de 32 bits y un mínimo de 2 MB de memoria disponible para la plataforma y las aplicaciones asociadas.

2.5.1 Perfiles

Para que poder proveer un ambiente completo en los dispositivos móviles, las configuraciones mencionadas anteriormente, deben de combinarse con un conjunto de

API’s de alto nivel llamadas perfiles, que definan la interfaz de usuario, el acceso a las características propias de cada dispositivo y al ciclo de vida de cada aplicación .

Mobile information device profile (MIDP): Este perfil esta diseñado para dispositivos tales como teléfonos celulares y PDA’s. Ofrece una base para la funcionalidad que requieren las aplicaciones móviles, incluyendo la interfaz de usuario, conectividad a la red, almacenamiento local de datos, y manejo de la aplicación. Combinado con CLDC, MIDP provee un JRE que aumenta la capacidad de los dispositivos y minimiza tanto la el uso de memoria como el consumo de energía.

Foundation Profile: Dado que los perfiles de CDC están diseñados en capas, es posible añadir más perfiles, tantos como se necesite la aplicación para diferentes tipos de dispositivos. El Foundation Profile (FP) es el perfil de más bajo nivel para CDC. Provee una implementación de red, así que se puede utilizar en implementaciones que no tengan interfaz de usuario. También puede combinarse con otros perfiles como el Personal Basis Profile y el Personal Profile para dispositivos que requieren una interfaz de usuario gráfica (GUI).

Personal Profile: El Personal Profile (PP) es el perfil CDC para dispositivos que requieren una GUI completa o soporte para applets. Este perfil incluye todas las librerías del Java Abstract Window Toolkit (AWT) y ofrece soporte para correr applets diseñados para usarse en computadoras de escritorio. El Personal Basis Profile (PBP), es un subconjunto de PP, provee soporte para trabajar en red para dispositivos que soportan un nivel limitado de presentación gráfica o que requieren un conjunto de herramientas gráficas especializadas para aplicaciones específicas. Una vez definidas las características de los dispositivos móviles, las plataformas de desarrollo y sistemas operativos, es necesario tomar en cuenta los protocolos existentes para realizar la comunicación entre los diferentes componentes de un sistema.

2.6 Sockets

Un socket puede ser visto como un tubo virtual entre dos puntos por donde se puede enviar información. Existen dos clasificaciones principales de sockets de acuerdo al tipo de conexión que con ellos se puede realizar, los TCP y los UDP, estos últimos también son llamados sockets de datagramas.

Cada socket es identificado en la red de manera única con dos números. El primero es número de 32 bits llamado dirección de internet (IP), el segundo número es de 16 bits llamado puerto. La dirección de internet o dirección IP, es única en toda la internet, es decir, cada computadora conectada a la red, debe tener una y esta debe ser única, de esta forma un socket puede conectarse con otro a través de sus direcciones de internet y sus puertos.

2.6.1 Sockets UDP

El protocolo de comunicación UDP provee un modo de comunicación en donde el envío de información de un nodo a otro se hace a través de paquetes de datos llamados datagramas. Un datagrama es un mensaje independiente enviado a través de la red donde el destinatario, tiempo de llegada y contenido no están garantizados.

2.6.2 Sockets TCP

Los sockets TCP, también llamados orientados a conexión. Un socket TCP es una analogía de una conexión telefónica. Existen dos tipos de sockets TCP, los ordinarios (también llamados clientes) y los servidores. Un socket de tipo servidor nunca es utilizado para transmisión de información, ya que su único propósito es recibir y atender peticiones de conexión que hacen los sockets ordinarios. Cuando un socket cliente desea establecer comunicación con un socket servidor debe primero pedirle conexión y así cuando éste recibe la petición construye un socket ordinario para efectuar la conexión, y de esta forma puede volver a esperar por más peticiones.

2.7 XML

El lenguaje extensible de marcación – Extensible Markup Language (XML) – es un formato de texto simple y muy flexible derivado de SGML (ISO 8879) [11]. Originalmente diseñado para enfrentar los retos de la publicación electrónica a gran escala. XML también juega un importante papel en el intercambio de una gran cantidad de datos en internet.

XML describe una clase de objetos de datos llamados documentos XML, así como también describe de forma parcial el comportamiento de los programas de computadora que los procesan. Un documento XML puede consistir en una o más unidades de almacenamiento llamadas entidades las cuales contienen datos parseados o sin parsear.

Según Wayne Hanslo en su artículo “The efficiency of XML as an Intermediate Data Representation for wireless Middleware Communication” [15], la manera más fácil de de acoplar aplicaciones móviles con sistemas para acceso de información es utilizando una sola tecnología en un sistema “end to end” que acceda a los servicios directamente. Sin embargo, este método enlaza de manera muy firme a la aplicación móvil con los servicios que utiliza, los mensajes son en formato binario, el cual debe ser conocido previamente, los dispositivos móviles que no soporten la tecnología usada por el sistema “end to end” no podrán acceder a los servicios disponibles.

Una solución para estos problemas es el uso de XML para integrar aplicaciones móviles a nivel de datos. XML es formato auto descriptivo e independiente. Los mensajes XML pueden ser intercambiados entre diversos tipos de sistemas independientemente del hardware, sistema operativo o lenguaje de programación utilizados.

2.8 MYSQL

MYSQL [24] es un sistema manejador de base de datos relacional de código abierto. Como muchos manejadores modernos de base de datos MYSQL usa SQL (Structured Query Language), un lenguaje de programación para crear, actualizar y mostrar datos contenidos dentro de una base de datos relacional. SQL es la forma de comunicarse con el sistema manejador de base de datos.

La base de datos MYSQL se ha convertido en el sistema manejador de base de datos de código abierto más popular en el mundo por su consistente y buen desempeño, alto rendimiento y fácil uso. Es usada en más de 6 millones de instalaciones en todo el mundo, que van desde grandes corporaciones hasta implementaciones en aplicaciones especializas.

MYSQL cuenta con soporte de índices y búsquedas de texto completo. A los índices utilizados para realizar búsquedas de texto completo se le llama índices invertidos. Un índice invertido es un índice que se crea para cada palabra de un texto dado, dicho índice es accesado por algún método de búsqueda. En una base de datos los índices invertidos son una secuencia de pares formadas por una palabra clave y el registro en la base de datos donde ésta se encuentra (locación). El índice debe ser ordenado por las palabras clave para permitir una búsqueda rápida. Para indexar un texto con un índice de texto completo es necesario indexar cada palabra del texto, por lo cual puede llegar a causar un problema el

espacio en disco. El índice es invertido en el sentido de que la palabra llave es usada para encontrar el registro a diferencia de un índice común.

Cabe mencionar que MYSQL será utilizado en el prototipo para indexar a cada punto con su esqueleto y así poder ofrecer un servicio de búsquedas de recursos, el cual será explicado más adelante en el capítulo tres y cuatro.

2.9 Otros esfuerzos enfocados a la comunicación punto a punto

En esta sección se hablará un poco sobre otros esfuerzos enfocados a la comunicación punto a punto, sus características y el problema que intentan resolver.

2.9.1 XMIDDLE

XMIDDLE [19] es un middleware basado en XML enfocado hacia sistemas móviles y redes ad-hoc, que trata de resolver algunos los problemas de la comunicación en dispositivos móviles, tales como reconciliación de datos y sincronización de información. Debido a que en un ambiente punto a punto los datos están de manera compartida y estos datos deben ser consistentes en toda la red, es decir si un nodo cambia su información compartida es necesario que todos los hagan, XMIDDLE implementa una herramienta de sincronización de datos para mantener uniforme la información de la red. También implementa mecanismos de reconciliación de datos cuando existen desconexiones en medio de una transmisión.

2.9.2 P-Tree

P-Tree [22] es una estructura de índices para el descubrimiento de recursos en redes punto a punto. Básicamente la idea central de este proyecto es que cada nodo mantenga un XML que lo describa a él y a sus recursos. Las bases de datos centralizadas usan un índice de árbol B+[23]. La idea central de P-Tree es mantener partes semi-independientes árboles B+ en cada nodo. P-Tree básicamente resuelve problemas en cuanto a la baja escalabilidad al realizar búsquedas a través de todos los puntos aún y cuando no todos cuenten con la información requerida. La forma de descentralizar la información aunada a la idea de mantener índices para realizar de forma más rápida y eficiente las búsquedas de recursos.

2.9.3 Kosha

Kosha [21] es un tipo de sistema de archivos de red (NFS) punto a punto. Kosha aprovecha el espacio de almacenamiento redundante resultado del uso de un cluster de nodos para de esta forma proveer un sistema confiable de archivos compartido que actúa como un gran almacén con las semánticas normales de un NFS. Los sistemas punto a punto de almacenamiento proveen transparencia de localización, transparencia de movilidad, balanceo de cargas y replicación de archivos.

2.9.4 PeerDB

PeerDB [16] es un sistema punto a punto de datos distribuido. PeerDB se distingue de entre los demás sistemas punto a punto existentes en varias formas: Primero, es un sistema de administración de datos completamente autónomo que soporta búsquedas basadas en contenido de fina granularidad. Segundo, facilita la distribución de datos in un esquema distribuido. Tercero, combina el poder de los agentes móviles dentro de sistemas punto a punto para realizar operaciones del lado de los puntos (operaciones de consulta). Cuarto, La red PeerDB es auto-configurable.

2.9.5 BestPeer

BestPeer [20] es un sistema punto a punto genérico diseñado para servir como plataforma en donde aplicaciones P2P pueden ser desarrolladas de manera fácil y eficiente. La red consiste en dos tipo de entidades: Un gran número de computadoras (nodos), y un pequeño número de servidores de tipo “location independent global names lookup” (LIGLO). Cada nodo participante que corra el software BestPeer podrá ser capaz de comunicarse y compartir recursos con otros nodos. BestPeer cuenta con una serie de características que lo distinguen de los demás sistemas punto a punto: Primero, integra dos tecnologías poderosas: agentes móviles y sistemas punto a punto. Mientras que los sistemas punto a punto proveen de ciertas funcionalidades de distribución de recursos los agentes móviles crecen dichas funcionalidades. Segundo, BestPeer no solo facilita un esquema de búsqueda de datos donde es posible que un archivo sea parcialmente compartido, sino que también se comparte poder de cómputo. Tercero, un nodo en una red BestPeer puede ser auto-configurado de forma dinámica. Finalmente BestPeer introduce un servicio llamado Location Independent Global Names Lookup Server (LIGLO) para proveer a cada nodo de

un identificador único y global. En este sentido los nodos que tengan diferentes direcciones IP, cuenten con un ID único en toda la red. Básicamente el servidor LIGLO tiene dos funciones básicas: Generar identificadores únicos globales (BestPeer Global Identify) y mantener el estatus actual de los nodos, por ejemplo su dirección IP actual y si el nodo se encuentra en línea o fuera de línea.

2.10 Conclusiones

Tal parece que las tecnologías enfocadas a dispositivos móviles esta creciendo a pasos muy grandes, aunque es una tecnología nueva, las aplicaciones en teléfonos celulares y en PDA'S ha tenido mucho éxito a lo largo del tiempo, cada vez más y más personas tienen acceso a este tipo de tecnología y por lo tanto cada vez hay más aplicaciones para estos, por ejemplo hoy en día un teléfono celular ya no tiene cabida sin al menos un calendario, reloj, y juegos de calidad considerable. Esto ha causado que cada vez haya más gente de tecnologías de información que deseen incursionar en el desarrollo de aplicaciones para ambientes móviles, y este es el caso del ITESM que ha incursionado en la parte de bibliotecas digitales en dispositivos móviles.

Aunado a lo anterior la parte de tecnologías punto a punto en dispositivos móviles está muy poco investigada, es decir, no hay muchas aplicaciones P2P en celulares o PDA'S, por tal motivo para esta tesis se encontró que dentro de las tecnologías de dispositivos móviles haya un área de oportunidad en esta ámbito, y sabiendo que el ITESM esta en vías de desarrollar una biblioteca digital para ambientes móviles es una gran área de investigación el hecho de poder tener un ambiente punto en dispositivos móviles ofreciendo servicios de bibliotecas digitales.

En este capítulo se presentaron algunas tecnologías para el desarrollo de aplicaciones punto a punto, MYSQL como manejador de base de datos, XML, Java como lenguaje de programación y los sockets para comunicación en la red. Para la arquitectura propuesta que se presentará en el siguiente capítulo se decidió por JRRA como API para el desarrollo del sistema P2PDlib por su facilidad de uso y su orientación para dispositivos móviles, este API esta hecho en Java por lo que se continuará con su uso, además de contar con la ventaja de ser multiplataforma, MYSQL será utilizado como administrador de base de datos por ser de código abierto, robusto y eficiente. XML será utilizado para el

almacenamiento del esqueleto de la biblioteca digital debido a su estructura de árbol (igual a la estructura de una biblioteca) y su facilidad de uso, además de que el hecho de ser un estándar hace más fácil su integración con otras tecnologías.

3 Arquitectura Propuesta

En este capítulo se tratará la arquitectura de P2PDLib, es decir el mecanismo por cual se proveen funciones de bibliotecas digitales cuando el dispositivo móvil no esta conectado al servidor de datos, un panorama general de la forma en que funciona y las partes que lo formarán.

3.1 Introducción

Para que el modelo de comunicación punto a punto aquí presentando funcione es necesario que interactúe con otro proyecto de sincronización offline.

Como ya fue explicado anteriormente, ésta tesis esta basada en el proyecto PDLib [17], debido a que se presenta una propuesta de comunicación punto a punto ofreciendo servicios de bibliotecas digitales. El proyecto PDLib esta basado básicamente en tres partes, el cliente, el servidor de datos y el MCM [17], sin embargo la médula espinal es el servidor de datos debido a que en éste se encuentran almacenadas todas las colecciones y documentos de los usuarios de PDLib.

En este prototipo algo de lo que se pretende es compartir “vistas”- Estructura lógica de una biblioteca digital que puede comprender todas las colecciones o solo algunas de ellas – y algunos de los documentos en ellas contenidas entre los usuarios de PDLib sin hacer uso del servidor de datos, por lo que es necesario que en el dispositivo móvil se encuentre por lo menos la estructura lógica de la librería digital y algunos de los documentos que en ella se encuentran.

Para lo anterior es necesario que otro mecanismo pueda conectarse al MCM y hacer una copia al dispositivo móvil del esqueleto de la biblioteca digital que se encuentra en el servidor de datos. A la copia local del esqueleto de la biblioteca digital se le llamará en lo sucesivo “biblioteca digital local” o “digital library skeleton” y para referirse a la biblioteca digital que se encuentra en el servidor de PDLib se le llamará “PDLib” o “biblioteca digital remota”. Cabe mencionar que en esta copia local solo estará la pura estructura y algunos de los documentos de algunas colecciones debido al poco espacio de memoria de un dispositivo móvil. En la figura 3.1 se puede apreciar la forma en la que el cliente se

comunica a través del MCM con el servidor de datos para obtener el esqueleto de la biblioteca digital y viceversa, es decir la sincronización.

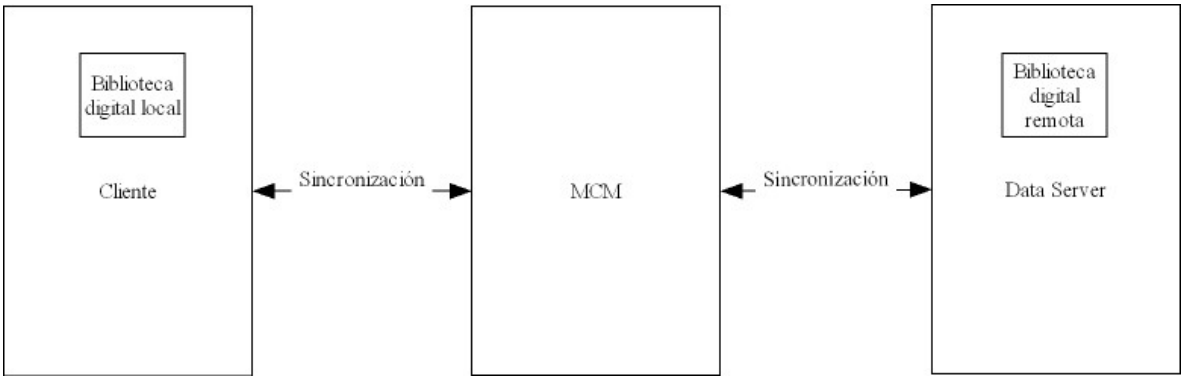


Figura 3.1Biblioteca digital local y remota.

La importancia de lo anterior para esta tesis es que se da por hecho que en el dispositivo móvil ya se encuentra la estructura lógica de la biblioteca digital para que, el prototipo descrito en esta tesis pueda interactuar con dicha copia. En otras palabras el proceso de comunicación para intercambio de documentos y vistas de una biblioteca digital a través de una comunicación punto a punto debe trabajar en conjunto con un mecanismo de sincronización entre el servidor de datos y el dispositivo móvil. En la figura 3.2 se puede apreciar mejor la comunicación punto a punto entre dispositivos móviles para ofrecer servicios de bibliotecas digitales haciendo uso de un punto privilegiado.

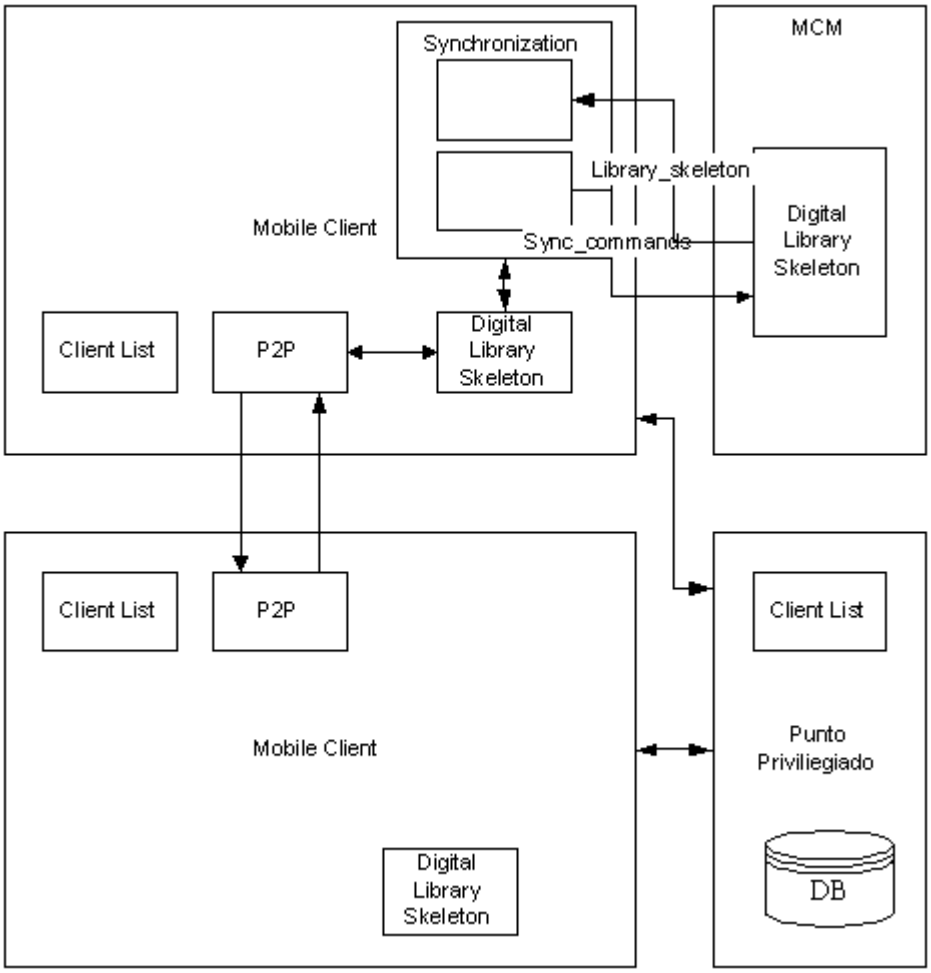


Figura 3.2 Arquitectura general del sistema punto a punto y sincronización con el servidor de PDLib.

3.1.1 Componentes de la arquitectura General

Clientes: Se les llama clientes a todos aquellos dispositivos móviles como PDA's, celulares, Laptops, Tablet PC's con conexión inalámbrica y que tienen comunicación con el punto privilegiado y/o con el servidor de datos a través de un middleware (Mobile Connection Middleware, MCM).

MCM: El Mobile Connection Middleware, es una capa de software que se encarga de adaptar las conexiones de los clientes con el servidor de datos.

Servidor de Datos: Es donde se encuentran almacenadas las bibliotecas digitales de todos los usuarios de PDLib. Se encarga de recibir peticiones de los diversos clientes (móviles y fijos), ejecutar operaciones, y de manera general de administrar las librerías de los usuarios.

Punto Privilegiado: El punto privilegiado es una capa de software que provee de ciertos servicios de conexión a los clientes, la necesidad de utilizar esta capa es debido a los problemas de conexión de los dispositivos móviles, como por ejemplo el cambio de lugar geográfico. Sin embargo este componente al estar en una computadora con más recursos de cómputo que un dispositivo móvil puede servir para resolver otros problemas propios de un sistema punto a punto en ambientes móviles.

Como se puede apreciar en la figura 3.2 existe una conexión de los clientes con otra capa de software (punto privilegiado). Esta capa, se encarga de proveer a todos los puntos con las direcciones de internet de todos sus similares, para que de esta forma todos estén conectados aún y cuando un nodo se desconecte y conecte en alguna otra área geográfica, ofrece también el servicio de generación de identificadores únicos para todos los nodos conectados a la red punto a punto, así mismo el punto privilegiado se encarga para recibir y procesar peticiones de búsquedas sobre recursos de la red, por lo tanto todos los clientes deben tener comunicación con esta capa.

Esta tesis solo se basará en la investigación e implementación de un ambiente punto a punto en dispositivos móviles ofreciendo servicios de bibliotecas digitales por lo que la parte de comunicación y sincronización con el MCM no entrarán en éste trabajo.

3.2 Arquitectura propuesta

Se propone crear un ambiente con tecnologías punto a punto para ofrecer servicios de bibliotecas digitales, específicamente del proyecto PDLib.

En un ambiente punto a punto los puntos son lo más importante; se definen como representaciones abstractas de una persona que desea compartir información con otras personas; sin embargo, para lograrlo es necesario contar con un punto especial que esté en posibilidad de conocer las direcciones físicas de todos los demás puntos, a fin de poder informar sobre algún cambio para que la comunicación siempre se establezca con éxito.

Una de las principales características de la comunicación punto a punto es que para compartir información entre los nodos, no es necesario utilizar algún servidor o intermediario; salvo un punto especial que sólo servirá para avisar en caso de existir algún cambio en la red que sea de interés para los nodos como, por ejemplo, la llegada de algún

nuevo punto o su desconexión. Sin embargo el punto privilegiado (PP) puede también ofrecer servicios de búsqueda e indexamiento de recursos.

La arquitectura que se propone es la siguiente: cada dispositivo móvil contará con un software cliente instalado dentro del mismo. Este cliente podrá comunicarse con los otros puntos (dispositivos móviles con el software cliente). Adicionalmente se contará con un punto privilegiado que como ya se mencionó será el encargado de mantener a los puntos en comunicación.

Como ya se mencionó, el prototipo se dividirá en dos grandes aplicaciones, la aplicación cliente (la que estará en el dispositivo móvil) y una aplicación de punto privilegiado (PP). En la figura 3.3 se puede observar a cada dispositivo móvil comunicándose eventualmente con el punto privilegiado y comunicándose con otros dispositivos. En dicha figura se muestra que la comunicación entre un dispositivo móvil y el PP es eventual, es decir solo lleva a cabo cuando es necesario hacer uso de algún servicio que solo provee este último.

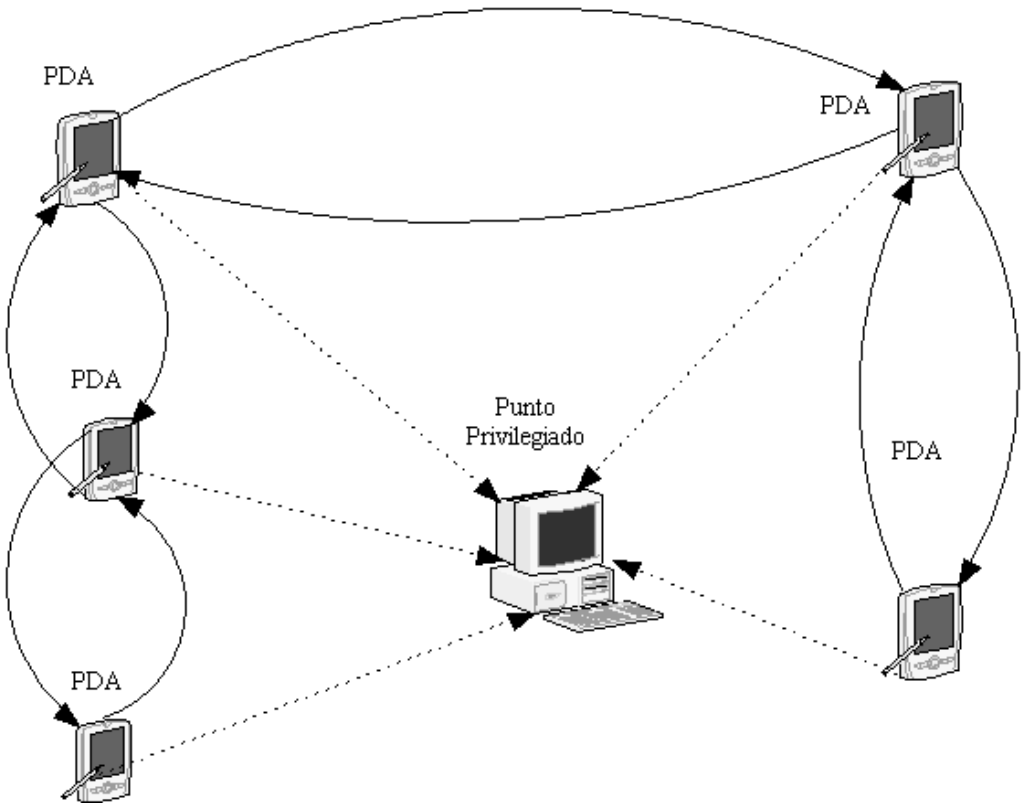


Figura 3.3 Arquitectura general del ambiente punto a punto

3.2.1 Servicios identificados a ofrecer

Parte importante de esta tesis es identificar los servicios de bibliotecas digitales que puede ser ofrecidos a través de la tecnología punto a punto, a continuación se presentan de manera muy general los que la arquitectura debe ofrecer:

- *Almacenamiento local de una copia de la biblioteca digital contenida en PDLib (Digital Library Skeleton):* Cada usuario tendrá una copia de su biblioteca digital dentro de su PDA, es decir, dentro de la aplicación que se encontrará en cada uno de los dispositivos móviles se contará con la estructura lógica de la biblioteca digital (biblioteca digital local) que se encuentra en los servidores de PDLib, en esta estructura se podrán observar las colecciones y documentos de manera lógica. Se refiere a manera lógica el hecho de que los documentos no necesariamente estarán en el dispositivo móvil, sin embargo se tendrá una referencia al mismo, de tal manera que el usuario pueda saber la existencia de un documento dentro de alguna colección.
- *Compartir colecciones de la biblioteca digital (vistas):* A partir de la estructura lógica de la biblioteca digital contenida en el dispositivo móvil, el usuario puede crear vistas de ésta (básicamente una vista es una colección) y compartirlas para que otros usuarios vean lo que hay en la vista compartida (otras colecciones y documentos). Las vistas son enviadas y recibidas para consulta de lo que hay en otras bibliotecas y solo se comparte la estructura lógica de la vista, es decir, en este servicio no hay intercambio de documentos. Una vista como ya se mencionó no es más que un subconjunto de la biblioteca digital local.
- *Compartir documentos entre puntos:* Dentro de los dispositivos móviles se pueden encontrar algunos documentos físicos, que fueron traídos del servidor de datos de PDLib. Dentro de la copia de la estructura de la biblioteca digital local, se tienen también, como ya se mencionó, los documentos de manera lógica, sin embargo existe un metadato que le indica al usuario si el documento existe en la PDA o solo se encuentra en el servidor. Si el documento está en el dispositivo éste puede ser enviado a algún otro usuario del sistema punto a punto. Cuando un usuario recibe un documento, éste puede seleccionar la colección dentro de su copia local de la

biblioteca donde se almacenará. Cabe mencionar que si el documento no se encuentra físicamente en el dispositivo móvil únicamente se enviaran sus metadatos

- *Generación de un identificador único por nodo o punto:* Debido a que las direcciones IP o direcciones de red de los puntos conectados son muy variables, es necesario que cada uno de ellos sea identificado de manera única.
- *Servicio de reconciliación por pérdida de conexión:* Dentro de la comunicación en dispositivos móviles existe el problema de las frecuentes desconexiones, por tal motivo se debe contar con un mecanismo de reconciliación de información para cuando esto ocurra.
- *Búsquedas de recursos en la red punto a punto:* Todos los puntos conectados a la red punto a punto podrán realizar búsquedas sobre recursos de los demás puntos a través del uso del punto privilegiado y del uso de un DBMS. Esta funcionalidad hereda algunas ideas del proyecto PeerDB [16], PeerDB es un sistema de compartición de datos distribuido, en donde al igual que éste proyecto hacen uso de una base de datos, sin embargo dicha base de datos se encuentra del lado de cada nodo, y las búsquedas se hacen en cada uno de ellos a través del uso de agentes móviles, es decir que hacen uso de un esquema totalmente descentralizado en donde no se hace uso de algún servidor. En el proyecto de P2PDLib no es posible tener un servidor de base de datos del lado de cada nodo debido a que son dispositivos móviles y tienen recursos muy limitados.
- *Chat:* Todos los usuarios tienen la funcionalidad de poder enviar y recibir mensajes de texto entre ellos, de tal forma que puedan mantener una conversación tipo chat.

En el proyecto PDLib se ofrecen algunos servicios de bibliotecas digitales que requieren hacer uso de algunos sus objetos, ya sea para su modificación o para su consulta, por ejemplo obtener un documento o modificar una colección. Por otro lado, los servicios que se ofrecen en este trabajo también requieren hacer uso de algunos de los objetos de la biblioteca digital. En PDLib ya existen algunos objetos clave para una biblioteca digital, sin embargo para este proyecto se agregaron algunos y se agregaron por supuesto algunos servicios nuevos propios de P2P que hacen uso de ellos.

En la figura 3.4 se puede observar una tabla con los nuevos objetos creados en este trabajo de tesis, así como los nuevos servicios que se ofrecerán sobre ellos. Los objetos y servicios subrayados en negritas son nuevos, es decir no existían en PDLib, así mismo el servicio en itálicas tampoco existía en PDLib, pero no es implementado en este proyecto, sin embargo si es utilizado, por lo que se hace distinción.

Objetos	**Servicios**
Digital Libray Skeleton	*Retriveral, Update and Deletion*
Library	Creation, Retrieval, Update and Deletion.
Collection View	**Creation, Update and Deletion, P2PSharing.**
Collection	Creation, Retrieval, Update, Deletion, Copy and Move.
Document	Submision, Retrieval, Update, Deletion, Send, **Search**, Email, and Copy, **P2PSharing**
Document Format	Submision, Retrieval, Update, Deletion and Convertion
Metadata Set	Creation, Retrieval, Update, Deletion, and Get/Set Collection Metadata Set
Document Metadata	Update Document Metadata, **Backup Document Metadata in device, P2PSharing.**
Permission	Get/Set Collection Permission and Get/Set Document Permission

Figura 3.4 Servicios de PDLib y P2PDLib

3.2.2 Metadatos

La biblioteca digital local que se tendrá en el dispositivo móvil estará en XML, debido a su fácil manejo. La biblioteca digital de PDLib o biblioteca digital remota está formada por colecciones, documentos y metadatos que son información sobre los documentos, es decir datos sobre el autor, tipo, etcétera.

Un ejemplo de un XML donde estará representada la biblioteca digital local puede consultarse en la figura siguiente:

```
<?xml version="1.0" encoding="windows-1252" ?>
- <Home>
  - <Collection>
      <Id>1</Id>
      <name>Java</name>
      <Path>Home</Path>
      <Permission>Public</Permission>
      <LastUpdate>04-09-2005</LastUpdate>
    - <Document>
        <Id>2</Id>
        <name>Mi Tesis</name>
        <Permission>Public</Permission>
        <LastUpdate>04-09-2005</LastUpdate>
      - <Metadata>
          <Type>Tesis</Type>
          <Title>Operaciones Offline</Title>
          <Topic>Bibliotecas digitales,Offline Operations</Topic>
          <Author>Luis Basto</Author>
          <Abstract>Esta tesis esta ...</Abstract>
          <Language>Español</Language>
          <Publisher>Luis Basto</Publisher>
          <Date>Today</Date>
        </Metadata>
      </Document>
    </Collection>
  - <Collection>
      <Id>3</Id>
      <name>Java</name>
      <Path>/Home/</Path>
      <Permission>Public</Permission>
      <LastUpdate>04-09-2005</LastUpdate>
    - <Document>
        <Id>4</Id>
        <name>Mi Segunda Tesis</name>
        <Permission>Privada</Permission>
        <LastUpdate>04-09-2005</LastUpdate>
      - <Metadata>
          <Type>Tesis 2</Type>
          <Title>The second Operaciones Offline</Title>
          <Topic>Bibliotecas digitales,Offline Operations</Topic>
          <Author>Valeria Soto</Author>
          <Abstract>Este es el segundo ejemplo</Abstract>
          <Language>English</Language>
          <Publisher>Valerita</Publisher>
          <Date>Yesterday</Date>
        </Metadata>
      </Document>
    </Collection>
  </Home>
```

Figura 3.5 XML para la copia local de la biblioteca digital

Como se puede observar en la figura 3.5 hay una parte dentro del XML para los metadatos remarcado en un cuadro, estos metadatos son los que se identificaron y se adoptaron para uso del ambiente de comunicación punto a punto.

3.2.3 Aplicación cliente

Esta aplicación cliente, a la que en lo sucesivo se le llamará "cliente", será desarrollada bajo la versión móvil de Java, debido a que deberá poder ejecutarse en dispositivos móviles, tales como teléfonos celulares y PDA's. En la figura 3.6 se muestran los objetos utilizados y los servicios ofrecidos en la aplicación cliente.

Dicho cliente deberá proveer los siguientes servicios:

- *Conectarse con el punto privilegiado para unirse a la red punto a punto:* Para esto, es necesario que todos los puntos en la red conozcan la dirección del punto privilegiado al que se conectarán.
- *Generar un par de llaves para poder autentificarse en la red punto a punto:* Cada vez que un nuevo punto desee conectarse a la red, deberá generar un nuevo par de llaves (una pública y una privada) para poder conectarse con el punto privilegiado de forma segura.
- *Crear búsquedas sobre recursos en la red punto a punto:* Dado que las vistas de cada punto estarán en los dispositivos móviles, los demás puntos no pueden de forma certera saber que documentos o colecciones tiene cada uno de los puntos, por tal motivo al momento de que un nodo se conecte a la red punto a punto le deberá enviar sus vistas al punto privilegiado para que éste a su vez la guarde en una base de datos y le genere índices para sus metadatos de tal suerte que cada punto a través de la aplicación cliente podrá realizar búsquedas y enviárselas al punto privilegiado, para que éste las procese y le devuelva los nodos donde el recurso buscado se encuentra. La idea principal de esta funcionalidad es que un punto x pueda crear consultas SQL y enviárselas en formato de texto plano al punto privilegiado para que éste la procese en su servidor de base de datos y le devuelva los puntos que tengan los recursos buscados.
- *Recibir y procesar la lista de puntos conectados cuando el punto privilegiado le haya dado el acceso a la red:* Cuando un punto es aceptado para estar en la red

punto a punto, este recibirá un mensaje con la lista de puntos conectados y procesarla, de tal forma que cada punto sepa en todo momento las direcciones y propiedades de todos los demás puntos similares conectados a la red.

- *Navegar a través de la biblioteca digital local (digital library skeleton):* Cada usuario podrá navegar entre las colecciones y documentos contenidos en el esqueleto de su biblioteca digital para poder consultar sus metadatos.
- *Enviar y recibir documentos físicos:* Si un usuario desea algún documento específico, puede pedírselo a algún otro y recibirlo a través de la red punto a punto, sin necesidad de hacer uso de algún servidor. Una vez que el usuario reciba un documento, éste podrá agregar los metadatos del mismo a la colección que desee, para que en un fututo pueda consultarlos en la interfaz de consulta y navegación del esqueleto de la biblioteca digital o biblioteca digital local.
- *Enviar y recibir vistas de una biblioteca digital:* Cada punto podrá almacenar una vista maestra (la biblioteca local completa) de su biblioteca digital, varios subconjuntos (vistas parciales o colecciones) de dicha vista maestra y, vistas de otros nodos, cada nodo a demás podrá compartir las vistas parciales con los demás nodos conectados a la red punto a punto. Las vistas parciales serán creadas por su propietario, ya que él es el único capaz de decidir lo que pueden o no ver los demás usuarios. Básicamente dentro de la aplicación cliente habrá tres almacenes de recurso: el maestro que servirá para guardar el esqueleto de la biblioteca digital local, el esclavo que servirá para almacenar las vistas parciales creadas por cada usuario, el almacén foráneo que contendrá a todas las vistas foráneas que sean recibidas por el usuario y el almacén de documentos que servirá como repositorio de los documentos físicos que cada usuario reciba.
- *Permitirle al usuario seleccionar las colecciones o documentos que desea compartir:* Dado que habrá colecciones o documentos que el usuario no desee que los demás vean, es necesario que el cliente le permita seleccionar que parte o partes de la biblioteca digital esta dispuesto a compartir, para así poder crear las llamadas vistas parciales.
- *Reconciliación:* Como ya se mencionó debido a las constantes desconexiones de los puntos es necesario contar con un mecanismo de reconciliación. La forma en la que

funciona este mecanismo es creando un liga lógica entre los dos puntos que desean transferirse algo, para que si llega a existir una desconexión no prevista, los puntos al encontrarse conectados puedan terminar con la transferencia. En la liga lógica se deben encontrar el identificador único del nodo que envía y del que recibe, para así cuando conecten ambos puntos puedan encontrarse aún y cuando sus direcciones IP cambien, esto debido a que el PP al enviarle la lista de nodos conectados le manda encapsulado las direcciones IP y sus ID's. Cuando un nodo se conecta a la red punto a punto y luego de recibir a los puntos conectados deberá revisar dentro sus ligas lógicas aquellas que no se hayan completado para poder reestablecer conexión.

- *Enviar y recibir mensajes de texto para mantener una comunicación tipo Chat con cualquier punto conectado en la red:* Este servicios les permite a los puntos conectados a la red punto a punto mantener contacto con los demás nodos a través del intercambio de mensajes de texto a través de una arquitectura punto a punto.

El cliente deberá poder ofrecer los servicios de envío y recuperación de recursos (archivos, vistas, referencias) en los dos sistemas en las que la tecnología punto a punto lo permite (orientado al punto y al recurso). Para el primer sistema, es necesario que el punto que desea el recurso conozca previamente el nodo que lo tiene, paras así poder pedírselo y que éste a su vez se lo envíe a través de la red punto a punto. En este tipo de sistemas casi no se hace uso del punto privilegiado, ya que solo sirve para avisarle a los nodos de algún cambio en la red.

Para el segundo sistema, es necesario que el cliente pueda crear búsquedas y enviárselas al punto privilegiado para que este a su vez las procese y haciendo uso de un índice previamente creado, le devuelva el punto o puntos que tienen el recurso pedido y además cual de ellos está más cerca.

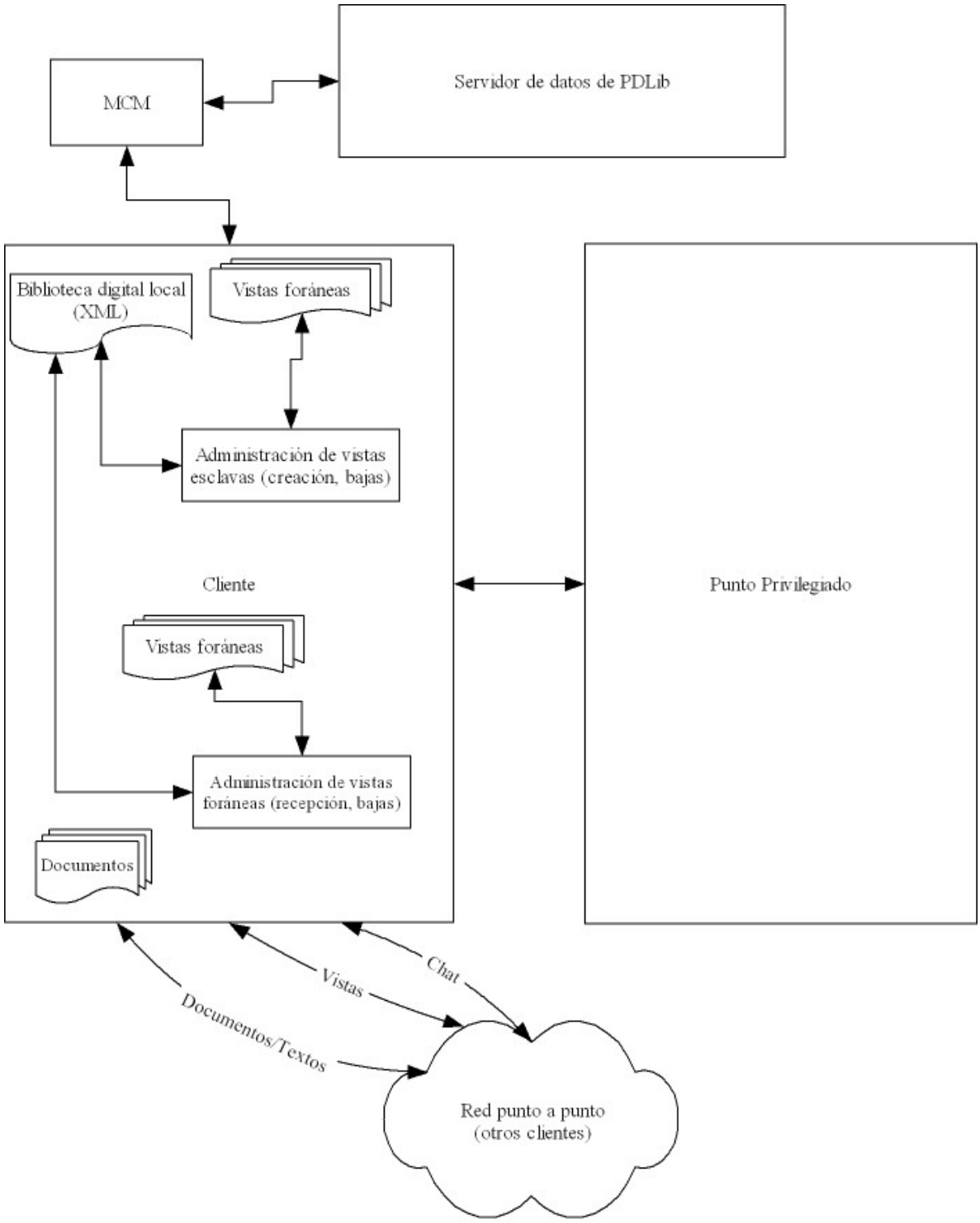


Figura 3.6 Funcionamiento general de la aplicación cliente

3.2.3.1 Escenarios

Para ilustrar el funcionamiento del proyecto se usarán algunos escenarios, los cuales describirán lo que en una situación real sucedería.

Escenario 1 - Envío y recepción de un documento físico contenido tanto en la biblioteca digital local, como en la pda del cliente que desea compartir.

1. El punto “A” y el punto “B” se conectan a la red punto a punto a través del punto privilegiado: Cada uno de los puntos debe enviar un mensaje especial al punto privilegiado pidiendo acceso a la red punto a punto, para que éste a su vez le envíe una lista con todos los puntos conectados a la red y sus direcciones entre otras cosas.
2. El punto “A” a través de un Chat le hace saber al punto “B” que desea que éste le envíe un documento contenido en su biblioteca digital local: La aplicación cliente cuenta con un Chat con el que todos y cada uno de los puntos pueden comunicarse entre ellos a través de texto plano.
3. El punto “B” selecciona de su biblioteca digital local el documento que va a enviar: La aplicación cliente cuenta con una interfaz donde el usuario puede ver su librería digital local completa, es decir, sus colecciones y los documentos contenidos en dichas colecciones, así como saber si un documento se encuentra o no su pda localmente.
4. Una vez que el usuario elija el documento a enviar deberá seleccionar de una lista de puntos conectados al punto “A”, para así poder enviarle dicho documento.
5. Cuando el punto “A” reciba el documento, la aplicación le preguntará la ubicación en su biblioteca digital local.

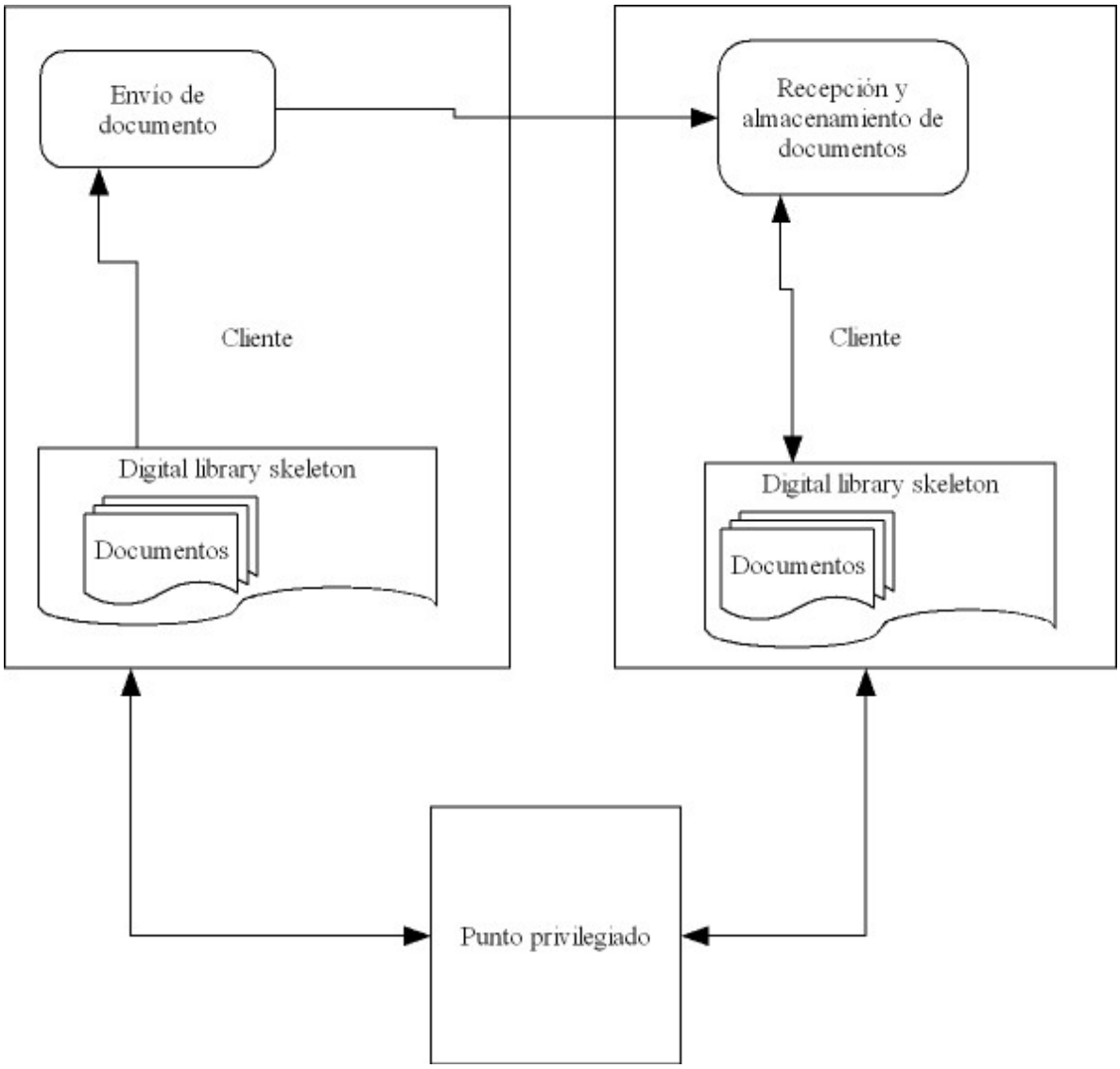


Figura 3.7 Escenario 1 Envío y recepción de documentos

Escenario 2 - Creación de vistas parciales para compartir

Para que la funcionalidad de envío y recepción de vistas pueda ser llevada a acabo, es necesario que exista un proceso de creación de dichas vistas dentro del dispositivo origen. A estas vistas se les llamará vistas parciales.

1. El usuario entra a una interfaz donde puede apreciar toda su biblioteca digital local (colecciones y documentos) y así poder seleccionar una colección y compartirla, es decir crear una vista: La biblioteca digital local de cada usuario se le presenta a través de una interfaz tipo explorador de archivos donde cada

colección es una carpeta y cada documento es un archivo, para que de esta forma él pueda seleccionar una colección en particular, crear una mini biblioteca digital local y guardarla en su pda para posteriormente enviársela a otro punto.

2. El usuario puede entrar a una interfaz donde se le presentan todas sus vistas parciales para así poder eliminarlas, enviarlas a otro punto conectado o simplemente consultarlas.

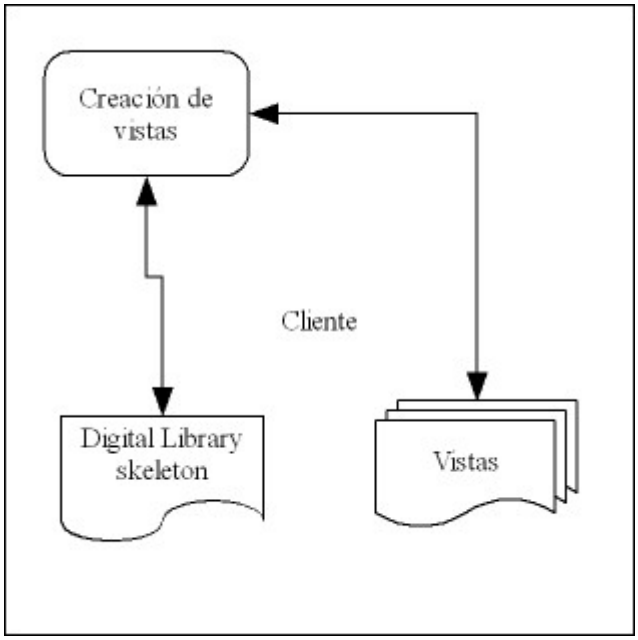


Figura 3.8 Creación de vistas parciales

Escenario 3 - Envío y recepción de una vista

1. El punto “A” y el punto “B” se conectan a la red punto a punto a través del punto privilegiado: Cada uno de los puntos debe enviar un mensaje especial al punto privilegiado pidiendo acceso a la red punto a punto, para que éste a su vez le envíe una lista con todos los puntos conectados a la red y sus direcciones entre otras cosas.
2. El punto “A” a través de un Chat le hace saber al punto “B” que desea que éste le envíe una vista de su biblioteca digital: La aplicación cliente cuenta con un Chat con el que todos y cada uno de los puntos pueden comunicarse entre ellos a través de texto plano.

3. El punto “B” selecciona de entre sus vistas parciales, la vista que le interesa al punto “A”: La aplicación cliente cuenta con una interfaz donde el usuario puede ver eliminar o enviar una vista previamente creada, de tal forma que cuando él lo decida podrá enviar cualquiera de dichas vistas a otros puntos n la red.
4. Una vez que el usuario del punto “B” ha seleccionado la vista a enviar deberá seleccionar de una lista de puntos conectados al punto “A”, para así poder enviarle dicha vista.
5. Una vez que el punto “A” reciba la vista, ésta se almacenará en un repositorio de vistas “foráneas”, para que el usuario pueda consultarla cuando lo requiera.

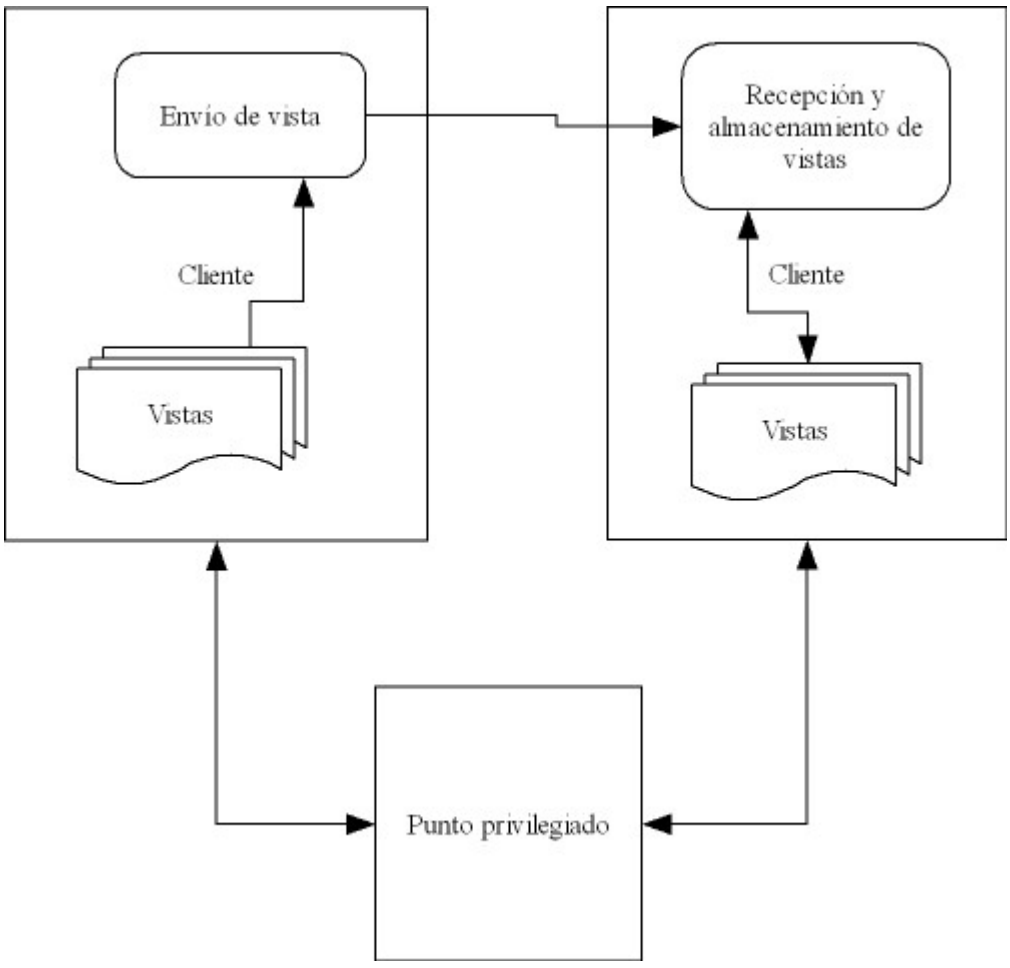


Figura 3.9 Escenario 3 Envío y recepción de vistas

Escenario 4 – Reconciliación por pérdida de conexión

1. El punto “A” y el punto “B” establecen una conexión para la transmisión de algún recurso y cada uno crea una liga de conexión lógica con su similar.
2. El punto “A” se encuentra transmitiéndole al punto “B” un recurso.
3. El punto “B” se queda temporalmente sin conexión.
4. El punto “A” al no recibir respuesta del punto “B” da por pérdida la conexión.
5. El punto “B” se vuelve a conectar a la red y al revisar en sus ligas incompletas encuentra al punto “A” y le envía un mensaje para continuar la transmisión.
6. El punto “A” revisa en la liga para saber en donde se quedó la transmisión y continúa con ella.

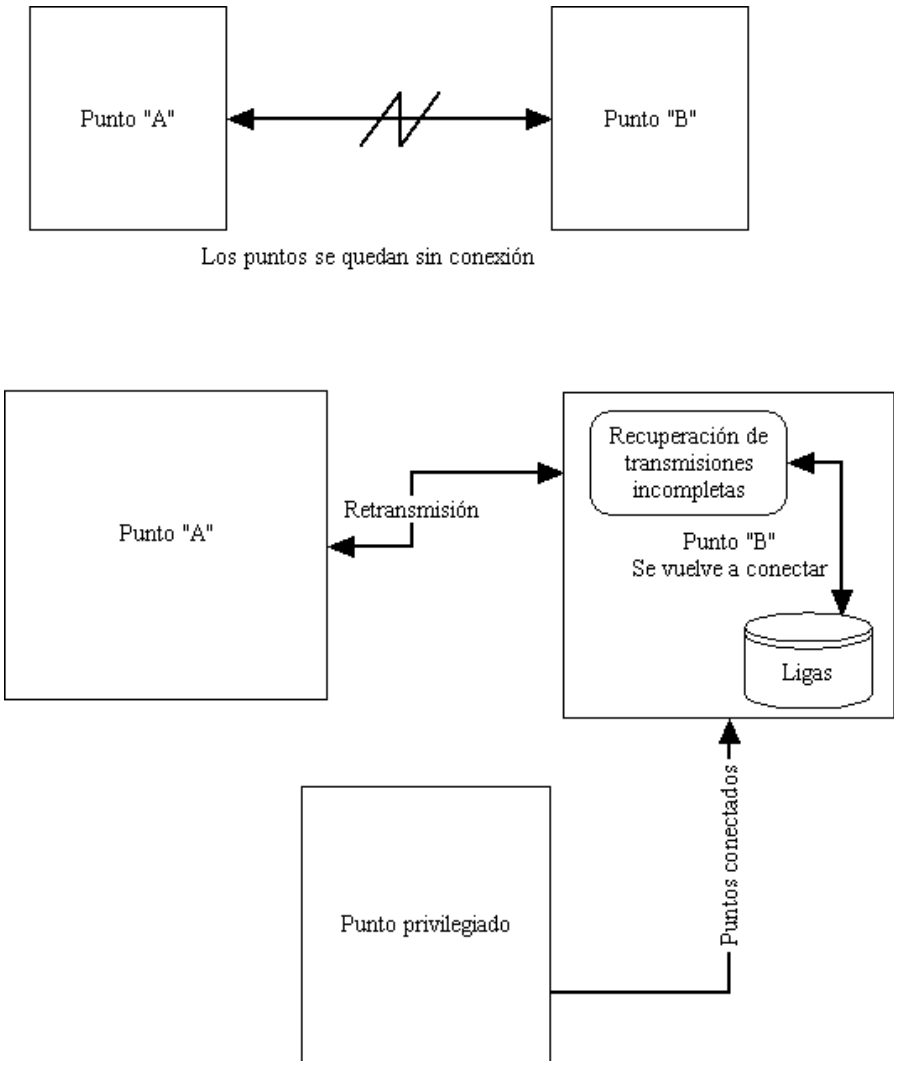


Figura 3.10 Escenario 4 Reconciliación

Escenario 5 –Búsquedas de recursos en la red P2P

1. El punto “A” desea saber cual de los puntos conectados tiene el documento “X”.
2. El punto “A” le envía el nombre del documento al PP como consulta.
3. El PP realiza una búsqueda dentro de su base de datos de vistas de los puntos conectados a la red y encuentra que el punto “D”, “E” Y “F” tienen el documento solicitado.
4. El PP le manda la lista de estos tres puntos con sus direcciones IP’s.
5. El punto “A” se pone en contacto con alguno de estos puntos para pedirle el documento.

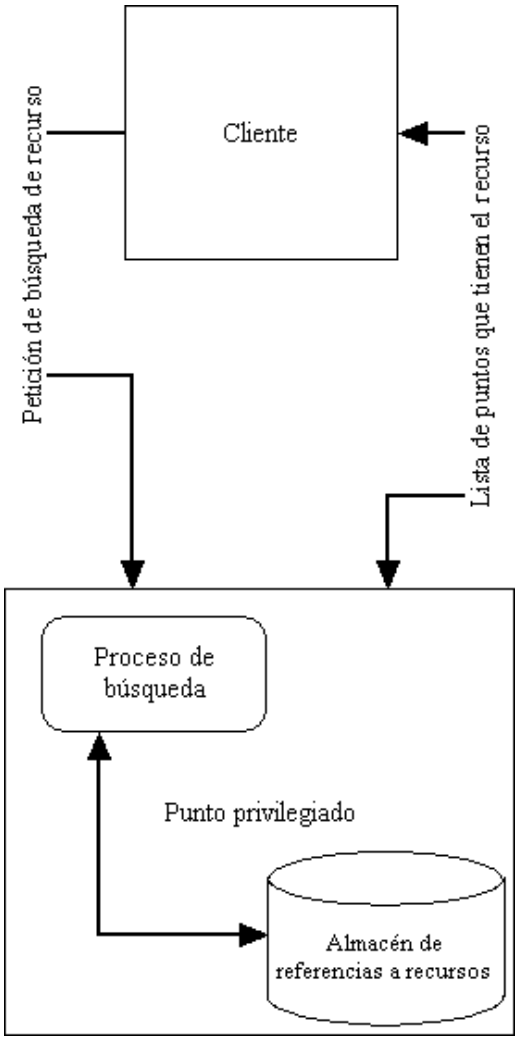


Figura 3.11 Escenario 5 Búsqueda de recursos

Escenario 6 - Conversación a través de Chat.

1. El punto “A” y el punto “B” se conectan a la red punto a punto a través del punto privilegiado: Cada uno de los puntos debe enviar un mensaje especial al punto privilegiado pidiendo acceso a la red punto a punto, para que éste a su vez le envíe una lista con todos los puntos conectados a la red y sus direcciones entre otras cosas.
2. El punto “A” ingresa a la interfaz donde se encuentran todos los puntos conectados al punto “B” para enviarle un mensaje de texto e iniciar así una sesión de Chat.
3. El punto “B” le puede devolver los mensajes y continuar con la conversación.

3.2.4 Punto privilegiado

Dado que en una red punto a punto los dispositivos conectados pueden cambiar de lugar físico, es previsible que su dirección IP también cambie, más aún si los dispositivos de la red son móviles.

Esto es un problema ya que, cuando un punto cambie de lugar físico y luego desee volverse a conectar a la red, los demás puntos no conocerán su nueva dirección; incluso no la conocerían aún cuando fuese la primera vez que éste se conecta, ya que tampoco conoce las direcciones de los demás puntos para poder indicarles de alguna forma que ya esta en red y desea compartir recursos.

Para resolver la problemática anterior es necesario un punto especial, llamado punto privilegiado.

El punto privilegiado será desarrollado bajo la versión estándar de Java (J2SDK 1.4), debido a que deberá estar en alguna computadora de escritorio, y aunque no es necesario que tenga características de servidor, si es necesario que cuente con un poder de cómputo mayor a la de un dispositivo móvil.

El punto privilegiado deberá proveer los siguientes servicios:

- *Indexamiento de los nodos conectados y sus recursos compartidos:* El punto privilegiado deberá proveer el servicio de creación de índices a partir de los puntos y sus recursos compartidos, es decir, cada vez que un punto se conecta a la red, el punto privilegiado recibe un mensaje de su parte en donde, entre otras cosas, deberá

recibir las vistas de dicho punto, para así poder indexarlas. Para que las búsquedas puedan ser llevadas a cabo es necesario contar con alguna forma de relacionar los puntos con sus vistas compartidas (las colecciones y documentos contenidas en las vistas para ser más específicos), para esto se utiliza algo llamado índice que a grandes rasgos, no es más que una relación entre dos cosas, que para este caso sería la relación entre las vistas de cada punto y el punto en cuestión.

- *Procesar las búsquedas hechas por los puntos:* Cada punto podrá crear búsquedas sobre algún recurso y enviárselas al punto privilegiado, para que éste a su vez le devuelva los puntos donde se encuentra dicho recurso. Estas búsquedas son básicamente sentencias SQL en formato de texto plano que el cliente le envía al punto privilegiado, para que éste a su vez las ejecute en su base de datos.
- *Envío de una lista que contenga todos los nodos conectados a la red punto a punto*, así como la información completa propia de cada nodo; por ejemplo, su dirección física, nombre, etcétera. Esto tiene la finalidad que todos los puntos se conozcan entre sí y se puedan comunicar.
- *Autentificar todos y cada uno de los nodos en la red*, para poder enviarle la lista de puntos conectados y darle acceso a la red, a través de las llaves públicas del nodo y de él mismo.
- *Anunciar a todos los nodos la llegada de uno nuevo que desee conectarse a la red*: Cada vez que un nodo nuevo llega a la red punto a punto, el punto privilegiado deberá anunciarlo a todos los demás nodos a través del envío de la lista de nodos conectados.
- *Anunciar a todos los demás puntos en caso de que un punto se desconecte de la red:* De la misma forma en la que el punto privilegiado anuncia la llegada de un nuevo nodo, deberá anunciar también su salida.
- *Generación de identificadores únicos por nodo o punto de la red P2P:* Aprovechando que en el punto privilegiado se puede tener un manejador de base de datos (DBMS), y aprovechando también que en esta base de datos se tendrán todos los nodos de la red punto a punto, podemos asumir que cada uno de ellos contará con un identificador único que puede ser llevado al nivel del identificador global único de toda la red, utilizando siempre validación en esta capa de software para

que dos puntos no se den de alta dos veces. Con esto cada nodo que forma parte de la red punto a punto contará con un identificador que lo distinguirá de forma única de todos los demás.

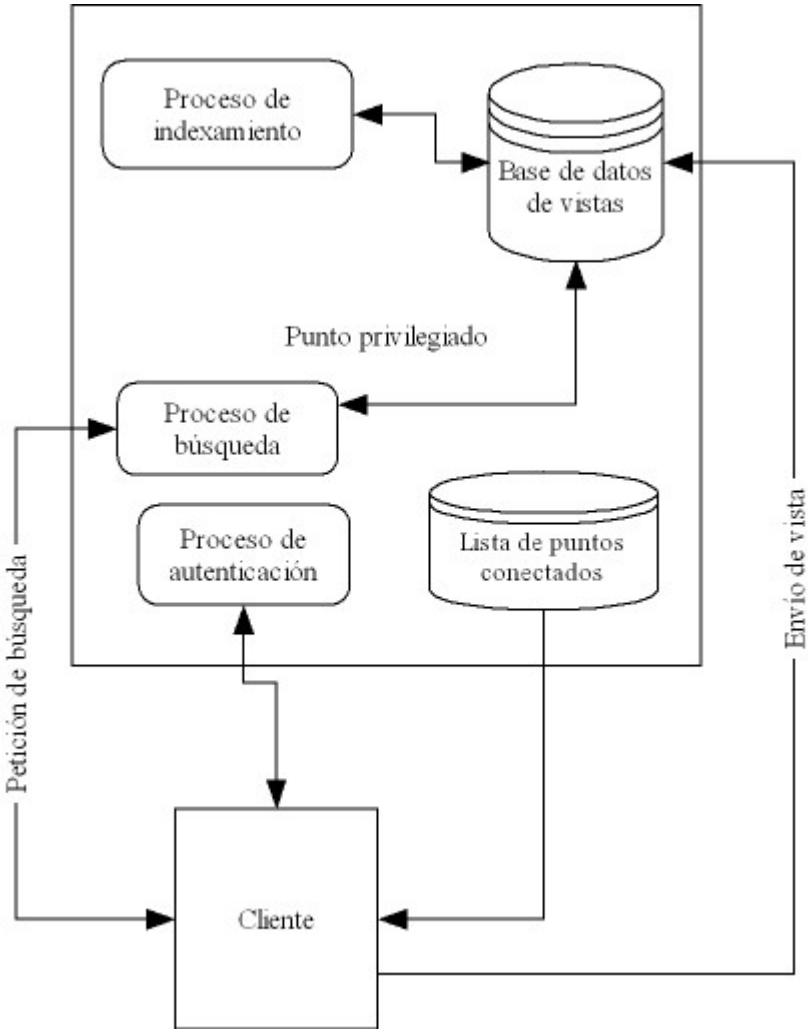


Figura 3.12 Funcionamiento del punto privilegiado

3.3 Retos enfrentados en la realización del proyecto.

A continuación se presentan algunos de los retos que se presentaron al desarrollar el proyecto.

1. Encontrar, evaluar e implementar una tecnología punto a punto para dispositivos móviles que esté hecha en java: La tecnología punto a punto en general es relativamente nueva, no fue sino hasta la década de los noventa cuando fue

explotada a su máximo. Muchas investigaciones son necesarias hoy en día con respecto a este tópico. Dado lo anterior es de esperarse que haya mucho más investigación en esta tecnología aplicada a ambientes móviles, es decir, a dispositivos tales como, PDA's, teléfonos celulares, entre otros, por lo que encontrar, evaluar e implementar una tecnología punto a punto para ambientes móviles desarrollada en java fue uno de los principales retos a resolver.

Para la problemática anterior se detectaron dos tecnologías punto a punto desarrolladas en java para ambientes móviles: JXTA y JRRA. JXTA es un conjunto de protocolos abiertos que permiten conectar cualquier dispositivo en la red, incluyendo PDA's y teléfonos celulares con PC's y servidores para comunicarse y colaborar dentro de un ambiente punto a punto [3]. Esta tecnología aunque muy soportada y extensa todavía no estaba muy madura y tenía algunas deficiencias hablando de dispositivos móviles, además de que es muy compleja para las necesidades del prototipo.

JRRA [4] ofreció una excelente funcionalidad que cubrió muy bien las necesidades del prototipo que son básicamente, ofrecer servicios de bibliotecas digitales en un ambiente punto a punto, además de que al igual que JXTA, son de código abierto.

2. Implementar la tecnología seleccionada en el proyecto que aquí es presentando: Una vez seleccionada la tecnología punto a punto a utilizar fue necesario investigarla, entenderla (incluso a nivel código fuente), para poder implementarla en el prototipo que se presentará en el siguiente capítulo.

   En cuanto a la implementación de JRRA se tuvieron algunos problemas. JRRA tiene dos implementaciones, una para la versión estándar y otra la versión micro de java, sin embargo cuando se trató de llevar a cabo la comunicación entre el punto privilegiado, el cual fue desarrollado con la versión estándar, y los puntos, los cuales fueron desarrollados utilizando la versión micro, se encontró que no eran cien por ciento compatibles, más que nada en cuanto a la forma en que las dos versiones de java utilizan sockets, dado que la comunicación es diferente para un dispositivo móvil que para un dispositivo fijo. La solución a este problema fue implementar la forma de comunicación y sus protocolos de la versión micro de

JRRA en su versión estándar, para que de esta forma pudieran comunicarse sin problemas.

3. Seleccionar las funcionalidades que el prototipo ofrecerá para hacer la demostración y a su vez, aplicarlas a las capacidades de un dispositivo móvil: Dadas las capacidades tan limitadas de un dispositivo móvil, tanto de cómputo como de interfaz de usuario, será necesario realizar un proceso de selección para las funcionalidades y su forma de presentación.

   Las limitaciones de los dispositivos móviles son varias, por ejemplo, para el caso de este prototipo, el mantener la estructura de forma lógica dentro del dispositivo puede llegar a ser un problema, para un primer prototipo se pensó en mantener una base de datos dentro del PDA, sin embargo hasta la fecha no existe una que sea de código abierto, además de que puede crecer mucho, otra posibilidad era en tener un servidor centralizado, como por ejemplo el punto privilegiado, donde se mantuvieran dichas estructuras, pero se podían venir otros problemas tales como: ¿Qué pasa si la conexión se pierde? O ¿Qué pasa si la red es muy lenta?, entre otras cosas. Por lo anterior era un hecho que se debía encontrar la forma de tener la estructura en forma de archivos ASCII, debido a que son muy ligeros, sin embargo con esta infraestructura es muy difícil mantener una estructura de padres e hijos, tal y como es la de una librería digital donde hay colecciones que contienen otras colecciones o documentos, por tal motivo se optó por utilizar XML, que a demás de ser un estándar y ser enteramente archivos ASCII, si se puede mantener una estructura de padres e hijos debido a su naturaleza. Esto ayudó también a otro problema que se presentaba: ¿Qué tanta cantidad de bytes puedo enviar desde un dispositivo móvil a otro?, ya que, uno de los servicios que se proponen es poder enviar y recibir vistas de otros usuarios de la red punto a punto, por lo que el enviar un XML es mucho más fácil y ligero que el enviar una base de datos.

4. La transferencia de documentos a través de la red punto a punto fue un reto a vencer, ya que es necesario mandar tanto los bytes del documento en cuestión, como sus metadatos para que el usuario destino pueda incluirlo en su biblioteca digital local. Dado lo anterior es necesario un mecanismo para poder enviar a través de la red punto a punto dos paquetes o mandar uno solo pero con algún tipo de

separación para saber que parte es el documento y que parte son los metadatos, ya que finalmente al ser enviados son bytes que pasan a través de la red y para que el punto destino los pueda separar y tratar de diferente forma a cada uno es necesario dividirlo al enviarlo. La otra forma es mandar dos paquetes una atrás del otro, sin embargo esto puede traer problemas como deadlocks – Cuando dos proceso están esperando ser desbloqueados entre ellos – debido a que el proceso de inclusión de documento en el esqueleto tiene que esperar hasta que ambos paquetes lleguen y si alguno no llega podría quedarse esperando, por lo que si se implementa este esquema debe de existir un mecanismo para asegurarse de que ambos paquetes lleguen o que se desechen si alguno no llegue después de tiempo considerable. La solución para este problema fue utilizar el esquema de envío de un paquete con los metadatos y el documento juntos, para lo que se utilizará un mecanismo para poder dividir el paquete en dos segmentos, de tal forma que cuando el nodo que reciba el recurso pueda agregar los metadatos a su biblioteca digital y el archivo físico a su repositorio en el dispositivo móvil.

5. Transferencia de recursos por partes: Debido a que el ancho de banda en el que un dispositivo móvil puede transmitir es muy pequeño muchas veces la transferencia no se pueda realizar con recursos muy grandes. Por lo anterior se tiene que implementar un algoritmo para dividir el recurso en paquetes de transmisión más pequeños para así poder enviarlos uno por uno y no saturar el poco ancho de banda que se tiene. La solución de envío por partes lleva consigo una pregunta: ¿De qué tamaño deberá ser cada paquete?. En [1] se implementó un algoritmo, que en base al estado de la red y el tamaño del recurso a transferir se seleccione el tamaño ideal de paquete, a este mecanismo se le llamó adaptación automática a la conexión, sin embargo el objetivo principal de esta tesis fue el de explorar la comunicación P2P y se utilizo un mecanismo sencillo de división de paquetes en tamaños fijos".

   Así mismo es necesario contar con un sistema de acuse de recibo, es decir, que cuando se haya dividido un recurso en partes pequeñas es necesario estar seguros que todas las partes llegaron bien al destino. El sistema de acuse de recibo que se implementará es muy simple, el remitente envía el primer paquete y el destinatario al recibirlo debe enviar un mensaje de que le llegó bien, una vez hecho esto el

remitente le envía el siguiente. Si en medio de una transmisión el remitente o el destinatario pasan un periodo de tiempo establecido sin recibir noticias de la contraparte se dará por perdida la conexión.

6. Reconciliación en caso de pérdida de conexión: Dado a la movilidad a la cual están expuestos los PDA'S, es necesario contar con un mecanismo de reconciliación para poder reestablecer la conexión y que también ayudará a dos cosas: saber que hubo un problema y la transferencia no se completó y no tener que volver a transmitir todo el recurso otra vez. Desafortunadamente este reto no pudo ser resuelto por el tiempo tan corto que se tuvo para desarrollar el prototipo, además de que éste solo era para demostrar que es posible llevar a cabo comunicación punto a punto en dispositivos móviles. Sin embargo en el capítulo tres y cinco se habla un poco sobre el mecanismo y se proveen algunas ideas para su implementación.
7. El parsear un documento XML en el dispositivo móvil con un limitado poder de cómputo fue un reto importante, esto debido a que el proceso utilizado (DOM) es lento y requiere de una cantidad memoria volátil mayor a 10 megas. Se plantearon algunas soluciones, tales como utilizar otros métodos de parseo, sin embargo no se llegó a una solución satisfactoria, debido a que los otros métodos ocupan más poder de proceso (SAX), además de que es mucha más complicado hacerle cambios al documento, por lo que se decidió que el proceso de parseo que en un principio de planteó es el más indicado.

3.4 Conclusiones

En este capítulo se presentó la arquitectura de P2PDLib, se presentaron también los servicios que se utilizaron y de las tecnologías usadas, en este capítulo se habla de una arquitectura ideal con todos los servicios identificados. Básicamente se llegó a la conclusión de que es perfectamente factible el hecho de poder ofrecer servicios de bibliotecas digitales a través de uso de punto a punto en dispositivos móviles, se aprendió de las bondades del XML para transferencia y almacenamiento de datos aprendiendo que es una excelente herramienta para eso, se conoció a fondo la herramienta JRRA y sus servicios, así como de las facilidades de Java como lenguaje de programación para ambientes móviles.

4 Prototipo P2PDLib

En este capítulo se describirán detalladamente el prototipo, y cada una de sus partes, la forma en la que se construyeron, así como los servicios que se ofrecerán.

El prototipo presentando pretende probar la teoría de que es posible incorporar un ambiente punto a punto en dispositivos móviles, ofreciendo servicios de bibliotecas digitales. El servicio más importante que se ofrece es el de la transferencia de recursos entre dispositivos móviles haciendo uso de tecnología punto a punto, esto a raíz que es una de las principales funcionalidades en un ambiente de este tipo. El problema principal que se presenta es el de las desconexiones y reconexiones repetitivas en dispositivos móviles que traen como consecuencia cambios drásticos y frecuentes en sus direcciones de internet.

4.1 Aplicación Cliente

Para la aplicación cliente se hizo uso de la versión micro de JRRA, la cual como ya se mencionó, esta diseñada para dispositivos móviles. Como ya se mencionó también los nodos se comunican a través de sockets, y para la prueba de concepto se utilizarán los de tipo UDP, esto debido a la facilidad que ofrecen, dado que no es necesario crear un comunicación permanente y además tampoco es necesario contar con un servidor para estar esperando peticiones de conexión. La aplicación cliente contará con un menú principal donde el usuario podrá seleccionar el servicio o función que requiera, a continuación se presentarán cada una de las opciones:

1 *Creación de una vista parcial:* Con este mecanismo el usuario puede crear y almacenar vistas parciales para después enviarlas por la red punto a punto a otros usuarios conectados.

2 *Vistas parciales:* En esta opción se le presentan al usuario todas sus vistas parciales, para que éste tenga la capacidad de eliminar alguna o entrar a navegar dentro de una de ellas. También dentro de esta opción el usuario tiene la capacidad de enviar alguna vista a algún oto usuario conectado a red punto a punto.

3 *Vistas foráneas:* Al igual que en la opción de vistas parciales, al usuario se le presentarán las vistas foráneas recibidas, para que de igual forma, éste pueda eliminarlas o entrar a navegar en alguna de ellas.

4 *Puntos conectados:* En esta opción se le muestran al usuario todos los puntos que se encuentran actualmente conectados a la red punto a punto, para saber con cuales puede intercambiar algún tipo de recurso o "chatear".

5 *Chat:* Esta opción le presenta al usuario una pantalla donde puede ingresar algún texto y enviárselo a algún otro usuario de los que se encuentran conectados para poder iniciar una platica tipo "chat".

6 *Mi biblioteca digital:* Aquí el usuario puede navegar dentro de su biblioteca digital local (Digital library Skeleton), de tal forma que puede ver sus colecciones, los metadatos de los documentos. Esta funcionalidad cuenta también con la opción de enviar un documento y/o sus metadatos a otro punto de la red.

7 *Búsqueda de recursos:* En esta opción el usuario puede enviarle el nombre de un documento, el nombre de algún autor o algún otro metadato que puede tener un documento en la biblioteca digital al punto privilegiado, para que este a su vez realice una búsqueda en su base de datos para saber cual o cuales de los puntos conectados tienen el recurso en cuestión y así poder enviárselos al punto que hizo la petición de búsqueda. Cabe mencionar que si el recurso en cuestión es un documento, este puede o no puede encontrarse en el dispositivo móvil de los puntos resultados de la búsqueda, ya que como ya se sabe en el esqueleto están las referencias lógicas a los documentos y estos no necesariamente están en el dispositivo móvil.

En la figura 4.1 se puede apreciar mejor el menú antes presentado.

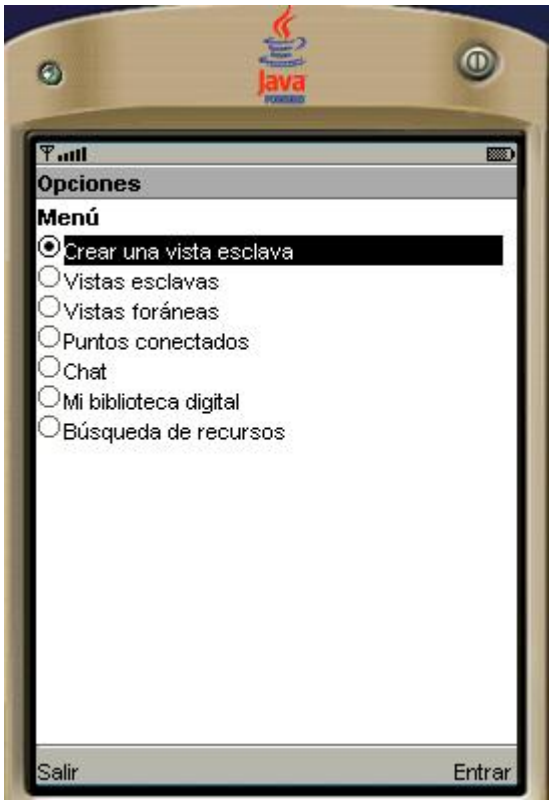


Figura 4.1 Menú principal

4.1.1 Funcionamiento general del prototipo

Como ya se mencionó este prototipo pretende evaluar la factibilidad de comunicación punto a punto entre dispositivos móviles ofreciendo servicios de bibliotecas digitales. Como ya se mencionó también, para demostrar lo anterior se hará uso del proyecto PDLib. Cada usuario de PDLib tiene asociada una o varias librerías digitales las cuales a su vez estas están formadas por colecciones y documentos. Una librería puede ser vista como una estructura jerárquica, en donde, una colección puede tener otras colecciones que depende de ella y documentos.

Esta librería hasta el momento solo se encuentra en la base de datos de PDLib y no en el dispositivo móvil, debido a esto se pensó en una vista lógica de la misma, la cual podría estar en el dispositivo (Digital Library Skeleton), para esto se requiere una sincronización con el servidor de datos y el dispositivo móvil, así mismo se asume también que ya hay algunos documentos físicos y no tienen que ser traídos del servidor de datos.

Los dispositivos móviles tales como teléfonos celulares, pagers y PDA's, normalmente cuentan con dos tipos de memoria, una volátil y otra persistente, es decir una que se pierde al terminar la aplicación que la crea y usa y otra que puede ser utilizada entre aplicaciones y que además no se borra cuando la aplicación que la usa termina.

J2ME define un sistema de base de datos simple llamado RMS (record management system), donde cada conjunto de datos similares es llamado almacén de registros y cada dato en particular es llamado registro [6].

Existe un paquete opcional llamado "paquete opcional de conexión a archivos" o FCOP [5] por sus silgas en inglés, este paquete provee funcionalidad para leer y escribir archivos en dispositivos móviles, sin embargo la configuración CLDC no soporta este paquete y debido a que Palm OS es un sistema operativo muy apegado a dicha configuración tampoco la soporta, por tal motivo no será posible utilizarla ya que el prototipo debe funcionar en estos ambientes.

RMS es una poderosa herramienta para almacenamiento de datos en un dispositivo móvil, sin embargo toda la información almacenada esta en un solo nivel, es decir, no se cuenta con una estructura jerárquica tipo un sistema de archivos (como el explorador de windows), donde se tiene carpetas que contienen a otras carpetas y a su vez archivos.

Para el almacenamiento de los documentos y vistas que estén contenidos en el dispositivo móvil se utilizará RMS, sin embargo como ya se mencionó una vista de una librería digital debe contar con cierta estructura jerárquica y RMS no ofrece este servicio, por lo que se optó por crear una estructura basada en XML, el cual por su naturaleza es jerárquica.

Para la parte del almacenamiento de los esqueletos en el punto privilegiado se utilizó MYSQL [24], que es un manejador de base de datos sin costo. MYSQL cuenta con un mecanismo para la creación de índices de texto completo y búsquedas en ellos. Dado que el prototipo ofrece la funcionalidad de búsqueda de recursos en los esqueletos de las bibliotecas digitales que se encuentren almacenados en el PP se optó por utilizar un índice de este tipo para indexarlos y así poder hacer búsquedas sobre ellos. Con esto es posible ofrecer búsquedas por metadatos, esto debido a que están contenidos en el esqueleto.

Dado lo anterior se puede concluir que hay un archivo XML montado sobre RMS, el cual contiene a los metadatos de las colecciones y documentos así como sus referencias en

al mecanismo de almacenamiento y que además guarda la estructura jerárquica de dichas colecciones y documentos (la biblioteca digital local en sí).

Cabe mencionar que dado que el dispositivo móvil es muy pobre en cuanto a memoria no todos los documentos de la librería digital pueden ser contenidos, así que en el XML se pueden tener una de dos cosas: la referencia al documento en el servidor de datos de PDLib o la referencia en el dispositivo móvil.

En el capítulo 3 se mencionan las llamadas vistas que son un subconjunto de la biblioteca digital local (colecciones y metadatos de documentos), dentro de la aplicación cliente se tienen dos tipos de vistas: foráneas y parciales, las primeras las crea el usuario para enviárselas a otros y las segundas son las vistas enviadas por otros usuarios. Las vistas tienen la finalidad de compartir la estructura y metadatos de parte de su biblioteca digital local.

Para almacenar las vistas maestras, parciales, foráneas, el esqueleto de la biblioteca digital y los documentos contenidos en el dispositivo móvil se cuentan con cuatro RMS's o almacenes de datos:

- Almacén maestro: Contiene únicamente al esqueleto de la biblioteca digital local (Digital Library Skeleton).
- Almacén esclavo: Contiene las vistas parciales.
- Almacén foráneo: Contiene a las vistas foráneas.
- Almacén de documentos: Contiene a los documentos que se encuentren en el dispositivo móvil.

En la figura 4.2 que se muestra a continuación se puede observar las tecnologías utilizadas en el proyecto de manera general, pero ilustrativa.

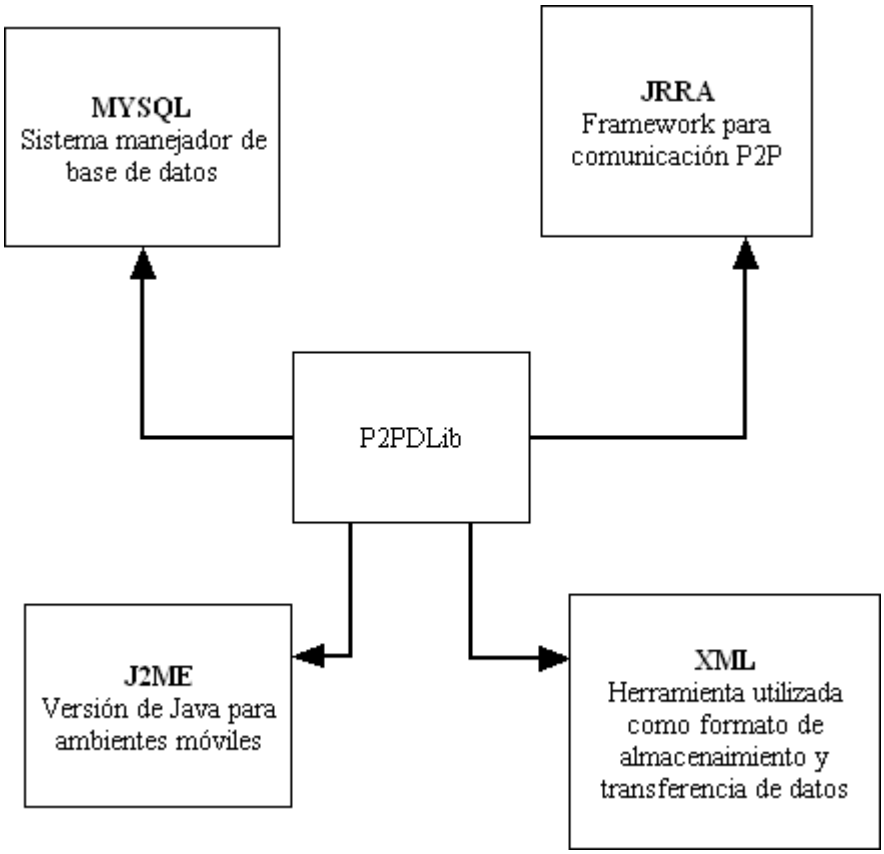


Figura 4.2 Tecnologías utilizadas

4.1.2 Servicios del prototipo

Los servicios que el prototipo ofrece son menores que los que se proponen en la arquitectura general, esto debido a que solo se utilizará para una prueba de concepto y así poder validar el uso de tecnología punto a punto entre dispositivos móviles para ofrecer servicios de bibliotecas digitales. El único servicio que no se implementó fue el de reconciliación por desconexión.

4.1.2.1 Creación y envío de vistas parciales

Cada usuario tiene en su dispositivo móvil el esqueleto de su biblioteca digital y a partir de éste puede crear tantas vistas parciales como necesite.

Para la creación de vistas se desarrolló un modulo en donde el usuario puede seleccionar las colecciones de las que desea hacer una copia, es decir, de lo que desea hacer una vista. Cuando el usuario le da guardar a la vista previamente creada, ésta es almacenada en un RMS llamado slaves, donde se almacenarán todas las vistas parciales creadas, de tal

forma que cuando se desee ver, eliminar o enviársela a otro usuario, basta con barrer dicho RMS y presentarlas.

Figura 4.3 Creación de vistas parciales

Como se puede apreciar en la figura anterior, el usuario debe navegar a través de las colecciones y subcolecciones de su biblioteca digital local en una interfaz tipo árbol creada a partir del XML maestro, para así poder crear su vista esclava posicionándose en la colección que desea compartir, dar clic en “compartir” y finalmente dar clic en “terminar”. Una vez hecho lo anterior el usuario puede visitar las vistas que hayan sido creadas y enviárselas a otros usuarios que estén conectados en la red punto a punto.

4.1.2.2 Envío y recepción de vistas

El envío y recepción de vistas funciona de la siguiente forma: Cuando un usuario desea enviarle un vista esclava a algún otro usuario conectado a la red punto a punto primero debe entrar a la opción “Vistas parciales” del menú principal, para poder ver todas

sus vistas y seleccionar la que mandará, para esto desarrolló un módulo para enviar y recibir paquetes de datos a través de la red punto a punto. Una vista es en realidad un XML que contiene a las colecciones seleccionadas en la opción de crear una vista esclava, cuando una vista es enviada a través de red, en realidad lo que se envía es su XML que debe ser transformada en bytes, que son empaquetados en un mensaje para su interpretación en el cliente destino, para que posteriormente éstas sean almacenadas en un RMS llamado foreigns, de tal forma que una interfaz alterna el usuario puede ver dichas vistas salvadas, navegar entre ellas y eliminarlas si así lo desea.

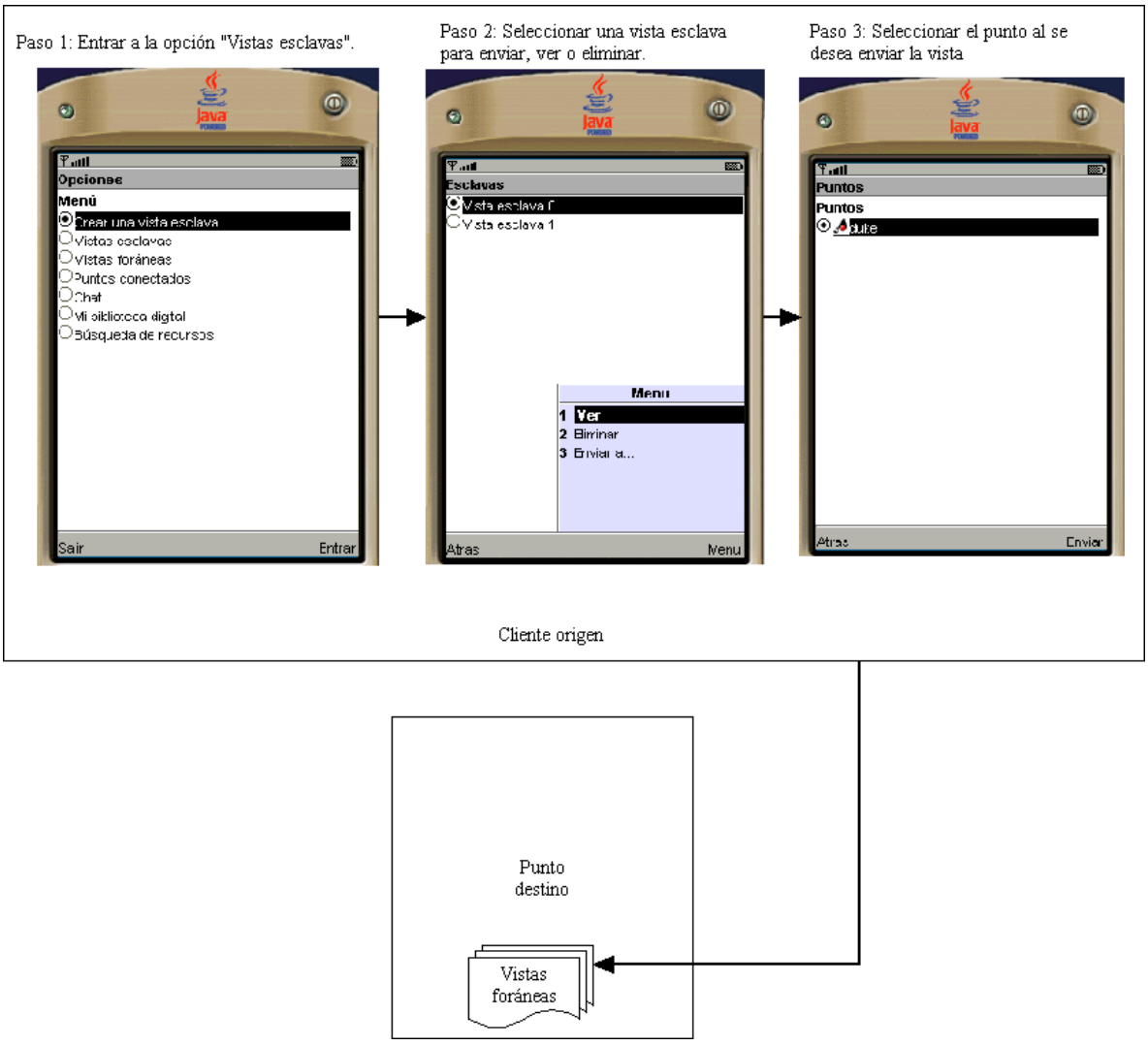


Figura 4.4 Envío y recepción de una vista

4.1.2.3 Envío de documentos de otros usuarios

Debido a que la parte medular del prototipo es la comunicación punto a punto es necesario ofrecer varios servicios que utilicen esta tecnología y entre estos tenemos el envió y recepción de documentos físicos contenidos en el dispositivo. Cuando un usuario desea enviar un documento físico debe entrar a la opción del menú principal llamado “Mi biblioteca digital”, donde puede navegar dentro del esqueleto local (Digital Library Skeleton), y entre otras cosas, enviar un documento físico. Para el envío y recepción de documentos es necesario hacer algo especial, debido a que por cada documento es necesario que se manden también sus metadatos. Debido a esto, se necesita enviar un paquete que contenga al documento en sí y la información sobre este, es decir, los metadatos, para lo cual se creó un paquete que pudiera ofrecer esto, de tal forma que una vez que el destinatario reciba el documento éste pueda incluirlo en su biblioteca digital local con todo y sus metadatos, por lo que se tuvo que desarrollar un módulo para el salvado de un documento en donde el usuario pudiera decidir bajo que colección se iba a almacenar.

Para el caso de los documentos, estos son almacenados en un RMS llamado “documents”, sin embargo además de esto, el usuario tiene la capacidad de agregarlo en cualquier parte de su biblioteca digital para su posterior envío o consulta, por lo que será necesario que sus metadatos sean almacenados en el la librería local del usuario. Cabe mencionar también que existe una colección extra que el usuario no ve en su esqueleto local, dicha colección es utilizada para almacenar los metadatos de los documentos recibidos, independientemente de que también sean guardados en la colección que el usuario haya seleccionado. La colección es llamada UNFILED y básicamente se utiliza para que cuando haya una sincronización con el servidor de datos de PDLib los documentos referidos en dicha colección le sean enviados.

Básicamente el envío de un documento tiene dos disyuntivas:

1. *Envío de solo metadatos del documento:* Dado que no siempre los documentos se encontrarán físicamente dentro del dispositivo móvil, el usuario puede realizar el proceso normal de envío, sin embargo el paquete solo contendrá a los metadatos y del lado de punto receptor el documento se asignará a una colección

dentro de su biblioteca digital local, pero el documentos físicamente no estará en la dispositivo móvil. Los metadatos le indicarán al usuario receptor que el documento no se encuentra físicamente, así como su ubicación en la biblioteca digital del usuario que se lo envió. Esta funcionalidad puede ser proveída también a través del envío de una vista que contenga a la colección donde se encuentra el documento, con la diferencia que dicho archivo no estará contenido dentro de la biblioteca digital local del punto receptor.

2. *Envío de documento físico más metadatos:* El envío de documentos con todo y sus metadatos es prácticamente el proceso descrito anteriormente, pero con la diferencia de que el documento si es enviado y almacenado en el RMS llamado "documents", es decir el paquete de datos contiene dos partes: los metadatos y el documento.

En cualquiera de los dos casos los metadatos del documento son almacenados dentro de la colección UNFILED para su futura sincronización con el servidor de datos de PDLib.

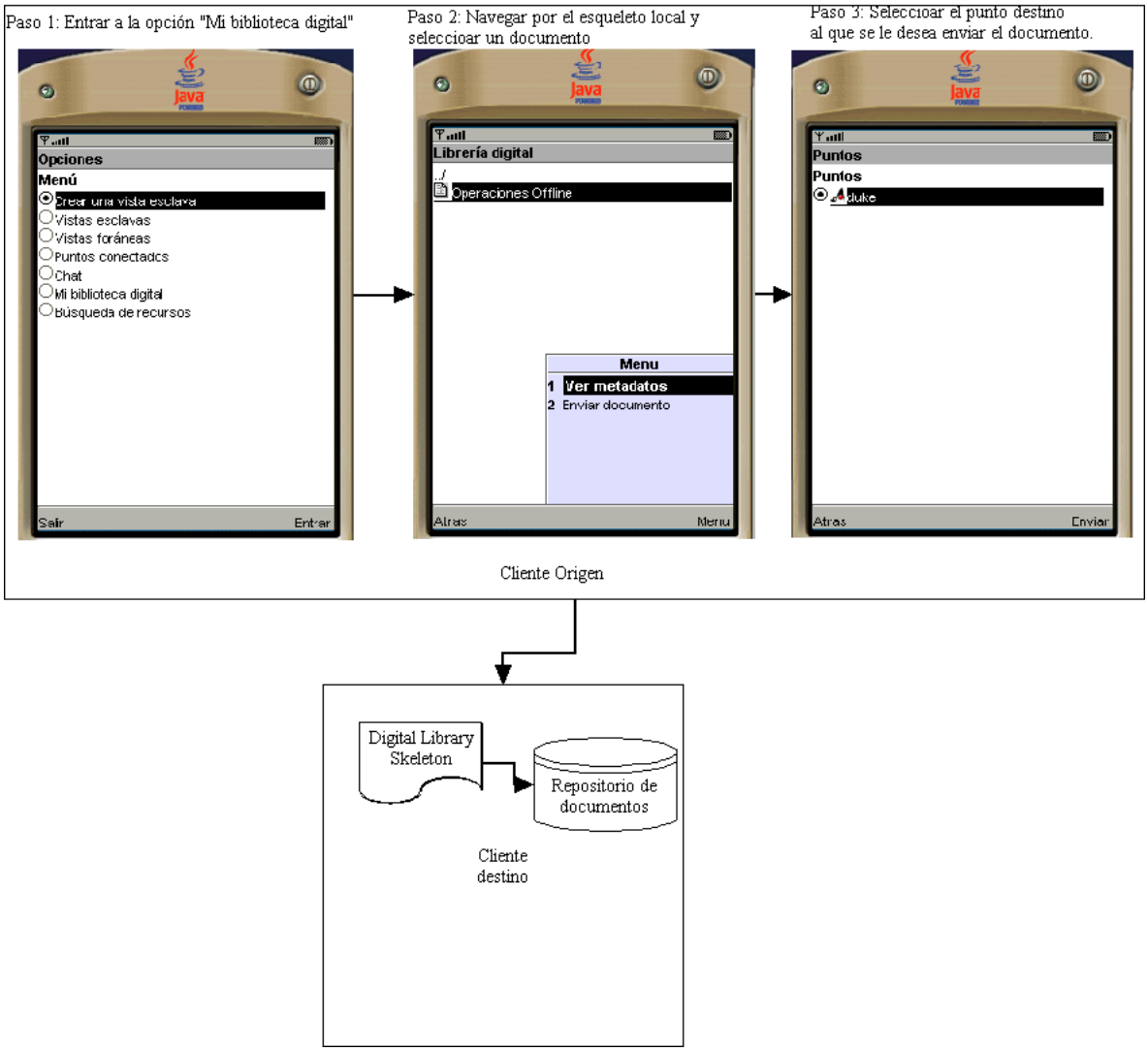


Figura 4.5 Envío y recepción de documentos

En las figuras 4.6 y 4.7 se puede observar la estructura de la biblioteca digital antes y después de que se ha recibido un documento, ya sea en cualquiera de sus dos modalidades: solo los metadatos o documento físico más metadatos. El documento se ha agregado dentro de la colección “MCT” que a su vez es hija se la colección “Graduados ETIE”.

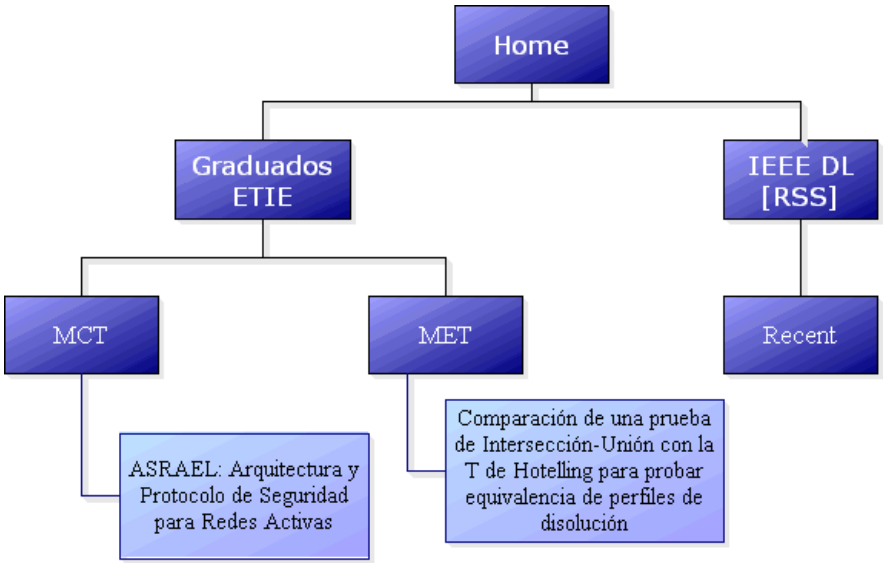


Figura 4.6 Biblioteca Digital local antes de recibir y agregar un documento.

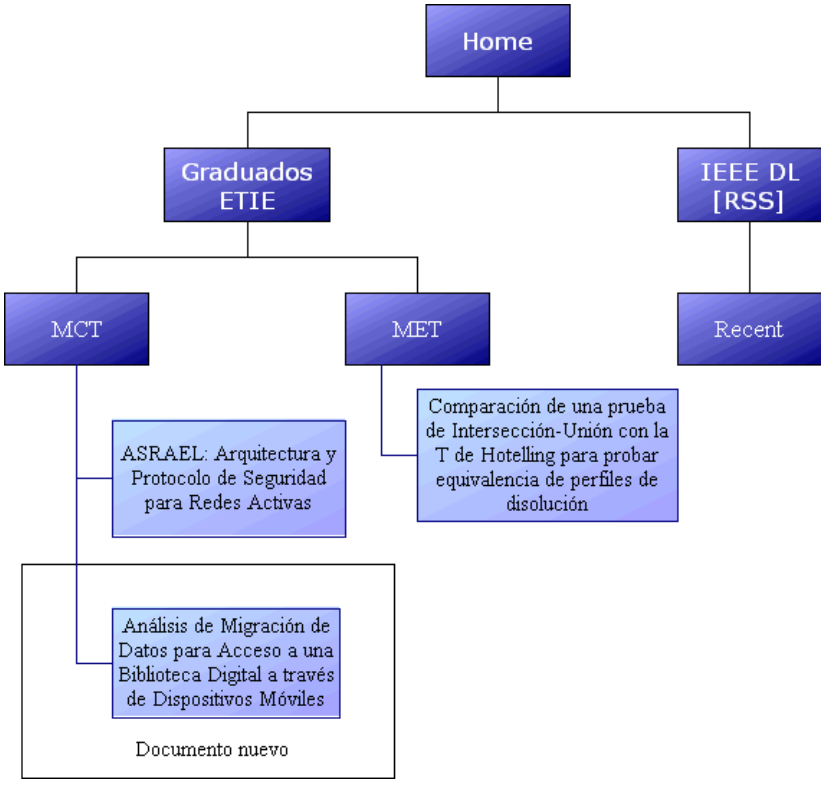


Figura 4.7 Biblioteca digital local después de recibir y agregar un documento.

Las figuras anteriores son abstractas, es decir, dado que una biblioteca digital puede ser vista como una estructura jerárquica se aquí es presentada tipo árbol jerárquico donde

tenemos una colección raíz, “Home” en este caso y ésta tiene otras colecciones que a su vez tienen documentos. Los documentos son los que se encuentran en los cuadros con fondo claro y las colecciones las que se encuentran en cuadros con fondo más fuerte.

En la figura 4.8 se muestra la pantalla del prototipo cuando se ha recibido un documento y se ha agregado a la biblioteca digital local. En este caso la figura muestra el prototipo funcionando, es decir, es una representación real de cómo se verá afectada una biblioteca digital local cuando un usuario reciba un documento y lo agregué a alguna colección.

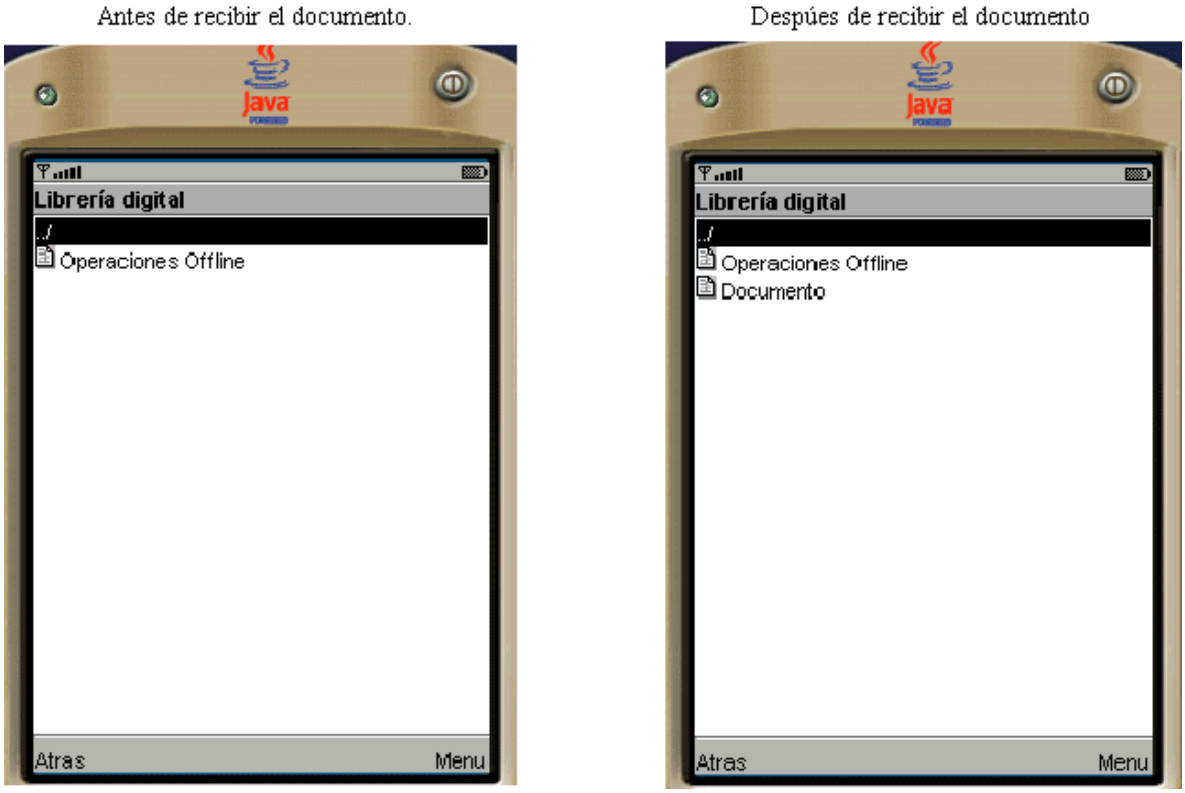


Figura 4.8 Prototipo antes y después de recibir y agregar un documento a la biblioteca digital local.

4.1.2.4 Chat

Por sus características, una red punto a punto permite implementar de forma relativamente simple un servicio de chat. Para este proyecto se implementó un servicio de Chat para poder ofrecer un servicio de comunicación personal entre puntos. Cuando un usuario desea enviar un mensaje de texto a otro debe de entrar la opción “Chat”del menú

principal y así ingresar a una pantalla donde pueda escribir un mensaje de texto, para después seleccionar algún punto conectado para enviárselo.

4.1.2.5 Navegación en la biblioteca digital local

Además de los servicios ya mencionados, el usuario también cuenta con una interfaz para ver su biblioteca digital local. En esta interfaz el usuario puede navegar a través de su librería, ver los metadatos de los documentos o enviar alguno a otro punto conectado en la red punto a punto.

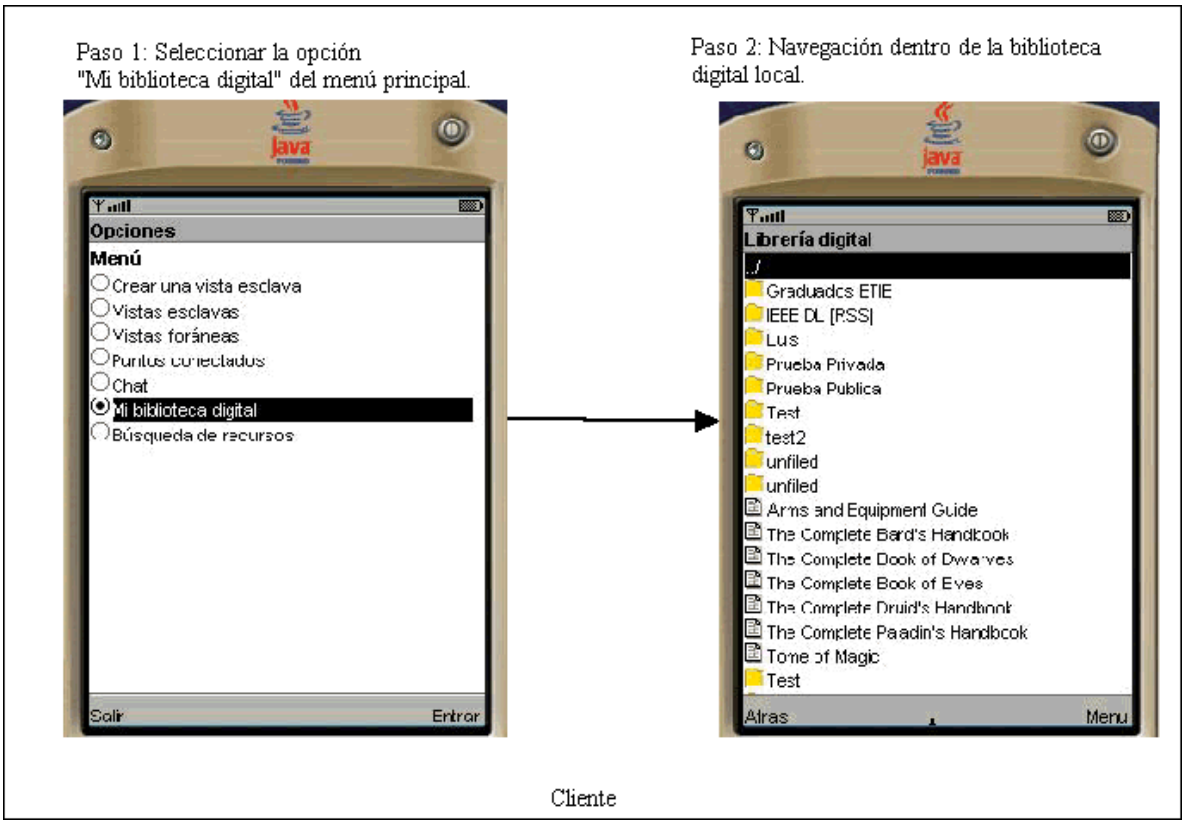


Figura 4.9 Navegación por el esqueleto de la biblioteca digital local

4.1.2.6 Búsqueda de recursos

Como ya se ha mencionado anteriormente cada punto al conectarse al punto privilegiado para hacer uso de la red punto a punto le envía el esqueleto de la biblioteca digital local, para que éste a su vez lo almacene e indexe en su base de datos. El esqueleto debe tener metadatos para las colecciones y documentos en el contenidos, dicho esqueleto es un XML que se almacena en la base de datos del punto privilegiado y sobre este XML se crean los índices para las búsquedas, de tal forma que las búsquedas pueden ser hechas

sobre los metadatos directamente, así un punto puede pedir búsquedas por autor, tipo y nombre. Cabe mencionar que en esqueleto existen más metadatos pero solo se implementaron estos tres para la prueba de concepto. La forma en la que se implementó esta funcionalidad es muy simple, dentro de la aplicación cliente existe una ventana donde el usuario puede escribir una palabra y seleccionar por cual de los metadatos se buscará el recurso (por ejemplo autor = ”Rodrigo Ruiz”), una vez hecho esto esa información es enviada al punto privilegiado para que este a su vez construya la sentencia SQL y la ejecute en el manejador de base de datos, posteriormente la base de datos le regresará los puntos que tienen el recurso haciendo uso del identificador único, y finalmente el punto privilegiado le regresará dichos puntos al punto que haya hecho la petición.

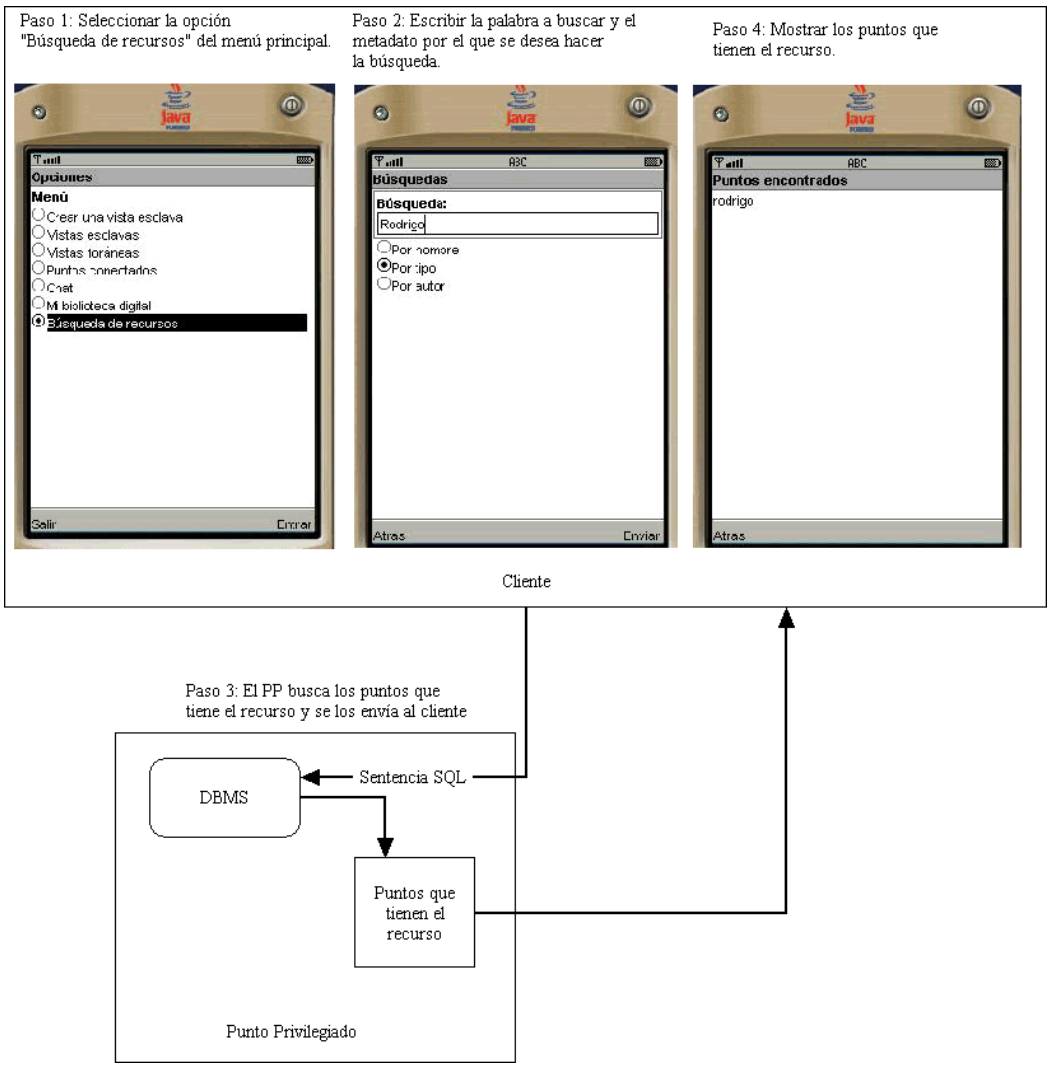


Figura 4.10 Búsqueda de recursos en la red punto a punto.

En la figura 4.10 se puede apreciar la forma en que funciona el mecanismo: El usuario selecciona del menú principal la opción “Búsqueda de recursos”, luego escribe la palabra a buscar, (“Rodrigo” en este caso) y selecciona el metadato sobre el cual se hará la búsqueda, esto es que se buscarán todos los documentos cuyo autor sea “Rodrigo”, finalmente se le presenta al usuario los puntos encontrados, para que éste a su vez los contacte a través de chat.

4.2 Punto privilegiado

El punto privilegiado será el encargado de mediar las conexiones entre los puntos que se requieran conectar a la red, como ya se mencionó anteriormente, debido a que los dispositivos móviles cambian constantemente de lugar geográfico es de pensar que sus direcciones de internet cambiarán también, esto es un problema ya que cuando un nodo intente conectarse a la red debe saber las direcciones de los demás para poder establecer una comunicación, otro problema son las continuas desconexiones a las que se enfrentan este tipo de dispositivos debido a su movilidad, esto es por el hecho de que tienen una red inalámbrica y cuando un usuario se mueve de un lugar a otro puede perder la señal. Dado lo anterior es necesario tener un punto especial llamado punto privilegiado al que todos lo puntos deben conocer para al conectarse a la red le hagan saber que ya están ahí y así poder enviarle su dirección actual y éste a su vez enviársela a todos los demás puntos. También el punto privilegiado es el encargado de mantener una lista de puntos conectados con sus direcciones y propiedades para luego mandársela a los puntos que van llegando, es decir que todos los puntos se conozcan entre ellos para poder llevar a cabo la comunicación punto a punto. El PP cuenta con un manejador de base de datos (DBMS) para almacenar los metadatos de los esqueletos de todos los puntos conectados y así poder ofrecer el servicio de búsqueda de recursos. Esta base de datos cuenta con un índice de texto completo para cada esqueleto, de tal forma que cuando un usuario le pregunta al punto privilegiado quien o quienes tienen el documento “D”, este pueda hacer una búsqueda en todos los esqueletos de todos lo puntos y así saber quienes lo tienen.

Se desarrolló un punto privilegiado haciendo uso de la versión estándar (para computadoras de escritorio) de JRRA, sin embargo cuando se intento hacer la comunicación con los puntos se encontró que no son compatibles, debido a que los sockets

en la versión micro de java funcionan de manera diferente que en la versión estándar y similarmente la versión micro de JRRA hace uso de sockets para J2ME y la versión estándar hace uso de la versión para escritorio de java, por lo que no existe un cien por ciento de compatibilidad entre las dos versiones de JRRA.

Debido a lo anterior se desarrolló una versión de JRRA estándar modificada, donde haciendo uso de el API de java estándar se implementen los sockets de la versión J2ME, de esta forma se hacen compatibles las versiones JRRA micro y estándar sin que ésta última pierda el poder de java 2 versión estándar.

4.3 Servicios del Punto Privilegiado

A continuación se presentan los servicio que el punto privilegiado ofrece para llevar a cabo la comunicación punto a punto.

4.3.1 Mantener en comunicación a todos los puntos conectados

Cada punto que desee conectarse a la red punto a punto debe enviar un mensaje al punto privilegiado con su dirección IP, para que este a su vez les informe a todos los demás puntos que un nuevo participante se ha conectado a la red. Este servicio provee a todos los usuarios la capacidad de saber que puntos están conectados y así poder interactuar con ellos. Cuando el punto “A” se desea conectar a la red, debe de enviarle un mensaje con su información (nombre, dirección IP, puerto) al punto privilegiado, para que este a su vez les informe a todos los demás puntos que ha entrado un nuevo participante dándoles a conocer al mismo tiempo la información necesaria para poderlo contactar.

4.3.2 Almacenar a los esqueletos de cada punto

Cada punto al conectarse al punto privilegiado para integrarse a la red, además del mensaje de petición de acceso ya descrito, le envía su esqueleto con todos sus metadatos para que este sea almacenado en la base de datos. Cada vez que un punto le hace llegar su esqueleto al punto privilegiado este es actualizado en la base de datos en caso de que ya existiera, esto para tener siempre la última versión de dicho esqueleto. Los esqueletos son indexados para que las búsquedas puedan ser hechas sobre todos los metadatos. Cuando se crean índices de texto completo la búsqueda sobre una palabra dentro de una cadena grande de caracteres es mucho más rápida que si se hiciera de forma secuencial.

4.3.3 Proveer de identificadores únicos a los puntos conectados

Cada punto de la red debe contar con un número que lo identifique de los demás, esto no puede ser llevado a cabo por los puntos, es necesario que haya una entidad que asigne identificadores de forma arbitraria para que no haya repeticiones. Haciendo uso de la base de datos que el punto privilegiado tiene para almacenar las vistas de los puntos conectados, se pueden tener también información adicional de cada punto, tal como su nombre (nick) y su identificador el cual debe ser único y es asignado la primera vez que se conecte a la red. Para ejemplificar asumamos que el punto "A" nunca se ha conectado a la red punto a punto y desea hacerlo enviando un mensaje al PP, una vez hecho esto, se le asignará un identificador único que será guardado en la base de datos y que será conocido por todos los demás puntos. Las siguientes veces que el punto "A" se desee conectar a la red punto a punto el PP ya no le creará un nuevo identificador, sino que le asignará el mismo que tenía antes, esto para mantener consistencia en la red.

4.3.4 Búsqueda de recursos

Cada punto puede mandarle al PP el nombre de un documento o su autor para que éste a su vez busque en sus base de datos cuales puntos lo tienen y así poder informárselo al punto que haya hecho la petición. El servicio funciona de la siguiente manera: Un punto cualquiera P conectado a la red, desea saber quien tiene el documento D, y para esto le envía una consulta de tipo SQL (totalmente transparente para el usuario) al punto privilegiado para que éste a su vez la ejecute en la base de datos y pueda obtener como resultados el o los puntos que lo tienen y finalmente enviárselos al punto P. El manejador de base de datos utilizado es MYSQL [24], esto debido a que es gratis, robusto, bastante popular y además cuenta con la funcionalidad de poder crear índices de texto completo, este tipo de índices sirve para poder realizar búsquedas sobre un texto entero, mas o menos como funcionan la mayoría de los buscadores de la internet, por ejemplo google y yahoo.

Para llevar a cabo este mecanismo se hizo uso de dos tablas una llamada node y otra llamada view, en las cuales se almacenan los nodos y sus esqueletos respectivamente. En la tabla de view se tiene un índice de texto completo para que los esqueletos puedan ser mapeados a cada nodo, entonces por ejemplo cuando el PP recibe una petición de búsqueda para saber quien tiene el documento "Operaciones Offline", éste debe hacer un query de la

siguiente forma: “select from view where match(view) against('<Name> Operaciones Offline </Name>' in boolean mode)”, como se muestra en este query es necesario que la búsqueda contenga el tag “Name”, esto limita la búsqueda al metadato nombre, de igual forma las búsquedas pueden delimitarse a cualquier metadato, para efectos de prueba de concepto se implementaron los metadatos de nombre, autor y tipo. El query presentando regresará como resultado a los nodos en los que en su esqueleto se encuentra el nombre del documento buscado.

4.4 Resultados de la experimentación con el prototipo

Después de realizar algunos experimentos con el prototipo se encontraron los siguientes resultados: El parseo del XML del esqueleto de la biblioteca digital en los dispositivos móviles puede causar problemas de memoria cuando el archivo es muy grande en el sentido de tener muchas colecciones, subcolecciones y documentos, esto debido a que se utiliza DOM como método de parseo, DOM sube toda la estructura del XML a memoria, sin embargo se utilizó este método por su facilidad para hacerle modificaciones a la estructura. Estas pruebas se hicieron con un XML de aproximadamente 700 kilobytes que contiene un esqueleto real utilizado actualmente en PDLib. Dado a lo anterior se realizaron algunas investigaciones y se encontró que la memoria a la cual java tiene acceso puede ser incrementada y que por omisión el emulador donde se hicieron las pruebas trae un máximo de 600 kilobytes de memoria, y sabiendo que en una palm se pueden llegar a tener hasta 64 megabytes o más, se procedió a incrementar la memoria del emulador a 20 megas donde el parseo no tuvo ningún problema, salvo el tiempo que requiere este proceso el cual es de aproximadamente 3 minutos para el XML antes mencionado con el que se hicieron las pruebas, dado que esto puede llegar a ser molesto para el usuario se implementó un subproceso, es decir que el usuario puede realizar otras actividades como realizar búsquedas, chatear, enviar o recibir vistas o documentos mientras el esqueleto es cargado en el dispositivo. Se encontró también que la transferencia entre dispositivos y el punto privilegiado en archivos grandes tal como el mismo XML del esqueleto de la biblioteca, puede tener problemas y no llegar a completarse la transmisión, por tal motivo se implementó un algoritmo de envío y recepción por partes, el cual resultó satisfactorio. Se Descubrió también que MYSQL encuentra de manera eficiente las palabras haciendo uso de índices de texto completo y que una base de datos del lado del punto privilegiado no

causa ningún problema de rendimiento y además funciona muy bien para las funcionalidades de identificadotes únicos y búsquedas sobre los metadatos de los esqueletos. Otro resultado interesante que se obtuvo es que al conectarse al punto privilegiado es necesario enviarle el esqueleto local de la biblioteca y esto tiene un periodo de latencia no muy largo sin embargo sería mejor no enviar todo el esqueleto sin solo los cambios hechos a él.

4.5 Requerimientos técnicos

El prototipo funciona en cualquier dispositivo móvil que tenga instalada la versión micro de java (J2ME) y que cuente con soporte para MIDP 2.0 y CLDC 1.1, esto debido a que hace uso de algunos protocolos de red tales como sockets y tcp/ip. Los dispositivos más comunes que cuentan con este tipo de infraestructura son la palm y los teléfonos celulares.

4.6 Limitaciones

Los dispositivos móviles conocidos como pocket pc que tienen el sistema operativo windos ce, requieren una versión de java diferente a que distribuye sun, como por ejemplo j9 de IBM, sin embargo no se asegura el funcionamiento completo del prototipo.

4.7 Conclusiones

En este capítulo cuatro se llegó a la conclusión de manera experimental que es posible ofrecer servicios de bibliotecas digitales a través de comunicación punto a punto en ambientes móviles. Así mismo se llegó a la conclusión de XML es una poderosa herramienta para transferencia de datos y almacenamiento, esto debido a que es un estándar y no se necesita algún software especial para su uso (solo un paser de XML), esto ayudó bastante a poder tener metadatos de documentos y colecciones en el dispositivo móvil debido a la estructura jerárquica de árbol que provee de manera natural un XML, además de que en este tipo de arquitecturas es muy complicado tener un manejador de base de datos o algo parecido para almacenar datos. Se investigó a fondo la arquitectura de JRRA y se encontró que es una excelente herramienta para un ambiente simple en J2ME, sin embargo es buena idea revisar a JXTA en su última versión micro, ya que promete ofrecer mejores soluciones a futuro.

5 Conclusiones y trabajos futuros

5.1 Conclusiones

El crecimiento del número de usuarios de dispositivos móviles y el crecimiento en la infraestructura de comunicaciones provocan que cada vez un mayor número de usuarios requieran tener iteración con las aplicaciones que normalmente utilizan con dispositivos con mayores capacidades de cómputo [14]. Una de las aplicaciones que los usuarios utilizan normalmente son las bibliotecas digitales, hoy en día existe una basta cantidad de información disponible en forma electrónica: documentos, imágenes, multimedia y video son algunos de los formatos disponibles en lo que podemos encontrarla, debido a esto existe la necesidad de poder contar con repositorios para almacenar dicha información y poder consultarla en cualquier momento. Por lo tanto y teniendo en cuenta lo anterior, el poder tener una biblioteca digital enfocada hacia usuarios de dispositivos móviles para satisfacer las dos necesidades antes citadas, es una gran contribución.

La tecnología punto a punto, también llamada cómputo de puntos, es un paradigma emergente, que actualmente se ha visto como una tecnología potencial que puede cambiar de manera radical las arquitecturas distribuidas [16].

5.2 Aportaciones

La contribución principal de esta tesis es la creación de un ambiente punto a punto en dispositivos móviles para ofrecer servicios de bibliotecas digitales, de tal forma que haciendo uso de una tecnología nueva pero popular y poderosa se satisfaga una necesidad actual de la comunidad tecnológica. Sin embargo a continuación se presentan las aportaciones específicas de la tesis:

- Se identificaron los servicios de bibliotecas digitales que puede ser ofrecidos a través de una arquitectura punto a punto. Estos servicios son mencionados mas adelante.
- Se identificó la información de PDLib que debe mantenerse en el dispositivo móvil, esto se refiere a los metadatos de la biblioteca digital que deben mantenerse de manera local en el dispositivo, tales como nombre de la

colección o documento. Para más información favor de referirse a la figura 3.5 del capítulo tres.

- Se diseñó una estructura para almacenar los datos de la biblioteca digital en el dispositivo móvil. Se diseñó una estructura en forma de árbol contenida en un XML donde se encuentran las colecciones, documentos y metadatos guardando una estructura tipo árbol propia de una biblioteca digital. Para más información favor de referirse a la figura 3.5 del capítulo tres.
- Se construyó un prototipo funcional como prueba de concepto para validar le hecho de poder ofrecerse servicios de bibliotecas digitales en dispositivos móviles a través de una arquitectura punto a punto.
- Se evaluó la viabilidad de ofrecer servicios de bibliotecas digitales a través de una arquitectura punto a punto. A través del prototipo se pudo observar que es factible ofrecer servicios de bibliotecas digitales a través de punto a punto en dispositivos móviles.
- Se investigaron otros esfuerzos para el desarrollo de arquitecturas punto a punto, sin embargo no se encontró ninguno que funcionara de manera adecuada en dispositivos móviles.
- Se evaluó el desempeño del XML como almacén del esqueleto de una biblioteca digital: Se encontró que el parseo utilizado (DOM) no es muy eficiente, sin embargo satisface las necesidades del prototipo.
- Se adaptó la versión de JRRA versión estándar para trabajar con la versión micro, ya que como se mencionó anteriormente no eran totalmente compatibles.

Esta tesis se basa en resolver algunas de las necesidades del proyecto de bibliotecas digitales en dispositivos móviles llamado PDLib [17]. Básicamente las necesidades detectadas en PDLib son las siguientes: primero, existe una dependencia total con el servidor de datos, es decir que para que un usuario pueda consultar su biblioteca digital, obtener algún documento o realizar cualquier otra operación es necesario que este comunicado con el servidor de datos, aunque existe una forma de enviarle a otro usuario de

PDLib un documento contenido en la biblioteca digital a través de correo electrónico los metadatos no son incluidos en el mensaje, no es posible que un usuario de PDLib conozca la biblioteca digital de otro usuario. La propuesta aquí presentada resuelve las necesidades antes descritas, implementado un ambiente punto a punto y ofreciendo los siguientes servicios principales:

1. Transferir documentos entre puntos sin la necesidad de tener comunicación con el servidor de datos: Por medio de este servicio es posible que los puntos que se encuentren conectados a la red puedan compartir sus documentos físicamente o solo los metadatos.
2. Contar con una copia del esqueleto de la biblioteca digital en el dispositivo móvil: Dentro del dispositivo móvil se tiene una copia local de la biblioteca digital contenida en el servidor de datos, cabe mencionar que esta copia es solo el esqueleto de la misma, con este servicio el usuario puede ver su biblioteca digital en cualquier momento para su consulta.
3. Poder compartir vistas de la biblioteca digital local entre los usuarios de la red punto a punto: Cada usuario de P2PDLib puede crear subconjuntos de su biblioteca digital local principal, para luego poder compartirlas con los demás usuarios y de esta forma permitir que otros usurarios conozcan parte de su biblioteca digital.
4. Un servicio de chat para mantener comunicación entre los puntos: Con este servicio todos los usuarios de P2PDLib pueden estar en contacto por medio de un servicio de chat, es decir intercambio de mensajes de texto.
5. Búsquedas de recursos en los metadatos de los esqueletos de los puntos: A través de este servicio todos los puntos pueden hacer peticiones de búsquedas de recursos entre los demás puntos a través del punto privilegiado.

5.3 Trabajos futuros

A continuación se hablará de los trabajos futuros propuestos para su futura implementación y de esta forma hacer más robusto y escalable el ambiente punto a punto.

5.3.1 Mecanismo de reconciliación

Debido a la gran cantidad de desconexiones que puede sufrir un dispositivo móvil es necesario tener un mecanismo de contingencia en el caso de que esto suceda en medio de

una transmisión. A este mecanismo se le llama reconciliación debido a que trata de recuperar la transferencia en el lugar donde se quedo si necesidad de volver a transmitir todo desde el principio. La idea es crear ligas lógicas entre el receptor y el transmisor, dichas ligas estarán almacenadas en un XML y tendrán algunos atributos necesarios para la reconciliación, tales como los ID's de los nodos, el número de paquetes que forman el recurso completo, etcétera. Después de una desconexión, los dos nodos conservaran las ligas para que cuando uno de los dos se vuelva a conectar revise en su XML de ligas aquellas que no hayan sido terminadas y pueda contactar al punto con el que se estaba llevando la transmisión. Si la transmisión es llevada a cabo de manera satisfactoria la liga puede desechada. Roberto García en su trabajo de tesis [1], implementa un mecanismo de reconciliación similar al aquí propuesto, esta podría ser otra forma de resolver el problema.

5.3.2 Implementación de más de un punto privilegiado

Dado que una red punto a punto puede llegar a ser muy grande en cuanto a cantidad de nodos conectados se refiere, es necesario poder contar con más de un punto privilegiado, sin embargo esta no es una tarea fácil debido a que entre los puntos privilegiados debe haber una sincronización de los nodos conectados a la red y de las bases de datos de cada punto privilegiado, es decir que se debe contar con una arquitectura de bases de datos distribuidas para soportar tales funcionalidades.

5.3.3 Escalabilidad del sistema

Como se mencionó en este documento cuando un nodo se conecta a la red punto a punto debe enviar un mensaje con su dirección IP actual, al PP y este a su vez debe hacer un broadcast (mensaje para muchas entidades) a todos los demás puntos de la red, esto puede llegar a traer problemas cuando el número de nodos conectas sean muchos, por tal motivo se deja como trabajo futuro implementar un mecanismo mejor. Una posible solución a esto puede ser manejar un algoritmo de "bajo demanda", de tal forma que no se haga un broadcast a todos los nodos. Lo anterior puede ser llevado a cabo de la siguiente forma: si se implementan varios PP'S, cada uno puede tener como "suyos" a un subconjunto de todos los nodos conectados y solo hacer broadcast a estos. Cuando un nodo desee contactar a un nodo de otro punto privilegiado, el nodo interesado puede enviarle el identificador único (obtenido de una búsqueda) o el nombre de usuario para que este a su

vez pregunte entre los demás puntos privilegiados quien tiene al nodo en cuestión y así hacerle llegar al punto en interesado la dirección IP actual de éste último siempre y cuando este conectado. Otra implementación para hacer más sencillo este mecanismo puede ser el guardar una tabla en el dispositivo móvil con todos los nodos con los que ha tenido comunicación (sus nombres de usuario) contra sus identificadores únicos para así poder tener una lista de puntos posibles a contactar. Esta lista por supuesto irá creciendo de forma dinámica para que en un momento dado se tengan todos los participantes o por lo menos la gran mayoría.

5.3.4 Mejoras al parseo del archivo XML

Como ya se mencionó, en las pruebas realizadas al prototipo, se encontró que el tiempo de parseo es un poco lento, por tal motivo es necesario implementar otra manera de hacerlo, existe otro método de parseo llamado SAX, sin embargo no ofrece la posibilidad de hacer cambios al XML en su estructura de manera simple, además de que si se hicieran los cambios a través de SAX se tendría que estar parseando el XML cada vez ya que este método no guarda nada en memoria y se pueden tener problemas de procesamiento.

5.3.5 Sincronización de documentos con el servidor de datos de PDLib

Como ya se mencionó en los capítulos tres y cuatro, existe un servicio de transferencia de documentos entre puntos, así como que el punto que recibe puede seleccionar una colección de su biblioteca digital y meter ahí los metadatos del nuevo documento para que en una futura navegación por el esqueleto ese documento ya este ahí. Sin embargo este servicio trae consigo una problemática, ¿Cómo se va a llevar a cabo la sincronización de documentos y sus metadatos con el servidor de datos de PDLib?, para responder a esta pregunta es necesario hacer referencia al trabajo de tesis de Luis Basto Díaz llamado “Modelo de Soporte de Operaciones Offline en Dispositivos Móviles para una Biblioteca Digital**”** [27], el cual es utilizado para obtener la primer versión del esqueleto con el que la arquitectura punto a punto trabaja (Digital Library Skeleton). Dicho trabajo provee al usuario de PDLib con ciertos servicios de operaciones fuera de línea, es decir que el usuario pueda crear, actualizar o eliminar colecciones y/o documentos sin necesidad de estar conectado al servidor de datos, de tal forma que al conectarse se sincronicen los cambios hechos. La forma en la que Luis Basto realiza esta tarea es creando

y almacenando una seria de comandos para cada operación fuera de línea de tal forma que al conectarse estos comandos puedan ser ejecutados en el servidor de datos, de entre estos comandos se cuenta con uno para crear un nuevo documento y almacenarlo en la biblioteca, dicho comando sería el que idealmente se debe implementar para llevar a cabo la sincronización con el servidor a la hora de recibir un documento a través de la red punto a punto para su futuro almacenamiento en PDLib.

Apéndice I - Códigos y diagramas UML

Diagramas UML y códigos más importantes de las clases creadas en el prototipo

A continuación se presentan algunos de los diagramas UML y códigos más importantes de las clases creadas para el desarrollo del prototipo, así mismo se presenta una breve descripción de la forma en la que se estructuró y organizó el código fuente. En estos diagramas y códigos se pueden apreciar los métodos utilizados, las propiedades de cada clase y sus dependencias con otras clases. Además, se presenta una breve descripción de cada una de las clases y su uso de manera muy general.

Con estos diagramas en UML cualquier otro desarrollador puede darse una idea de manera abstracta lo que hace cada clase y para qué sirve. Cabe mencionar que las clases presentadas son las que fueron utilizadas para ofrecer los servicios del prototipo, es decir las clases más importantes para ofrecer servicios de bibliotecas digitales.

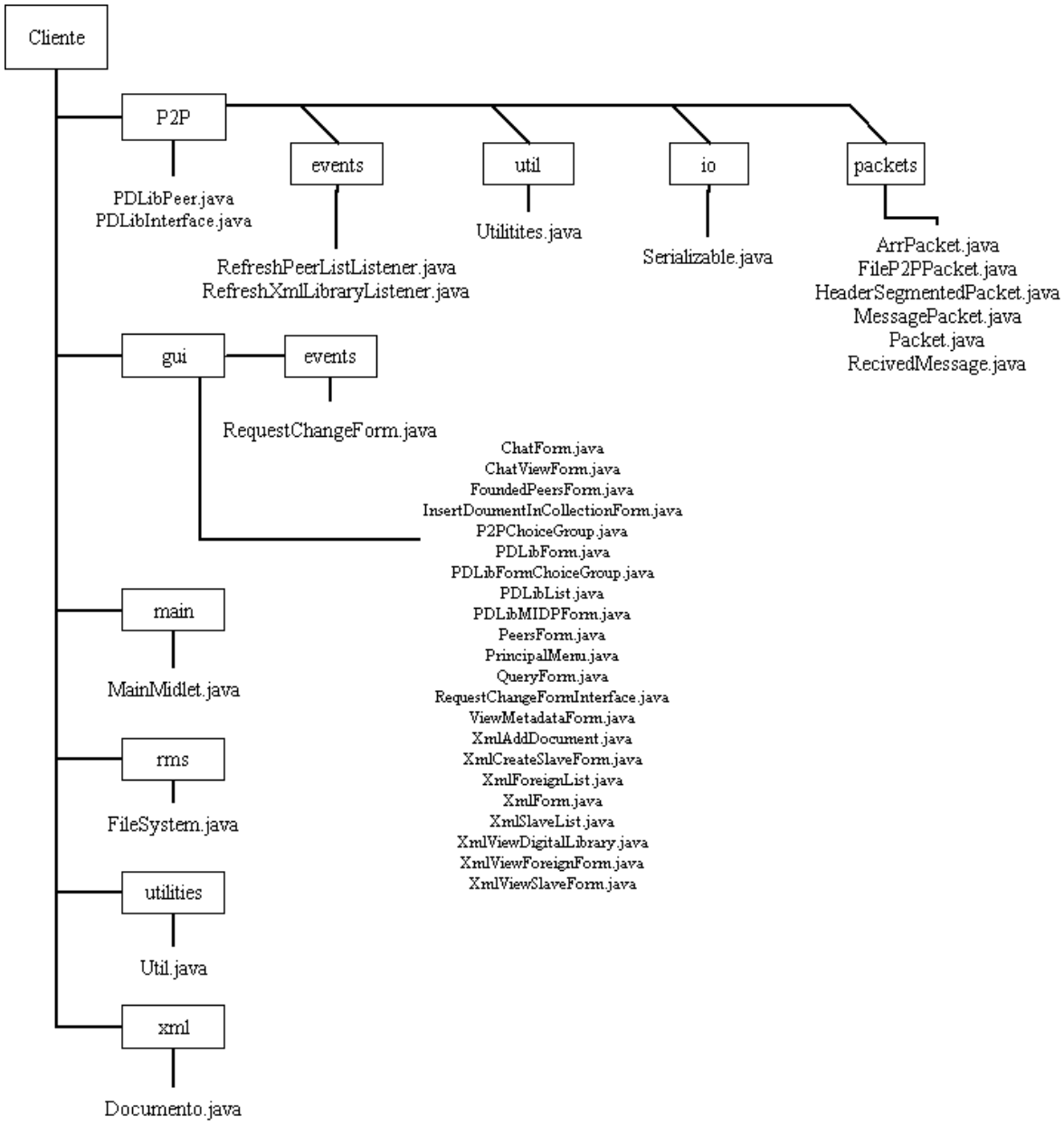
Aplicación cliente

El código de la parte del cliente esta organizado y estructurado de la siguiente forma: Existe una clase de suma importancia llamada PDLibPeer, la cual es la encargada de inicializar al punto y mantener comunicación con los demás puntos y con el PP. Existen también seis interfaces muy importantes: PacketListener, PDLibInterface, RequestChangeFormInterface, RefreshPeerListListener, RefreshXmlLibraryListener y RequestChangeForm, la primer interfase sirve para avisar de la llegada de un paquete al punto, la segunda es una interfase que deben implementar todas las clases del cliente debido a que provee métodos para la obtención y fijación de la referencia a la instancia del objeto Peer (el punto del cliente en cuestión), la tercera provee métodos para avisar a otras clases de la necesidad de un cambio de ventana, la cuarta es la encargada de avisar de la necesidad de refrescar la lista de puntos, la quinta sirve para avisar de la necesidad de refrescar el esqueleto de la biblioteca digital y la sexta es para informar que se debe realizar un cambio de ventana. Como se puede observar las interfaces juegan un papel muy importante esto es por que la aplicación está organizada de forma que todos los cambios tanto en interfaz gráfica como en la llegada de paquete sea a través de eventos y de esta

forma tener un código más limpio sin tantas referencias entre clases, proveyendo de un modelo parecido al MVC [28], de esta forma podemos tener clases que observan (las que implementan las interfaces) y clases observadas (las que mantienen referencia a cada observador a través de la interfase). Existen unas clases especiales que sirven para abstraer a los objetos que serán enviados por la red, todos estos objetos deben implementar la interfaz PacketData, que provee métodos para serializarlas y enviarlas a través de la red de tal forma que se puedan compartir objetos (un mensaje, un documento, etcétera).

La aplicación cliente esta organizada en seis paquetes (así se le llama en java a las carpetas que contienen clases dentro) o carpetas principales: p2p, gui, main, rms, utilities y xml los cuales contiene otros paquetes y clases que proveen funcionalidad que va acorde con su paquete, es decir que dentro del paquete p2p están contenidas las clases que sirven para proveer a la aplicación de funcionalidad para conectarse a la red punto a punto, enviar paquetes y mantener al punto lo más actualizado posible en cuanto a la situación de la red P2P de refiere, dentro de este paquete se encuentra otro paquete llamado packets donde están contenidas las clases que representan a los objetos que serán enviados por la red, el paquete gui contiene a todas las clases que sirven para crear la interfaz gráfica (GUI), el paquete main contiene a una sola clase llamada MainMidlet, la cual como su nombre lo indica es el midlet principal, dentro del paquete rms se encuentran algunas clases para acceso a lo que sería el equivalente de un sistema de archivos en un computadora de escritorio, en el paquete utilities se encuentran clases de uso común para todas las clases y por último dentro del paquete xml se encuentran clases para el manejo de los XML’S.

A continuación se muestra una figura donde se puede observar los paquetes y las clases que contienen de manera gráfica.


Cliente
P2P
PDLibPeer.java
PDLibInterface.java
events
RefreshPeerListListener.java
RefreshXmlLibraryListener.java
util
Utilitites.java
io
Serializable.java
packets
ArrPacket.java
FileP2PPacket.java
HeaderSegmentedPacket.java
MessagePacket.java
Packet.java
RecivedMessage.java
gui
events
RequestChangeForm.java
ChatForm.java
ChatViewForm.java
FoundedPeersForm.java
InsertDoumentInCollectionForm.java
P2PChoiceGroup.java
PDLibForm.java
PDLibFormChoiceGroup.java
PDLibList.java
PDLibMIDPForm.java
PeersForm.java
PrincipalMenu.java
QueryForm.java
RequestChangeFormInterface.java
ViewMetadataForm.java
XmlAddDocument.java
XmlCreateSlaveForm.java
XmlForeignList.java
XmlForm.java
XmlSlaveList.java
XmlViewDigitalLibrary.java
XmlViewForeignForm.java
XmlViewSlaveForm.java
main
MainMidlet.java
rms
FileSystem.java
utilities
Util.java
xml
Documento.java

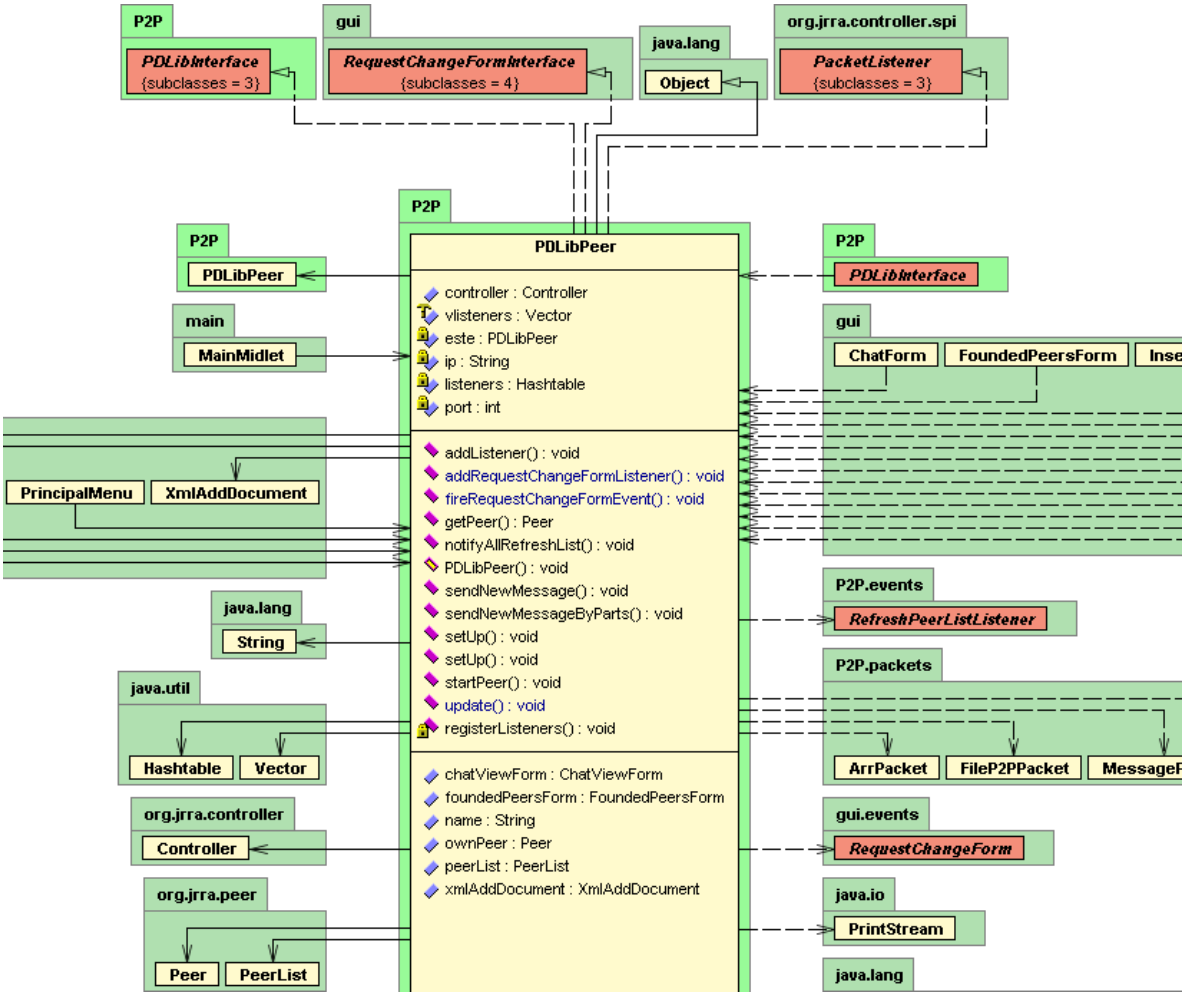
A continuación se presentan los diagramas UML y los códigos de la parte cliente, es decir la que se debe instalar en el dispositivo móvil.

PDLibPeer.java

Clase principal que abstrae a un punto de la red. Aquí se inicializa el punto. Esta clase provee métodos para el recibo y envío de mensajes de y para otros puntos.

Diagrama

El siguiente diagrama fue recortado de su tamaño original por que es demasiado grande, sin embargo muestra lo más importante de la clase.



Código

```
public class PDLibPeer
    implements PacketListener, PDLibInterface, RequestChangeFormInterface {
```

```
/** Crea una nueva instancia de PDLibPeer */
public PDLibPeer() {
  listeners = new Hashtable();
  peerList = new PeerList();
  chatViewForm = new ChatViewForm("Mensaje");
  xmlAddDocument = new XmlAddDocument("Agregar documento", List.IMPLICIT, null, this);
  foundedPeersForm = new FoundedPeersForm("Puntos encontrados", este, null);
}

/**
 * Inicia el controlador del punto y realiza el envio al PP para entrar a
 * la red P2P
 */
public void startPeer() {
  System.out.println("IP: " + utilities.Util.getOwnIp());
  setUp(utilities.Util.port, utilities.Util.getOwnIp(), utilities.Util.name);
  RegisterPeerRequestPacket reqPacket = new RegisterPeerRequestPacket(ownPeer);
  sendNewMessage(utilities.Util.getPrivilegedPeer(), reqPacket);
  try {
    Thread.sleep(5000);
    FileSystem fs = new FileSystem("PDLib");
    byte[] library = fs.getRecord(main.MainMidlet.idMasterRecord);
    sendNewMessageByParts(utilities.Util.getPrivilegedPeer(),
                    utilities.Util.segmentMessage(library));
  }
  catch (Exception e) {
    e.printStackTrace();
  }
}

public ChatViewForm getChatViewForm() {
  return chatViewForm;
}

public FoundedPeersForm getFoundedPeersForm() {
  return foundedPeersForm;
}

public XmlAddDocument getXmlAddDocument() {
  return xmlAddDocument;
}

/**
 *  Setea la dirección IP, el purto y el nombre del punto
 */
public void setUp(int port, String ipAdrees, String name) {
  ip = ipAdrees;
  port = port;
  name = name;
  //ownPeer.setDefaultProtocolType("org.jrra.network.spi.UPD");
  setUp(ip, port);
  setName(name);
  JRRA jrra = JRRA.init();
  this.controller = jrra.createController(ownPeer);
  registerListeners();
```

```
    this.controller.initReceiver();
  }

  /**
   *  Setea la dirección IP, el purto del punto
   */
  public void setUp(String inetAddress, int port) {
    String protocol = "org.jrra.network.spi.UDP";
    Address address = new Address();
    this.ownPeer = new Peer(utilities.Util.id);
    address.setAttribute("inetAddress", inetAddress);
    address.setAttribute("port", port);
    ownPeer.setAddress(protocol, address);
    ownPeer.setDefaultProtocolType(protocol);
  }

/**
   * Registra los listeners para que el controlador informe de la llegada de un paquete.
   */
  private void registerListeners() {
    controller.addListener(org.jrra.packets.privilegedpeer.
                        RegisterPeerRequestPacket.class.getName(), this);
    //controller.addListener(org.jrra.packets.general.MessagePacket.class.getName(), this);
    controller.addListener(org.jrra.packets.privilegedpeer.PeerListPacket.class.
                        getName(), this);
    controller.addListener(P2P.packets.ArrPacket.class.getName(), this);
    controller.addListener(P2P.packets.FileP2PPacket.class.getName(), this);
    controller.addListener(P2P.packets.MessagePacket.class.getName(), this);
    controller.addListener(org.jrra.packets.SegmentedPacket.class.getName(), this);
    controller.addListener(P2P.packets.RecivedMessage.class.getName(), this);
    controller.addListener(P2P.packets.HeaderSegmentedPacket.class.getName(), this);
    controller.addListener(org.jrra.packets.privilegedpeer.PeerListQueryPacket.class.
                        getName(), this);
  }

  /**
   * Evento que se dispara al recibir un paquete
   */
  public void update(EventMessage event) {
    System.out.println("Recibí algo...");
    PacketData data = event.getPacketData();
    System.out.println("data " + data);
    if (data instanceof PeerListPacket) {
      PeerListPacket plp = (PeerListPacket) data;
      PeerList pl = plp.getPeer();
      Peer temp = pl.lookup(ownPeer.getUin());
      ownPeer.setP2PId(temp.getP2PId());
      notifyAllRefreshList(plp, event.getPeer());
    }
    else if (data instanceof ArrPacket) {
      ArrPacket arr = (ArrPacket) data;
      Document d = utilities.Util.parseXmlDocument(arr.getByteArray());
      FileSystem fs = new FileSystem("foreigns");
      fs.addFile("<?xml version=\"1.0\"?>" + utilities.Util.printDom(d));
    }
```

```
  else if (data instanceof MessagePacket) {
    String t = ( (MessagePacket) data).getMessage();
    chatViewForm.setText(t);
    fireRequestChangeFormEvent(chatViewForm);
  }
  else if (data instanceof FileP2PPacket) {
    FileP2PPacket dp = (FileP2PPacket) data;
    xmlAddDocument.start();
    xmlAddDocument.setMeta(dp);
    fireRequestChangeFormEvent(xmlAddDocument);
  }
  else if (data instanceof SegmentedPacket) {
    SegmentedPacket sp = (SegmentedPacket) data;
  }
  else if (data instanceof PeerListQueryPacket) {
    PeerListQueryPacket qlp = (PeerListQueryPacket) data;
    PeerList pl = qlp.getPeer();
    foundedPeersForm.start(pl);
    fireRequestChangeFormEvent(foundedPeersForm);
  }
}



/**
 *  Notifica a todos los observadores que deben refrescar la lista de puntos conectados
 */
public void notifyAllRefreshList(PacketData d, Peer p) {
  Enumeration enume = listeners.keys();
  for (; enume.hasMoreElements(); ) {
    String key = (String) enume.nextElement();
    PacketListener list = (PacketListener) listeners.get(key);
    if (list instanceof RefreshPeerListListener) {
      list.update(new EventMessage(d, p));
    }
  }
}

public void setPeerList(PeerList pl) {
  this.peerList = pl;
}

/**
 * Envía un paquete por partes al punto pasado como parámetro.
 */
public void sendNewMessageByParts(Peer peer, PacketData packet) {
  Vector v = ( (Packet) packet).getSegments();
  for (int x = 0; x < v.size(); x++) {
    sendNewMessage(peer,
                new SegmentedPacket( (byte[]) v.elementAt(x), v.size(),
                                    x + 1, utilities.Util.name));
    try {
      Thread.sleep(2000);
    }
    catch (Exception e) {}
```

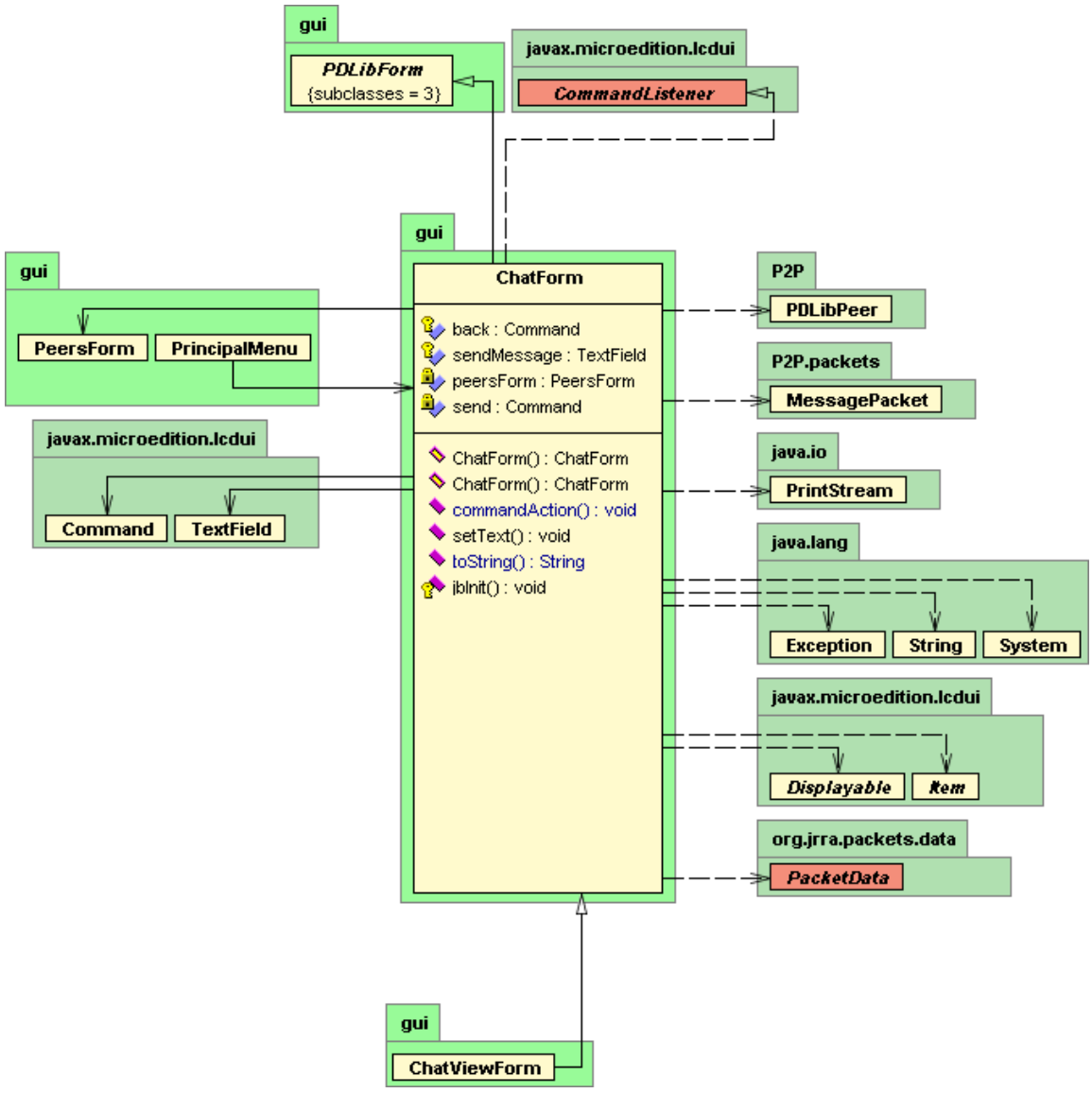
```
  }
}

/**
 * Envía un paquete al punto pasado como parámetro.
 */
public void sendNewMessage(Peer peer, PacketData packet) {
  if (packet instanceof Packet) {
    sendNewMessageByParts(peer, packet);
  }
  else {
    this.controller.sendMessage(packet, peer);
  }
}

public void setName(String name) {
  ownPeer.setAttribute("name", name);
}

/**
 * Obtiene el nombre del punto
 */
public String getName() {
  return (String) ownPeer.getAttribute("name");
}

/**
 * Agrega un obervador o listener.
 */
public void addListener(String packetName, PacketListener listener) {
  listeners.put(packetName, listener);
}

public void setOwnPeer(PDLibPeer este) {
  this.este = este;
}

/**
 * Obtiene la instancia al punto (A este punto)
 */
public PDLibPeer getOwnPeer() {
  return this;
}

public Peer getPeer() {
  return ownPeer;
}

public void addRequestChangeFormListener(RequestChangeForm l) {
  this.vlisteners.addElement(l);
}

public void fireRequestChangeFormEvent(Displayable f) {
  for (int x = 0; x < listeners.size(); x++) {
    ( (RequestChangeForm) vlisteners.elementAt(x)).requestChangeFormEvent(f);
```

```
  }
 }
}
```

ChatForm.java

Clase GUI para interfaz de la forma de chateo (tanto para envío como para recibo). Esta clase desplazó a ChatViewForm.

Diagrama



Código

public class ChatForm extends PDLibForm

implements CommandListener {
protected Command back = new Command("Atras", Command.BACK, 1);

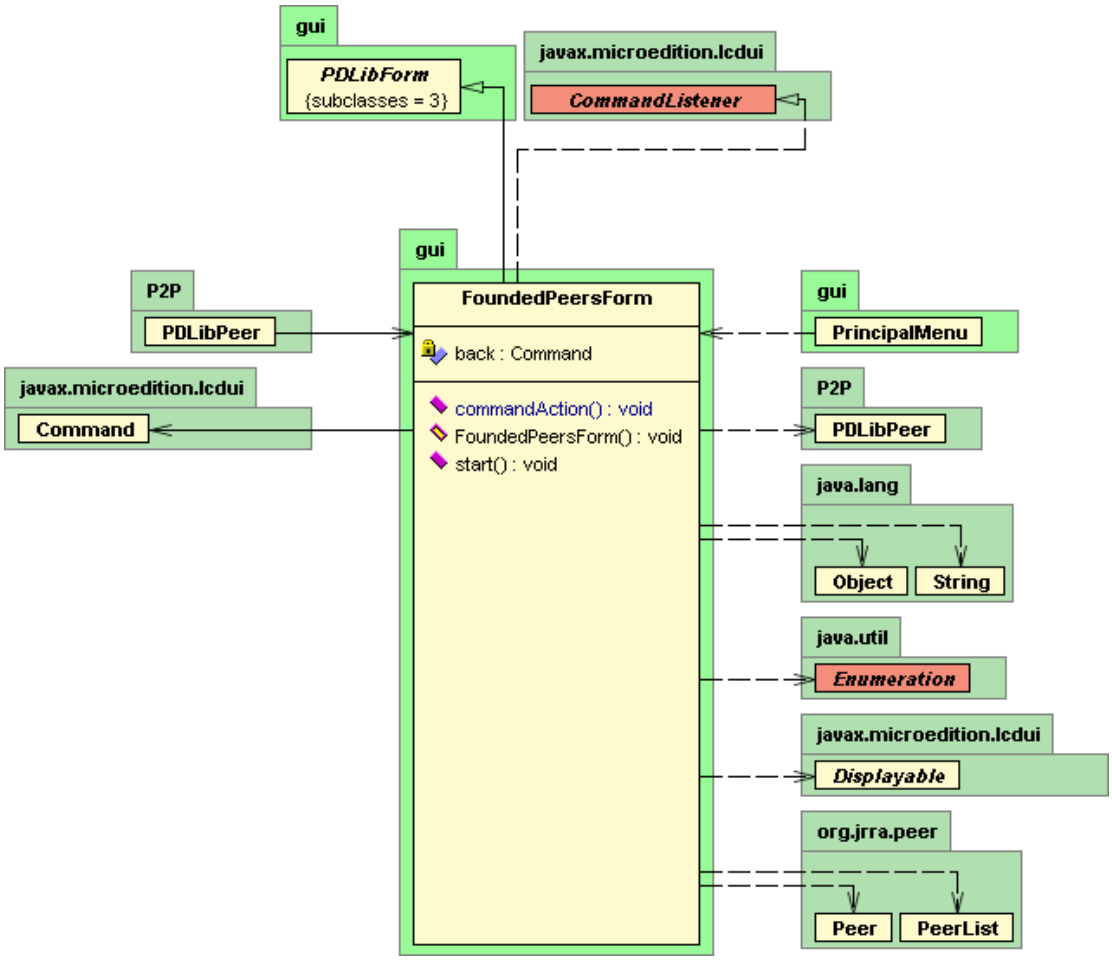
```
private Command send = new Command("Enviar", Command.OK, 2);
protected TextField sendMessage = new TextField("Message: ", "Hola PDLib!!!", 100, TextField.ANY);
private PeersForm peersForm;

    /**
     * Crea una nueva instancia de ChatForm
     * @param title: Titulo de de la forma
     * @param ownPeer: Instancia de PDLibPeer
     * @param peersForm: Referencia a la isntancia de PeersForm
     * @param parent: Referencia al padre, para la opción back
     */
    public ChatForm(String title, PDLibPeer ownPeer, PeersForm peersForm, Displayable parent) {
        super(title, ownPeer, parent);
        this.peersForm = peersForm;
        try {
            jbInit();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Crea una nueva instancia de ChatForm con el titulo pasado como parametro.
     */
    public ChatForm(String title) {
        super(title);
        try {
            jbInit();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Setea el mensaje a enviar
     */
    public void setText(String txt) {
        this.sendMessage.setString(txt);
    }

    public String toString() {
        return "Chat";
    }

    /**
     * Método que inicia la interfaz gráfica
     */
    protected void jbInit() throws Exception {
        // Set up this Displayable to listen to command events
        setCommandListener(this);
        this.addCommand(back);
        this.addCommand(send);
        append(sendMessage);
    }
```

```
    /**
     * Método que se dispara al seleccionar una opción
     */
    public void commandAction(Command command, Displayable displayable) {
        /** @todo Add command handling code */
        if (command == back) {
            System.out.println("Chat Form back");
            super.fireRequestChangeFormEvent(null);
        } else if (command == send) {
            MessagePacket data = new MessagePacket(sendMessage.getString());
            peersForm.setPacketData(data);
            super.fireRequestChangeFormEvent(peersForm);
        }
    }
}
```

FoundedPeersForm.java

Clase GUI para la interfaz gráfica de los puntos encontrados en una búsqueda al PP.

Diagrama



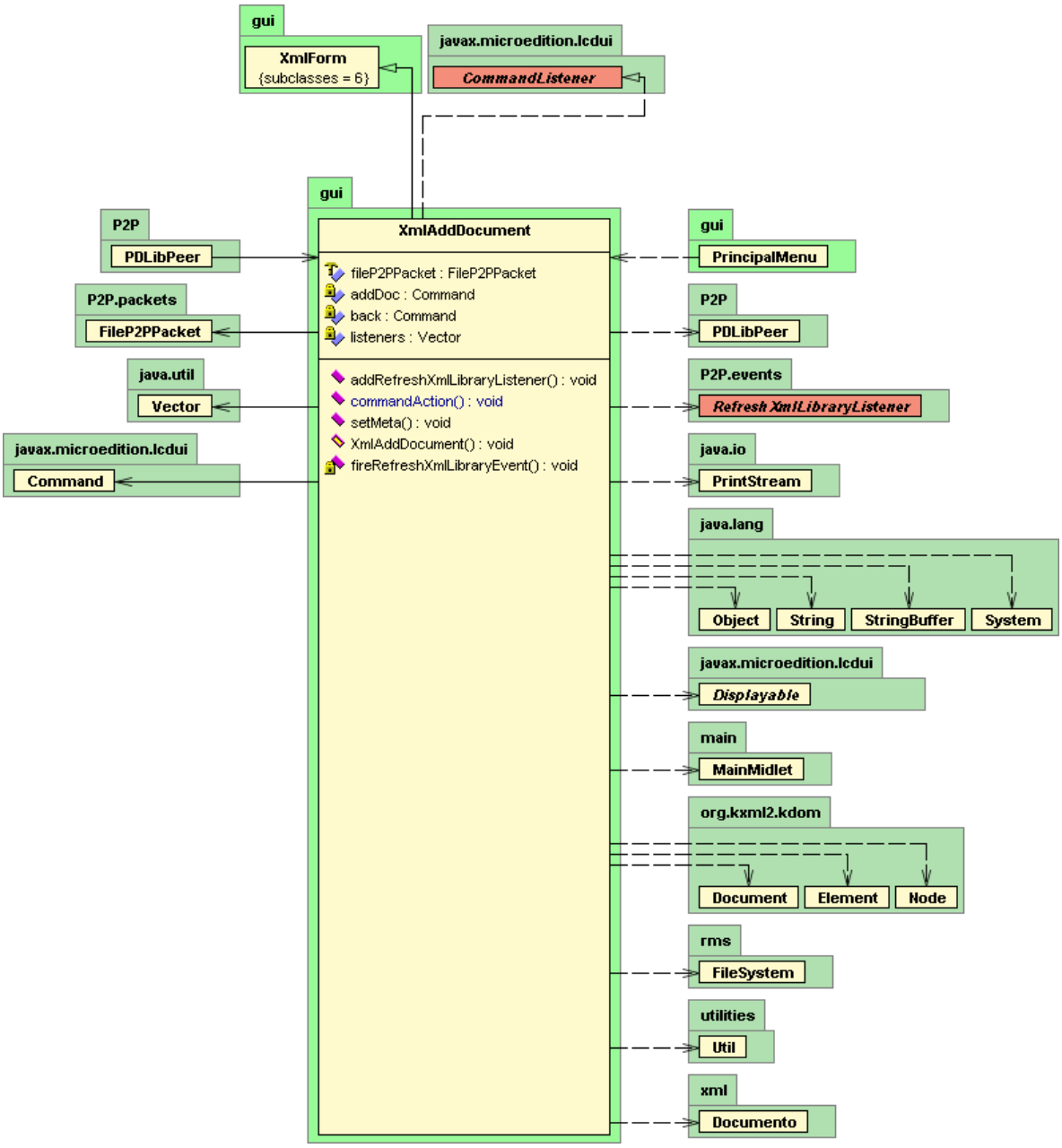
Código

```
public class FoundedPeersForm extends PDLibForm
      implements CommandListener {

   private Command back = new Command("Atras", Command.BACK, 1);

   /** Crea una nueva instancia de FoundedPeersForm
    * @param title: Titulo de de la forma
    * @param ownPeer: Instancia de PDLibPeer
    * @param parent: Referencia al padre, para la opción back
    */
   public FoundedPeersForm(String title, PDLibPeer ownPeer, Displayable parent) {
      super(title, ownPeer, parent);
      this.addCommand(back);
      setCommandListener(this);
   }

   /**
    * Método que inicia la interfaz gráfica (pone los nombres de los puntos encontrados)
    *
    */
   public void start(PeerList list) {
      Enumeration enume = list.elements();
      super.deleteAll();
      while (enume.hasMoreElements()) {
         Peer p = (Peer) enume.nextElement();
         super.append(p.getStringAttribute("name") + "\n");
      }
   }

    /**
    * Método que se dispara al seleccionar una opción
    */
   public void commandAction(Command command, Displayable displayable) {
      if (command == back) {
         super.fireRequestChangeFormEvent(super.getParent());
      }
   }
}
```

XmlAddDocument.java

Clase GUI para meter un documento recibido en alguna colección. Presenta la GUI para seleccionar la colección donde irá el nuevo documento. Se usa al momento de recibir un documento a través de P2P.

Diagrama



Código

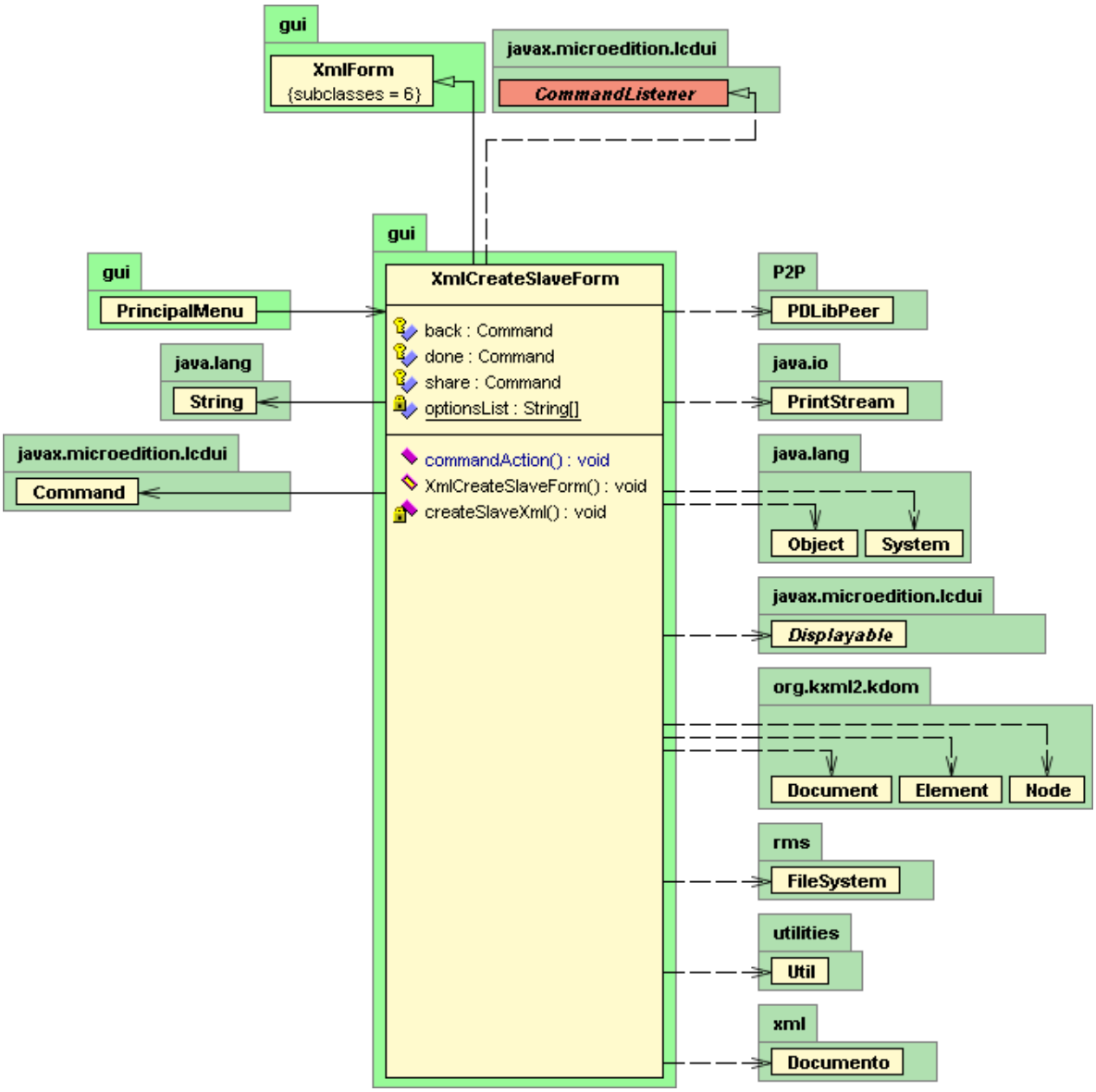
```
public class XmlAddDocument extends XmlForm implements CommandListener {
   private Command back = new Command("Atras", Command.BACK, 1);
   private Command addDoc = new Command("Agregar documento aquí", Command.ITEM, 2);
   private Vector listeners = new Vector();
   FileP2PPacket fileP2PPacket;

   /** Crea una nueva instancia de QueryForm
    * @param title: Titulo de de la forma
    * @param ownPeer: Instancia de PDLibPeer
    * @param parent: Referencia al padre, para la opción back
    */
   public XmlAddDocument(String title, int type, Displayable parent, PDLibPeer ownPeer) {
      super(title, type, ownPeer, parent);
      this.setCommandListener(this);
      this.addCommand(back);
      this.addCommand(addDoc);
   }

   /**
    * Método que agraga observadores que requieren saber cuando hacer un refresh de la biblioteca
    */
   public void addRefreshXmlLibraryListener(RefreshXmlLibraryListener l) {
      this.listeners.addElement(l);
   }

   /**
    * Setea el paquete recibido de tipo FileP2PPacket
    */
   public void setMeta(FileP2PPacket f) {
      fileP2PPacket = f;
   }

   /**
    * Dispara el método update de los observados que implementan RefreshXmlLibraryListener
    */
   private void fireRefreshXmlLibraryEvent() {
      for (int x = 0; x < listeners.size(); x++)
         ((RefreshXmlLibraryListener) listeners.elementAt(x)).refresh();
   }

   /**
    * Método que se dispara al seleccionar una opción
    */
   public void commandAction(Command command, Displayable displayable) {
      if (command == back) {
         super.fireRequestChangeFormEvent(super.getParent());
      }
      else if (command == addDoc) {
         Documento doc = super.getSelectedElement();
         Element meta = utilities.Util.genXmlTransferredFile(fileP2PPacket.getMetas()).getRootElement();

         doc.getElement().addChild(Node.ELEMENT, meta);
         Node root = doc.getElement().getRoot();
```

```
        Element inFile = utilities.Util.getInFileCollection(root);

        inFile.addChild(Node.ELEMENT, meta);

        String xml = "<?xml version=\"1.0\"  encoding=\"iso-8859-1\"?>" + Util.printDom(root);
        FileSystem fs = new FileSystem("PDLib");
        fs.updateRecord(main.MainMidlet.idMasterRecord, xml.getBytes());
        fs.closeFileSystem();

        byte[] file = fileP2PPacket.getFile();
        System.out.println("docto " + new String(file));
        //Si el file es null no hay documento
        if (file != null) {
           FileSystem fsdocto = new FileSystem("doctos");

           fsdocto.add(file);
           int num = fsdocto.getNumRecords();
           for (int x = 0; x < num; x++)
              System.out.println("docto " + x + " " + new String(fsdocto.getRecord(x)));
           fsdocto.closeFileSystem();
        }

        fireRefreshXmlLibraryEvent();
     }
     else
        getElements(getSelectedElement(), "document");
  }

}
```

XmlCreateSlaveForm.java

Clase que presenta la GUI para crear vistas parciales. Provee métodos para la creación de los XML'S parciales para las vistas parciales.

Diagrama



Código

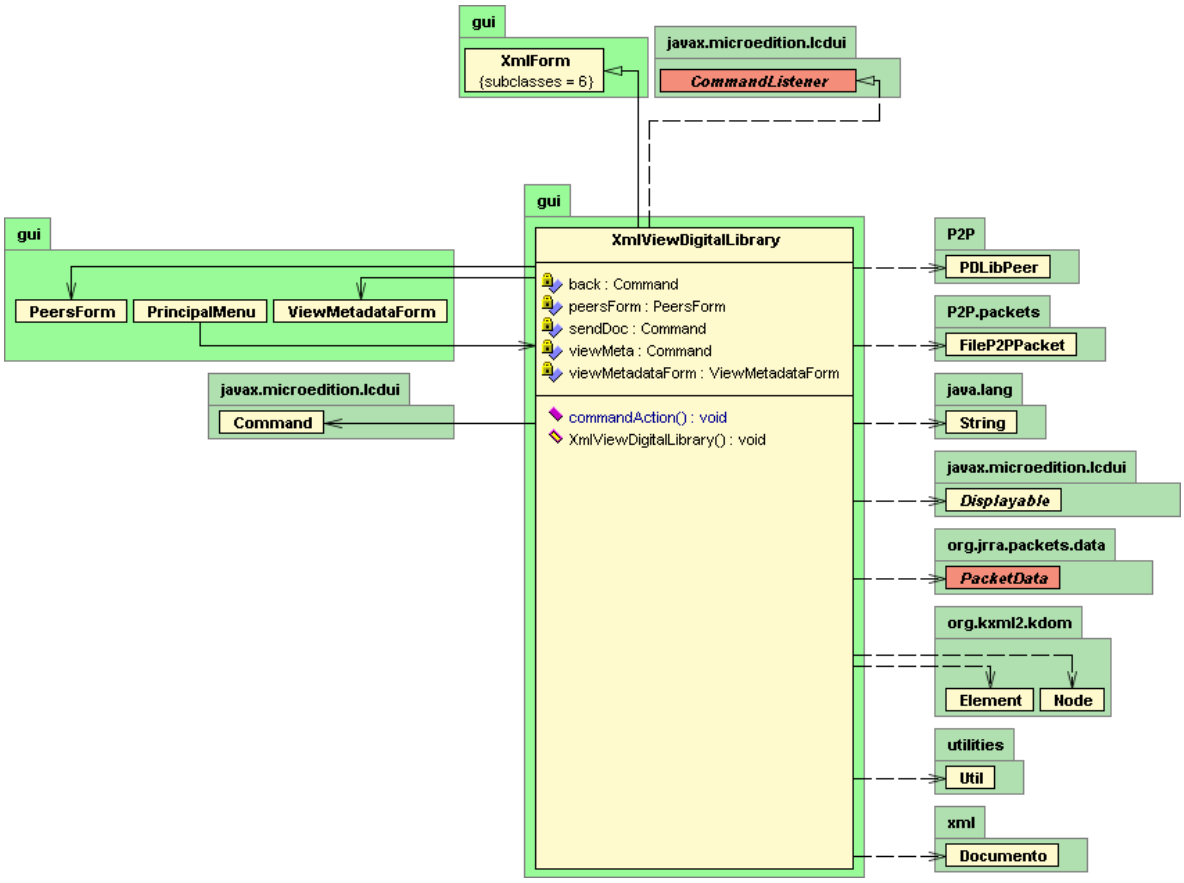
```
public class XmlCreateSlaveForm extends XmlForm implements CommandListener {
    protected Command share = new Command("Compartir", Command.ITEM, 2);
    protected Command done = new Command("Terminar", Command.ITEM, 5);
    protected Command back = new Command("Atras", Command.BACK, 1);
    private final static String[] optionsList = { "Compartir", "No compartir"};
```

```
/** Crea una nueva instancia de QueryForm
 * @param title: Titulo de de la forma
 * @param type: El tipo de la lista (XmlForm hereda de PDLibList)
 * @param ownPeer: Instancia de PDLibPeer
 * @param parent: Referencia al padre, para la opción back
 */
public XmlCreateSlaveForm(String title, int type, PDLibPeer ownPeer, Displayable parent) {
   super(title, type, ownPeer, parent);
   setCommandListener(this);
   this.addCommand(share);
   this.addCommand(done);
   this.addCommand(back);
}

/**
 * Le cuelga un hijo a rootDest con la colección root pasada como parametro
 */
private void createSlaveXml(Element root, Element rootDest) {
   int count = root.getChildCount();
   for (int x = 0; x < count; x++) {
      Element child = root.getElement(x);
      if (child != null && child.isShare()) {
         Element newChild = child.cloneNode(true);
         rootDest.addChild(Node.ELEMENT, newChild);
      }
   }
}

/**
 * Método que se dispara al seleccionar una opción
 */
public void commandAction(Command command, Displayable displayable) {
   if (command == back) {
      super.fireRequestChangeFormEvent(super.getParent());
   }
   else if (command == share) {
      Documento doc = getSelectedElement();
      doc.getElement().setIsShare(true);
   }
   else if (command == done) {
      Documento doc = getSelectedElement();
      Element root = doc.getElement().getRoot().getElement(1);
      Element rootDest = root.cloneNode(false);
      createSlaveXml(root, rootDest);
      Document slave = Util.parseXmlDocument(Util.getXmlViewSlave());
      slave.addChild(Node.ELEMENT, rootDest);
      FileSystem fs = new FileSystem("slaves");
      System.out.println("<?xml version=\"1.0\"?> " + Util.printDom(slave));
      fs.addFile("<?xml version=\"1.0\"?>" + Util.printDom(slave));
      //fs.closeFileSystem();
   }
   else {
      getElements(getSelectedElement());
   }
}   }
```

XmlViewDigitalLibrary.java

Clase para navegar a través de la biblioteca digital local

Diagrama



Código

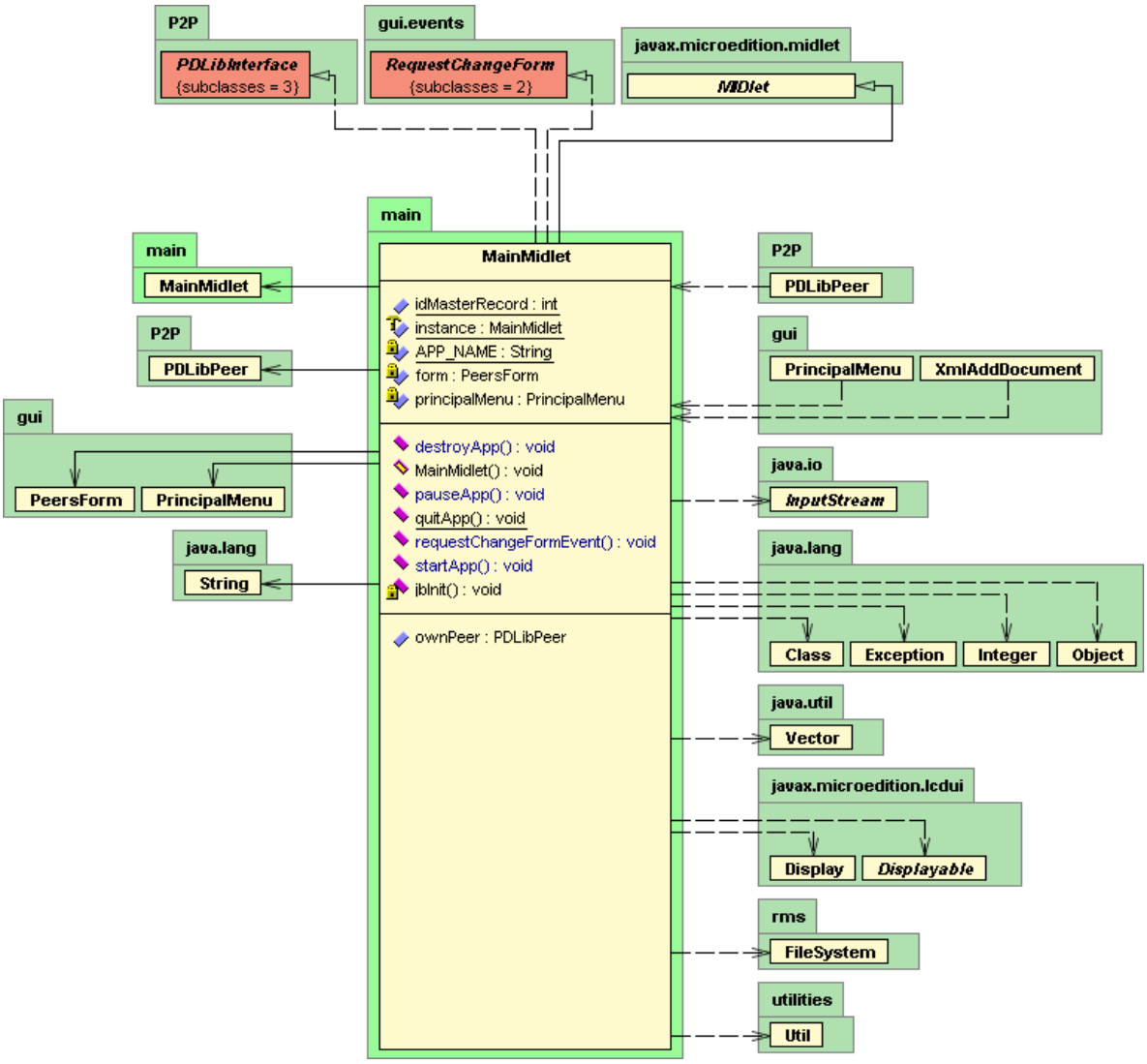
```
public final class XmlViewDigitalLibrary extends XmlForm implements CommandListener {
    private Command back = new Command("Atras", Command.BACK, 1);
    private Command viewMeta = new Command("Ver metadatos", Command.ITEM, 2);
    private Command sendDoc = new Command("Enviar documento", Command.ITEM, 3);
    private ViewMetadataForm viewMetadataForm;
    private PeersForm peersForm;

    /** Crea una nueva instancia de XmlViewDigitalLibrary
     * @param title: Titulo de de la forma
     * @param type: El tipo de la lista (XmlForm hereda de PDLibList)
     * @param ownPeer: Instancia de PDLibPeer
     * @param parent: Referencia al padre, para la opción back
     * @param viewMetadataForm: Referencia a la clase ViewMetadataForm, donde se muestra los metadatos
de un documento
     * @param peersForm: Referencia a la clase PeersForm, para enviar una vista esclava.
     */
    public XmlViewDigitalLibrary(String title, int type, PDLibPeer ownPeer, Displayable parent,
ViewMetadataForm viewMetadataForm, PeersForm peersForm) {
        super(title, type, ownPeer, parent);
```

```
        this.viewMetadataForm = viewMetadataForm;
        this.peersForm = peersForm;
        this.setCommandListener(this);
        this.addCommand(back);
        this.addCommand(viewMeta);
        this.addCommand(sendDoc);
    }

    /**
     * Método que se dispara al seleccionar una opción
     */
    public void commandAction(Command command, Displayable displayable) {
        if (command == back) {
            super.fireRequestChangeFormEvent(super.getParent());
        } else if (command == sendDoc) {
            byte[] arr = Util.getDocument();
            Documento doc = getSelectedElement();
            FileP2PPacket data = new FileP2PPacket(arr, Util.printDom(doc.getElement()));
            peersForm.setPacketData(data);
            super.fireRequestChangeFormEvent(peersForm);
        } else if (command == viewMeta) {
            viewMetadataForm.createInterface(getSelectedElement());
            super.fireRequestChangeFormEvent(viewMetadataForm);
        } else
            getElements(getSelectedElement());
    }
}
```

MainMidlet.java

Clase Main principal. Esta clase es el Midlet principal que se carga al inicializar el prototipo.

Diagrama



Código

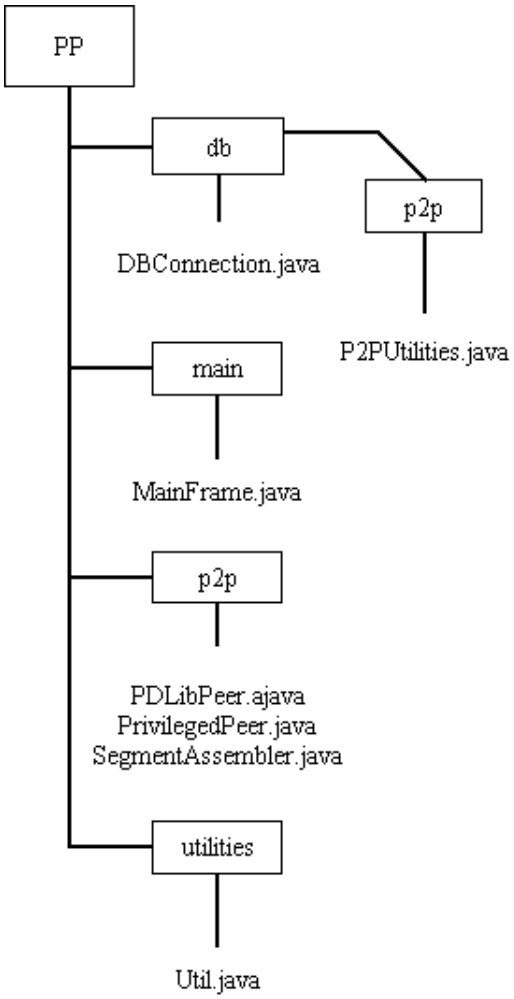
```
public class MainMidlet extends MIDlet implements PDLibInterface, RequestChangeForm {
  static MainMidlet instance;
  private PeersForm form;
  private PrincipalMenu principalMenu;
  public static int idMasterRecord = 1;
  private static String APP_NAME = "P2PDlib";

  //private Peer ownPeer;
  private PDLibPeer ownPeer;
```

```
/**
 * Crea una nueva instancia de MainMidlet
 */
public MainMidlet() {
  instance = this;
  utilities.Util.id = Integer.parseInt(this.getAppProperty("p2p-id"));
  utilities.Util.port = Integer.parseInt(this.getAppProperty("p2p-port"));
  utilities.Util.name = this.getAppProperty("p2p-name");

  try {
    jbInit();
  }
  catch(Exception e) {
    e.printStackTrace();
  }
}

/**
 * Método que crea una nueva instancia de PDLibPeer, y arranca el punto
 */
public void startApp() {
  ownPeer = new PDLibPeer();
  principalMenu = new PrincipalMenu(ownPeer);
  principalMenu.addRequestChangeFormListener(this);
  Display.getDisplay(this).setCurrent(principalMenu);
}


public void pauseApp() {
}

public void destroyApp(boolean unconditional) {
}

public static void quitApp() {
  instance.destroyApp(true);
  instance.notifyDestroyed();
  instance = null;
}

/**
 * Método que lee del RMS el XML de la biblioteca digital.
 */
private void jbInit() throws Exception {
  FileSystem fs = new FileSystem("PDLib");
  java.io.InputStream in = this.getClass().getResourceAsStream(utilities.Util.resfile_name);
  if (fs.getNumRecords() == 0) {
     fs.add(utilities.Util.getXmlBytes(in));
  }
  else {
     java.util.Vector v = fs.getAllRecordsIds();
     int id = ((Integer)v.elementAt(0)).intValue();
     idMasterRecord = id;
     fs.updateRecord(id, utilities.Util.getXmlBytes(in));
  }
```

```
    //fs.deleteAllRecordsIds();

    //fs.closeFileSystem();
    //fs.add(utilities.Util.getXmlBytes());
    //fs.updateRecord(2, utilities.Util.getXmlBytes());
    //fs.add(utilities.Util.getXmlBytes());

  }

  /**
   * Setea la instancia PDLibPeer del punto
   */
  public void setOwnPeer(PDLibPeer ownPeer) {
   this.ownPeer = ownPeer;
  }


  /**
   * Obtiene la instancia PDLibPeer del punto
   */
  public PDLibPeer getOwnPeer() {
    return ownPeer;
  }

  /**
   * Método llamado por las clases que requieren hacer cambio de ventana. Ver RequestChangeForm.
   */
  public void requestChangeFormEvent(Displayable f) {
    Display.getDisplay(this).setCurrent(f);
  }
}
```

Punto privilegiado

El código del punto privilegiado es mucho más simple que el del cliente. El PP esta organizado en 4 paquetes o carpetas: db, main, p2p y utilities, en el paquete db están contenidas las clases que sirven para la conexión a la base de datos, en el paquete main esta la clase Main principal, en el paquete p2p están contenidas las clases que sirven para proveer a la aplicación de funcionalidad para conectarse a la red punto a punto y enviar y recibir paquetes a los puntos conectados, en el paquete utilities se encuentran clases de uso común para la aplicación. Debido a que el PP no ofrece mucha funcionalidad su código como ya se mencionó no es muy extenso ni complicado, la parte de comunicación P2P y de base de datos viene siendo la médula espinal del mismo (paquetes db y p2p). A continuación se presenta un diagrama con la estructura.

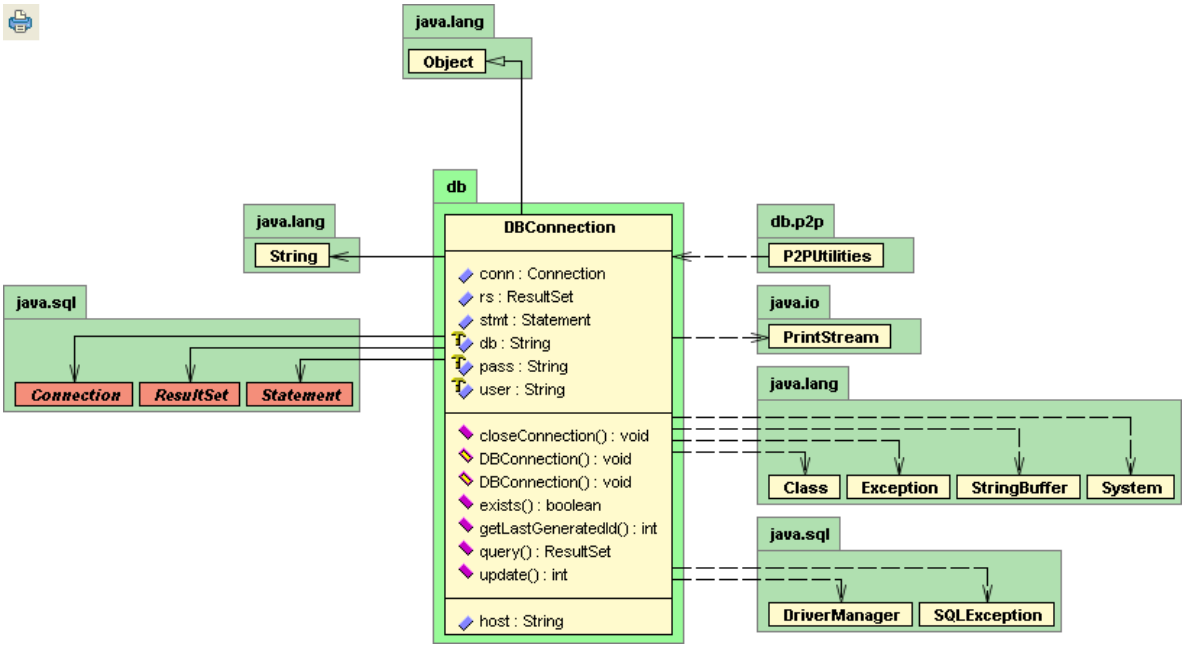


A continuación se presentan los diagramas UML y códigos más importantes del punto privilegiado, es decir la parte que debe ser instalada en una máquina con un poder de cómputo mayor al de un dispositivo móvil, por ejemplo una computadora de escritorio.

DBConnection.java

Clase utilizada para crear una conexión a la base de datos. Provee métodos para realizar la conexión, hacer un query, cerrar la conexión, entre otros.

Diagrama



Código

```
public class DBConnection {

   public Connection conn;
   public Statement stmt;
   public ResultSet rs;
   String host = System.getProperty("dbHost"),
          db = System.getProperty("dbName"),
          user = System.getProperty("dbUser"),
          pass = System.getProperty("dbPass");

   /**
    * Crea una nueva instancia de DBConnection.
    * @param dbHost: El nombre del host de la base de datos.
    * @param dbName: El nombre de la base de datos.
    * @param dbUser: El nombre de usuario de la base de datos.
    * @param dbPass: El password de la base de datos.
    */
   public DBConnection(String dbHost, String dbName, String dbUser, String dbPass) {
      host = dbHost;
```

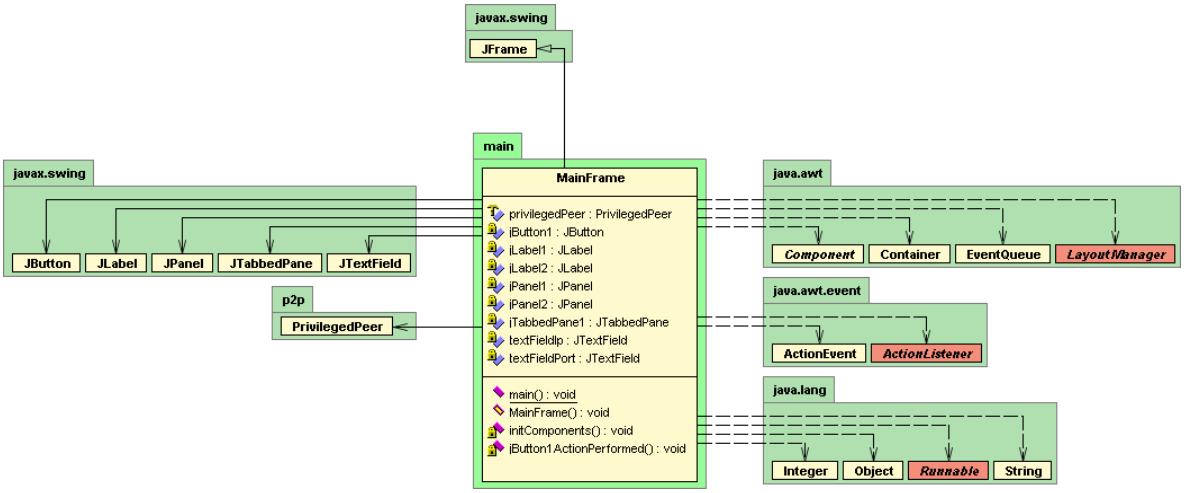
```
        db = dbName;
        user = dbUser;
        pass = dbPass;
        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            conn = DriverManager.getConnection("jdbc:mysql://"+host+"/"+db, user, pass);

            stmt = conn.createStatement();
            this.host = host;
            this.db = db;
            this.user = user;
            this.pass = pass;
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

/**
* Crea una nueva instancia de DBConnection. Tomando a el nombre del host, nombre de la base de datos, etc.
* desde parámetros de la máquina virtual.
*/
public DBConnection() {
        this(System.getProperty("dbHost"), System.getProperty("dbName"), System.getProperty("dbUser"),
System.getProperty("dbPass"));
    }

    /**
     * Obtiene el último id generado (Autoincrement)
     */
    public int getLastGeneratedId() {
        String q = "SELECT LAST_INSERT_ID()";
        int id = 0;
        ResultSet rs = query(q);
        try {
            if (rs.next()) id =  rs.getInt(1);
            rs.close();
            rs = null;
        } catch(Exception e) {
            e.printStackTrace();
        } finally {
            if (rs != null) {
                try {
                    rs.close();
                    rs = null;
                }catch(Exception e) {}
            }
        }
        return id;
    }

 /**
     * Regresa verdadero si existe al menos un registro del query pasado como parámetro.
     */
    public boolean exists(String q) {
        ResultSet rs = query(q);
```

```
    boolean exists = false;
    try {
        if (rs.next())
            exists = true;
        else
            exists = false;
        rs.close();
        rs = null;
    }catch (Exception e) {
        e.printStackTrace();
    } finally {
        if (rs != null) {
            try {
                rs.close();
                rs = null;
            }catch(Exception e) {}
        }
    }
    return exists;
}

/**
 * Obtiene un cursor devuelto por la base de datos al ejecutar el query pasado como parámetro.
 */
public ResultSet query(String q) {
    try {
        System.out.println(q);
        rs = stmt.executeQuery(q);
        return  rs;
    } catch(Exception e) {
        e.printStackTrace();
        return null;
    }
}

/**
 * Obtiene el nombre de host de la base de datos.
 */
public String getHost() {
    return host;
}

/**
 * Cierra la conexión actual.
 */
public void closeConnection() {
    try {
        this.stmt.close();
        this.conn.close();
    }catch(Exception e) {
        e.printStackTrace();
    } finally {
        if (stmt != null) {
            try { stmt.close();  } catch (SQLException e) { ; }
            stmt = null;
```

```
            }
            if (conn != null) {
                try { conn.close();  } catch (SQLException e) { ; }
                conn = null;
            }
        }
    }

    /**
     * Ejecuta el query de tipo update pasado como parámetro.
     */
    public int update(String q) {
        try {
            System.out.println(q);
            return stmt.executeUpdate(q);
        } catch(Exception e) {
            //Utilities.Util.errorMessage("Hubo error en DBConnection!!!");
            e.printStackTrace();
            return 0;
        }
    }
}
```

MainFrame.java

Clase Main principal para crear al punto privilegiado. Provee la interfaz gráfica.



Código

```
public class MainFrame extends javax.swing.JFrame {
    PrivilegedPeer privilegedPeer;

    /** Crea una nueva instancia de MainFrame */
    public MainFrame() {
        initComponents();
        privilegedPeer = new PrivilegedPeer();
```

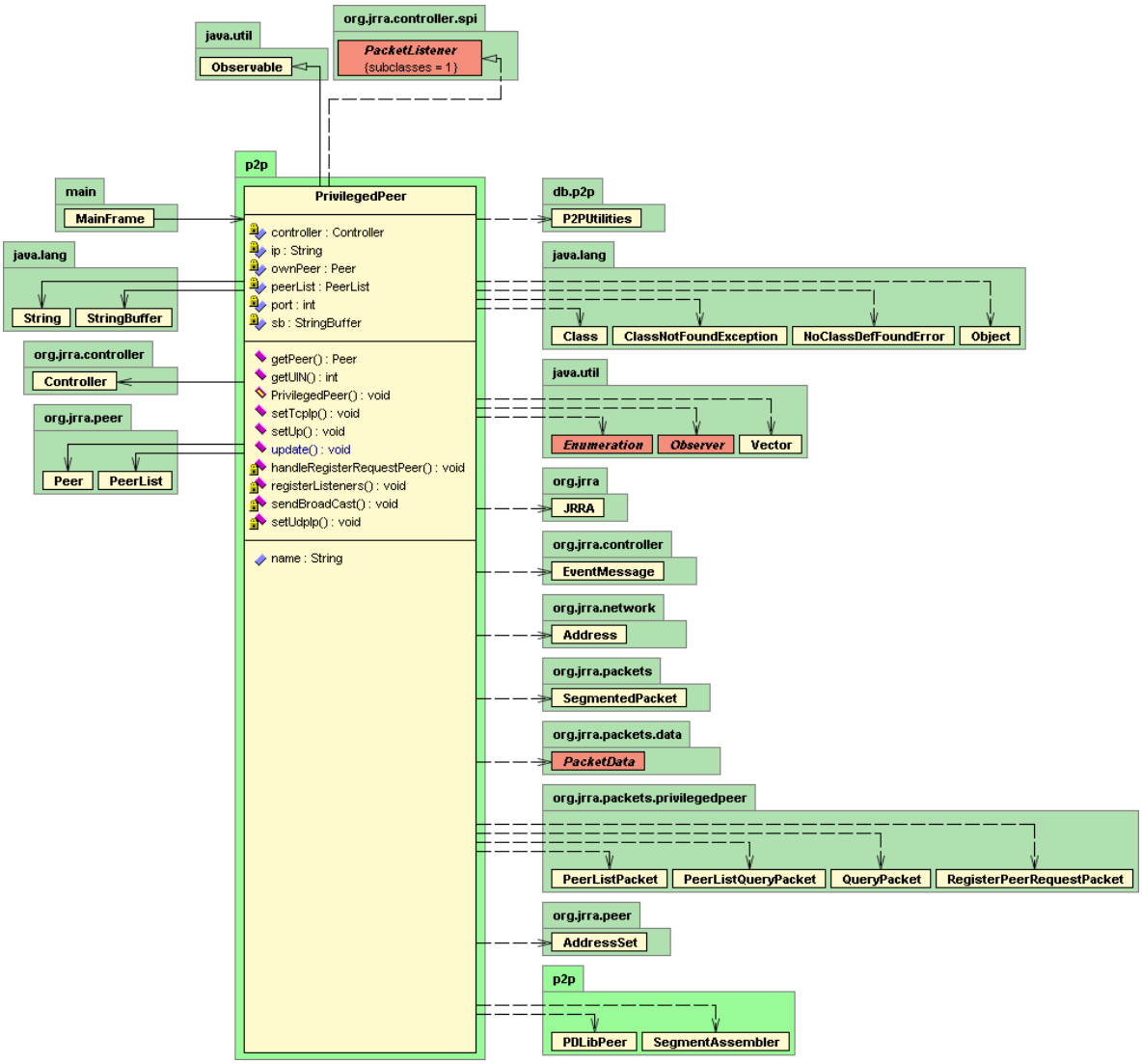
```
    this.setSize(350, 450);
}

/*/
 * Este método será llamado por el constructor al inicializar la forma.
 */
private void initComponents() {
    jTabbedPane1 = new javax.swing.JTabbedPane();
    jPanel1 = new javax.swing.JPanel();
    jLabel1 = new javax.swing.JLabel();
    textFieldIp = new javax.swing.JTextField();
    jLabel2 = new javax.swing.JLabel();
    textFieldPort = new javax.swing.JTextField();
    jPanel2 = new javax.swing.JPanel();
    jButton1 = new javax.swing.JButton();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
    jPanel1.setLayout(null);

    jLabel1.setText("Direcci\u00f3n IP:");
    jPanel1.add(jLabel1);
    jLabel1.setBounds(10, 10, 80, 14);

    textFieldIp.setText("localhost");
    jPanel1.add(textFieldIp);
    textFieldIp.setBounds(100, 10, 110, 19);

    jLabel2.setText("Puerto: ");
    jPanel1.add(jLabel2);
    jLabel2.setBounds(10, 50, 50, 14);

    textFieldPort.setText("8000");
    jPanel1.add(textFieldPort);
    textFieldPort.setBounds(100, 50, 70, 19);

    jTabbedPane1.addTab("Configuraci\u00f3n", jPanel1);

    getContentPane().add(jTabbedPane1, java.awt.BorderLayout.CENTER);

    jButton1.setText("Iniciar Servidor");
    jButton1.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButton1ActionPerformed(evt);
        }
    });

    jPanel2.add(jButton1);

    getContentPane().add(jPanel2, java.awt.BorderLayout.SOUTH);

    pack();
}
```

```
    /**
     * Método auxiliar para el evento de arrancar el servidor.
     */
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        int port = Integer.parseInt(this.textFieldPort.getText());
        this.privilegedPeer.setUp(port, this.textFieldIp.getText(), "Punto Privilegiado");
    }

    /**
     * Método Main
     */
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new MainFrame().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JTabbedPane jTabbedPane1;
    private javax.swing.JTextField textFieldIp;
    private javax.swing.JTextField textFieldPort;
    // End of variables declaration

}
```

PrivilegedPeer.java

Clase que representa al punto privilegiado. Provee métodos para enviar y recibir mensajes de los puntos.

Diagrama



Código

```
public class PrivilegedPeer extends Observable implements PacketListener {
   private Controller controller;
   private Peer ownPeer;
   private String ip, name;
   private PeerList peerList = new PeerList();
   private int port;
   private StringBuffer sb = null;
```

```
/** Crea una nueva instancia de PrivilegedPeer */
public PrivilegedPeer() {

}

/**
 * Registra a todos las clases de paquetes que deben ser tomadas en cuenta por el controlador al llegar
 */
private void registerListeners() {
    controller.addListener(org.jrra.packets.privilegedpeer.RegisterPeerRequestPacket.class.getName(), this);
    controller.addListener(org.jrra.packets.general.MessagePacket.class.getName(), this);
    controller.addListener(org.jrra.packets.SegmentedPacket.class.getName(), this);
    controller.addListener(org.jrra.packets.privilegedpeer.QueryPacket.class.getName(), this);
}

/**
 * Inicializa al punto privilegiado con la dirección ip, el puerto y el nombre pasados como parámetros.
 */
public void setUp(int port, String ipAdrees, String name) {
    ip = ipAdrees;
    port = port;
    name = name;
    setUdpIp(ip, port);
    setName(name);
    JRRA jrra = JRRA.init();
    this.controller = jrra.createController(ownPeer);
    registerListeners();
    this.controller.initReceiver();
}

/**
 * Inicializa al punto privilegiado haciendo uso de sockets tcp
 */
public void setTcpIp(String tcpIp, int port) {
    String protocol = "org.jrra.network.spi.TCP";
    this.ownPeer = new Peer(0);
    Address address = ownPeer.getAddressSet().lookup(protocol);
    if(address != null) {
        address.setAttribute("inetAddress", tcpIp);
        address.setAttribute("port", port);
    } else {
        address = new Address();
        address.setAttribute("inetAddress", tcpIp);
        address.setAttribute("port", port);
        ownPeer.getAddressSet().bind(address, "org.jrra.network.spi.TCP");
    }
}

/**
 * Inicializa al punto privilegiado haciendo uso de sockets udp
 */
private void setUdpIp(String udpIp, int port) {
    String protocol = "org.jrra.network.spi.UDP";
    this.ownPeer = new Peer(0);
    Address address = new Address();
```

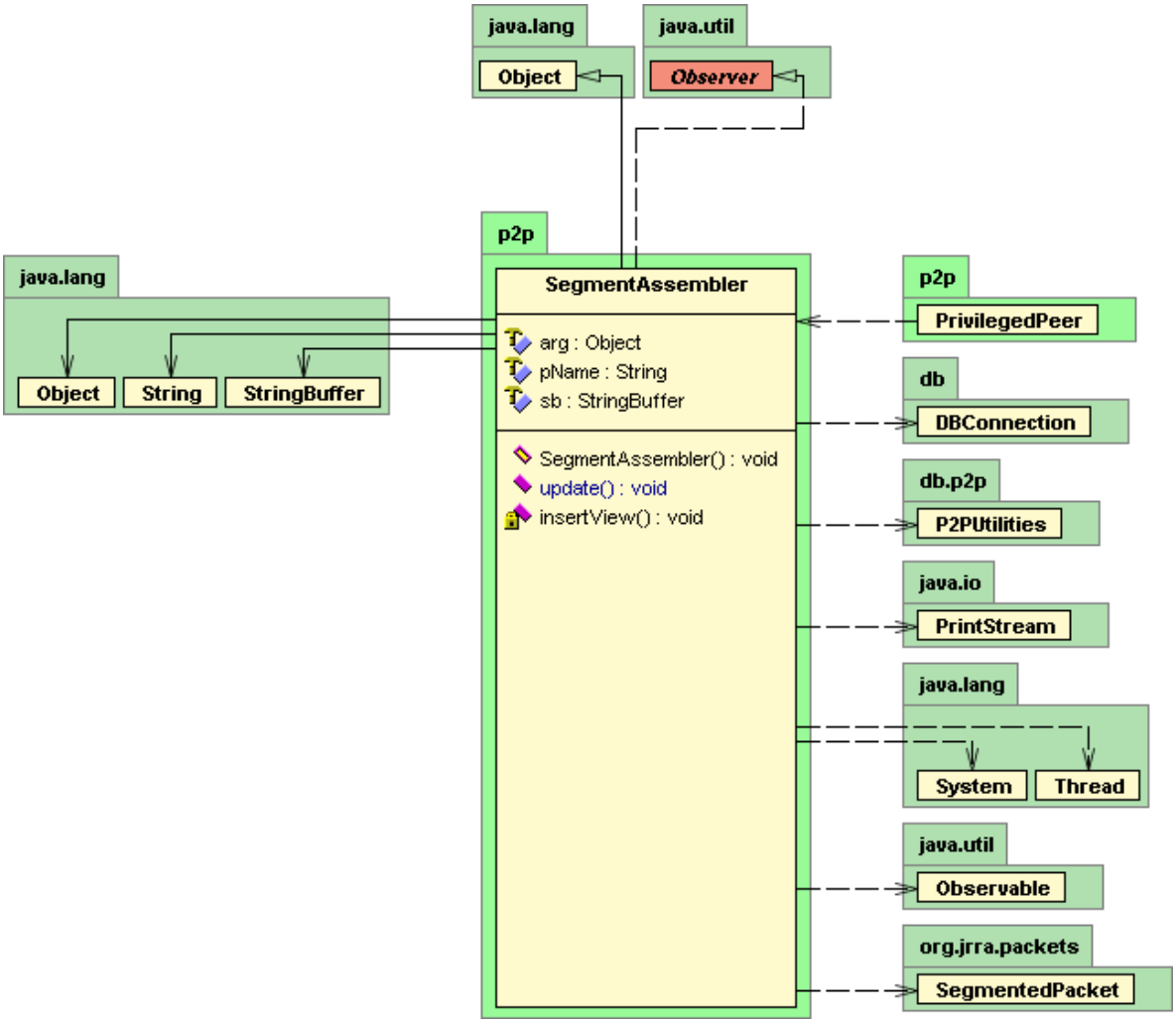
```
    address.setAttribute("inetAddress", udpIp);
    address.setAttribute("port", port);
    ownPeer.setAddress(protocol, address);
    ownPeer.setDefaultProtocolType(protocol);
}


/**
 * Setea el nombre del punto privilegiado
 */
public void setName(String name) {
    ownPeer.setAttribute("name", name);
}

/**
 * Obtiene el nombre del punto privilegiado
 */
public String getName() {
    //return ownPeer.getProperties().getPropertyAsString(new SerializeString("name"));
    return ownPeer.getStringAttribute("name");
}

/**
 * Obtiene la referencia al punto de la red P2P.
 */
public Peer getPeer() {
    return ownPeer;
}

/**
 * Obtiene el identificador único del punto.
 */
public int getUIN() {
    return ownPeer.getUin();
}


/**
 * Evento que se dispara al recibir un mensaje.
 */
public void update(EventMessage event) {
    PacketData data = event.getPacketData();
    if (data instanceof RegisterPeerRequestPacket) {
        RegisterPeerRequestPacket packet = (RegisterPeerRequestPacket) data;
        Peer peerToAdd = packet.getPeer();
        int idPeer = db.p2p.P2PUtilities.addNewPeer(peerToAdd.getStringAttribute("name"));
        peerToAdd.setP2PId(idPeer);
        handleRegisterRequestPeer(peerToAdd);
        sendBroadCast(new PeerListPacket(peerList));
    }
    else if (data instanceof SegmentedPacket) {
        SegmentedPacket sp = (SegmentedPacket) data;
        int segmentNum = sp.getSegmentNumber();
        String peerName = sp.getPeerName();
        if (segmentNum <= 1) {
```

```
            SegmentAssembler sa = new SegmentAssembler(peerName);
            this.addObserver(sa);
            this.setChanged();
            this.notifyObservers(new Object[]{sp, peerName});
         }
         else {
            this.setChanged();
            this.notifyObservers(new Object[]{sp, peerName});
         }
      }
      else if (data instanceof QueryPacket) {
         QueryPacket qp = (QueryPacket) data;
         int idPeerFrom = qp.getP2pPeerId();
         Peer peerFrom = peerList.p2pLookup(idPeerFrom);
         Vector v = db.p2p.P2PUtilities.getPeersInSearch(qp.getMessage());
         PeerList qpl = new PeerList();
         for (int x = 0; x < v.size(); x++) {
            PDLibPeer p = (PDLibPeer) v.get(x);
            Peer temp = peerList.p2pLookup(p.getId());
            if (temp!= null) qpl.bind(temp);
         }
         PeerListQueryPacket d = new PeerListQueryPacket(qpl);
         controller.sendMessage(d, peerFrom);
      }
   }

   /**
    * Registra a un nuevo punto en la lista de puntos conectados.
    */
   private void handleRegisterRequestPeer(Peer p) {
      //If the peer change his ip, the app must bind again that peer, so others
      //peers could know his new ip. Always bind the peer again here.
      //peerList.remove(p.getUin());
      peerList.bind(p);
   }

   /**
    * Envía un broaadcast a todos los puntos conectados de la lista de clientes conectados.
    */
   private void sendBroadCast(PacketData data) {
      Enumeration enume = peerList.elements();
      for (;enume.hasMoreElements();) {
         Peer peer = (Peer) enume.nextElement();
         controller.sendMessage(data, peer);
      }
   }
}
```

SegmentAssembler.java

Clase utilizada para ensamblar un paquete recibido por partes.

Diagrama



Código

```
public class SegmentAssembler implements Observer {
   String pName;
   StringBuffer sb = null;
   Object arg = null;
   /** Crea una nueva instancia de SegmentAssembler */
   public SegmentAssembler(String pName) {
      this.pName = pName;
   }

   /**
    * Evento que se dispara al recibir una parte del paquete.
    */
   public void update( Observable obj, Object a) {
```

```
        arg = a;
        Thread t = new Thread() {
            public void run() {
                SegmentedPacket sp = (SegmentedPacket) ((Object[]) arg)[0];
                String peerName = (String) ((Object[]) arg)[1];
                int segmentNum = sp.getSegmentNumber();
                if (peerName.equalsIgnoreCase(pName)) {
                    if (sb == null) sb = new StringBuffer();
                    byte[] temp = sp.getByteArray();
                    sb.append(new String(temp).replaceAll("'", ""));
                    if (segmentNum == sp.getSegmentsNum())
                        insertView(sb.toString(), sp.getPeerName());

                }
            }
        };
        t.start();
    }

    /**
     * Inserta la vista en la base de datos.
     */
    private void insertView(String view, String peerName) {
        int p = db.p2p.P2PUtilities.addNewPeer(peerName);
        DBConnection dbc = new DBConnection();
        String q = "select idView from view where idUser = " + p;
        if (dbc.exists(q)) {
            String u = "update view set view = '"+view+"' where idUser = " + p;
            dbc.update(u);
        } else {
            String u = "insert into view (idUser, view) values ("+p+", '"+view+"')";
            dbc.update(u);
        }
        dbc.closeConnection();
    }
}
```

Adecuaciones a JRRA para el PP

Como ya se mencionó en el capítulo tres, el código de la versión de JRRA para computadoras de escritorio no funciona de manera correcta con la versión para dispositivos móviles (versión micro), esto es debido a que JRRA utiliza sockets para llevar a cabo la comunicación y estos son manejados de manera diferente en la versión estándar que en la versión micro de la máquina virtual, esto provoca que JRRA sea incompatible. Por lo anterior se tuvo que implementar a la versión micro de JRRA en el punto privilegiado haciendo uso de las implementaciones de sockets de la versión micro de java, esto obligo a tener que implementar librerías de CDC y MIDP en el punto privilegiado, sin embargo en las pruebas que se hicieron funcionó bien. Por lo anterior podemos entender que el punto privilegiado hace uso de la máquina virtual de Java versión estándar, pero implementa librerías de la versión micro lo que nos da lo mejor de ambos mundos: un punto privilegiado que puede hacer uso del API de Java versión estándar y de esta forma poder una base de datos entre otras cosas y la capa de comunicación de la versión micro para poder tener comunicación directa con los dispositivos móviles.

Otra implementación que se le tuvo que hacer a esta versión modificada de JRRA fue la adición de paquetes, como ya se mencionó a lo largo de la tesis JRRA trabaja a través del envío y recepción de paquete que no son más que objetos serializados que viajan por la red. Debido a que se tuvo que implementar la versión micro en al punto privilegiado se tuvieron que agregar algunos paquetes del lado del punto privilegiado que no estaba implementados, así como agregar algunos otros para el propósito especifico del uso de servicios de bibliotecas digitales.

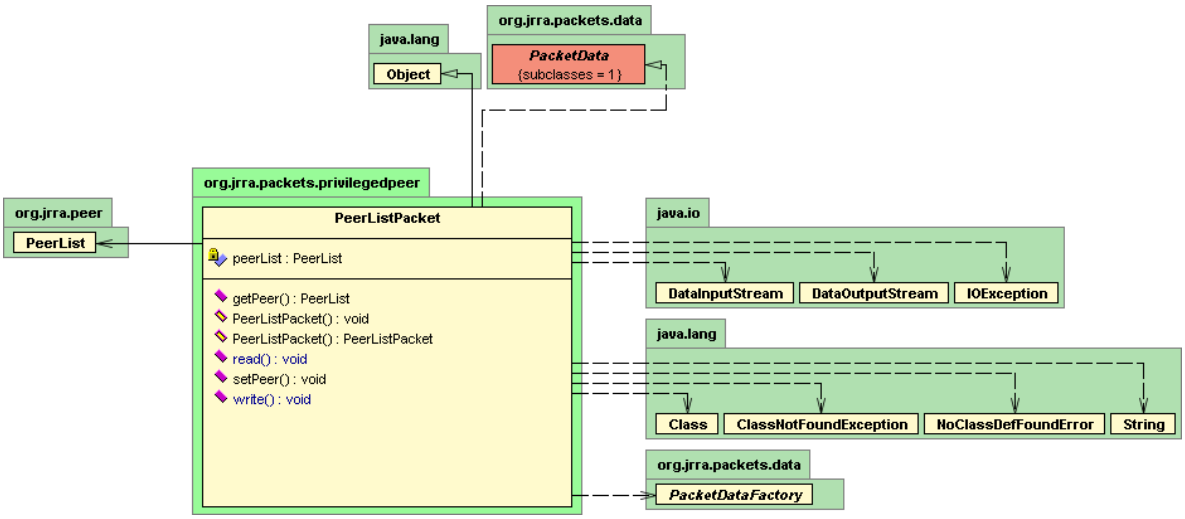
UML y código de las clases agregadas

A continuación se presentan los diagramas UML y el código de cada una de las clases de paquetes agregadas en el punto privilegiado, en el cual se pueden apreciar los métodos, los constructores, las interfaces implementadas y una breve descripción de la clase. Esto le da al desarrollador una idea de la forma en la que se implementaron las clases.

PeerListPacket

Clase para manejar al paquete que contiene la lista de puntos que será enviada a cada punto en la red. Implementa una interfaz llamada PacketData, proveída por JRRA que permite transferirlo por la red a través de punto a punto.

Diagrama



Código

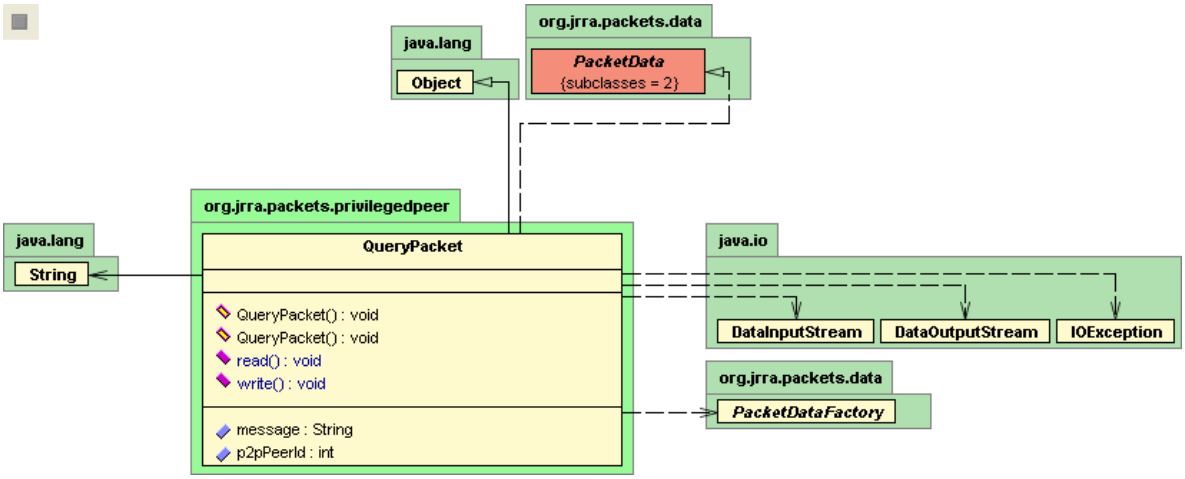
```
public class PeerListPacket implements PacketData {

   static class Factory extends PacketDataFactory {
      protected PacketData create() {
         return new PeerListPacket();
      }
   }


   static {
      PacketDataFactory.addFactory(org.jrra.packets.privilegedpeer.PeerListPacket.class.getName(), new
Factory());
   }

   private PeerList peerList = new PeerList();

   /** Crea una nueva instancia de PeerListQueryPacket
    */
   public PeerListPacket() {
   }

   /** Crea una nueva instancia de PeerListQueryPacket con la lista de puntos pasada como parámetro.
    */
   public PeerListPacket(PeerList peerList) {
      this.peerList = peerList;
   }
```

```
    /**
     * Setea la referencia a la lista de puntos.
     */
    public void setPeer(PeerList peerList) {
        this.peerList = peerList;
    }

    /**
     * Obtiene la referencia a la lista de puntos.
     */
    public PeerList getPeer() {
        return peerList;
    }

    /**
     * Método implementado de la clase Serializable para transferencia en la red
     */
    public void read(DataInputStream in) throws IOException {
        peerList.read(in);
    }

    /**
     * Método implementado de la clase Serializable para transferencia en la red
     */
    public void write(DataOutputStream out) throws IOException {
        peerList.write(out);
    }

}
```

QueryPacket

Clase para manejar al paquete que contiene al query que debe ser ejecutado en la bade de datos. Implementa una interfaz llamada PacketData, proveída por JRRA que permite transferirlo por la red a través de punto a punto.

Diagrama



Código

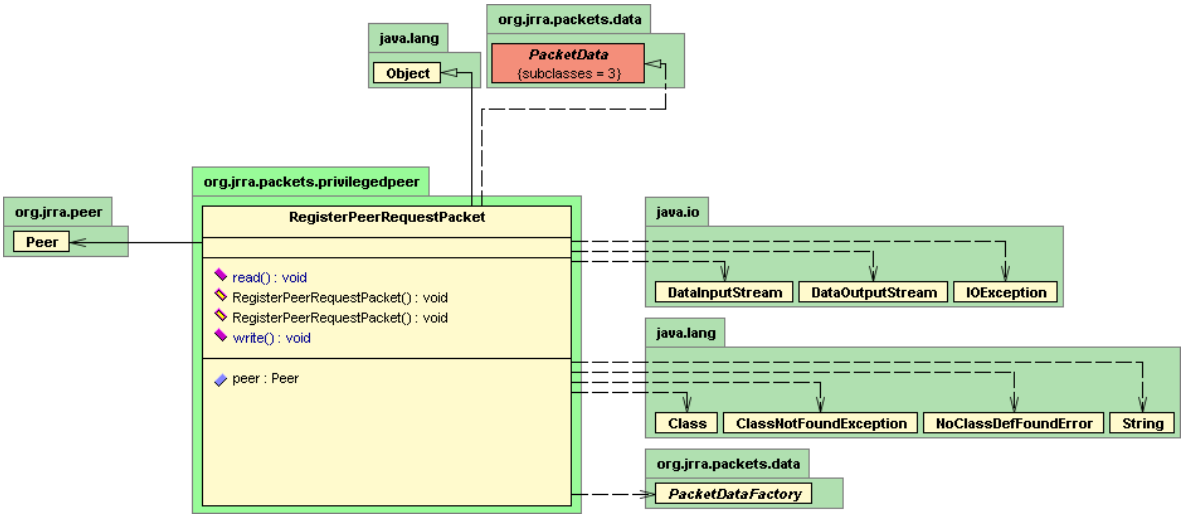
```
public class QueryPacket implements PacketData {

   static class Factory extends PacketDataFactory {
      protected PacketData create() {
         return new QueryPacket();
      }
   }

   /** Crea una nueva instancia de MessagePacket */
   public QueryPacket() {
   }

   private String message = "";
   private int p2pPeerId;

   /** Crea una nueva instancia de MessagePacket con el query y id del punto pasado como parámetro
    */
   public QueryPacket(String message, int p2pPeerId) {
      this.message = message;
      this.p2pPeerId = p2pPeerId;
   }

   /**
    * Obtiene la referencia al query
    */
   public String getMessage() {
      return this.message;
   }
```

```
    /**
     * Obtiene el id del punto que hizo la petición
     */
    public int getP2pPeerId() {
        return p2pPeerId;
    }

    /**
     * Setea la referencia al query
     */
    public void setMessage(String message) {
        this.message = message;
    }

    /**
     * Método implementado de la clase Serializable para transferencia en la red
     */
    public void write(DataOutputStream dataOut) throws IOException {
        dataOut.writeUTF(message);
        dataOut.writeInt(p2pPeerId);
    }

    /**
     * Método implementado de la clase Serializable para transferencia en la red
     */
    public void read(DataInputStream dataIn) throws IOException {
        message = dataIn.readUTF();
        p2pPeerId = dataIn.readInt();
    }

    static {
        PacketDataFactory.addFactory("org.jrra.packets.privilegedpeer.QueryPacket", new Factory());
    }
}
```

RegisterPeerRequestPacket

Clase para manejar al paquete que contiene al punto que acab de llegar a la red P2P y requiere acceso. Además sirve para indicar que el punto contenido en el paquete debe ser agregado a lista de puntos. Implementa una interfaz llamada PacketData, proveída por JRRA que permite transferirlo por la red a través de punto a punto.

Diagrama



Código

```
public class RegisterPeerRequestPacket implements PacketData {

   static class Factory extends PacketDataFactory {
      protected PacketData create() {
         return new RegisterPeerRequestPacket();
      }
   }


static {
         PacketDataFactory.addFactory(org.jrra.packets.privilegedpeer.RegisterPeerRequestPacket.class.getN
         ame(), new Factory());
   }

   private Peer peer = new Peer();

   /**
    * Crea una nueva instancia de RegisterPeerRequestPacket
    */
   public RegisterPeerRequestPacket() {
   }

   /**
    * Crea una nueva instancia de RegisterPeerRequestPacket con el punto pasado como parámetro.
    */
```

```
    public RegisterPeerRequestPacket(Peer peer) {
        this.peer = peer;
    }

    /**
     * Setea la referencia la punto.
     */
    public void setPeer(Peer peer) {
        this.peer = peer;
    }

    /**
     * Obtiene la referencia la punto.
     */
    public Peer getPeer() {
        return peer;
    }

    /**
     * Método implementado de la clase Serializable para transferencia en la red
     */
    public void read(DataInputStream in) throws IOException {
        peer.read(in);
    }

    /**
     * Método implementado de la clase Serializable para transferencia en la red
     */
    public void write(DataOutputStream out) throws IOException {
        peer.write(out);
    }

}
```

7 Bibliografía

1. Roberto García Sanchez. Técnicas de adaptación a la conexión para clientes móviles que accedan servicios de biblioteca digital. Tesis de Maestría, ITESM, Monterrey Campus, Diciembre 2004.
2. Sun Microsystems. Java 2 Platform Micro Edition (J2ME). Available at: *http://java.sun.com/j2me/index.jsp*. Last visited: November 2005.
3. JXTA. JXTA™ technology. Available at: *http://www.JXTA.org*. Last visited: June 2005.
4. JRRA. JRRA Project. Available at: *Http://www.JRRA.org*. Last visited: April 2005.
5. An Overview of the File Connection Optional Package. Available at: *http://developers.sun.com/techtopics/mobility/apis/ttips/fileconnection/*. Last visited: November 2005.
6. Sun Microsystems. MIDP Database Programming Using RMS: a Persistent Storage for MIDlets. Available at: *http://developers.sun.com/techtopics/mobility/midp/articles/persist/*. Last visited: October 2005.
7. ITESM, Monterrey Campus. Phronesis Biblioteca Digital. Available at: *http://copernico.mty.itesm.mx/phronesis/project/aboutdl.htm*. Last visited: October 2005.
8. Morpheus. Morpheus project. Available at: *http://morpheus.com/*. Last visited: August 2005.
9. Gnutella. Gnutella Project. Available at: *http://www.gnutella.com/*. Last visited: August 2005.
10. Napster. Napster Project. Available at: *http://www.napster.com/*. Last visited: August 2005.
11. Extensible Markup Language (XML) 1.0 (Third Edition). Available at: http://www.w3.org/TR/2004/REC-XML-20040204/. Last visited: October 2005.

12. B. Bhargava and M. Annamalai. Communication costs in digital library databases. *In Lecture Notes in Computer Science Series (LNCS 1978),* Database and Expert Systems Applications (DEXA '95), pages 1-13. Springer-Verlag, September 1995.
13. B. Bhargava and M. Annamalai. Digital Library Services in Mobile Computing. Evangelina Pitoura. *Department of Computer Science University of Ioannina*. December 1995.
14. M. Satyanarayanan. Fundamental challenges in mobile computing. *In Symposium on Principles of Distributed Computing*, pages 1–7, 1996.
15. Wayne Hanslo. The efficiency of XML as an Intermediate Data Representation for wireless Middleware Communication. ACM International Conference Proceeding Series; Vol. 75 Proceedings of the 2004 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries. Stellenbosch, Western Cape, South Africa Pages: 279 - 283. 2004.
16. B. Ooi, Y. Shu, K.L. Tan, and A.Y. Zhou. PeerDB: A P2P-based System for Distributed Data Sharing. *In ICDE*, 2003.
17. F. Álvarez, D. Garza-Salazar, J. Lavariega, L. Gómez-Martínez. M. Sordia. "PDLib: Personal Digital Libraries with Universal Access", *Joint International Conference on Digital Libraries, Denver CO*, July 2005.
18. Daniel Brookshier, Darren Govoni, Nayaneet Krishnan, Juan Carlos Soto. *JXTA: Java P2P Programming*. December 2002.
19. Cecilia mascolo, Licia capra, Stefanos zachariadis, Wolfgang emmerich. XMIDDLE: A Data-Sharing Middleware for Mobile Computing. *Department of Computer Science, University College London, Gower Street, London WC1E 6 BT, U.K.*
20. W. S. Ng, B. C. Ooi, and K. L. Tan. Bestpeer: A selfconfigurable peer-to-peer system. *In Proceedings of the 18th International Conference on Data Engineering*, page 272, San Jose, CA, April 2002 (Poster Paper).
21. Ali Raza Butt, Troy A. Johnson, Yili Zheng, and Y. Charlie Hu. Kosha: A Peer-to-Peer Enhancement for the Network File System. *Purdue University West Lafayette, IN 47907*.

22. Adina Crainiceanu, Prakash Linga ,Johannes Gehrke, Jayavel Shanmugasundaram. PTree: A P2P Index for Resource Discovery Applications. *Department of Computer Science, Cornell University*.
23. D. Comer. The ubiquitous B-tree. *In Computing Surveys, 11(2)*, 1979.
24. MYSQL. Available at: *http://www.MYSQL.com*. Last visit: October 20.
25. OAI. Available at: *http://www.openarchives.org/*. Last visit: November 15.
26. Michel Nelson, Simeon Warner, Carl Lagoze, and Carl Van de Pompel.The open archives initiative protocol for metadata harvesting. Available at: http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm. Last visit: November 2005.
27. Luis Basto. Modelo de Soporte de Operaciones Offline en Dispositivos Móviles para una Biblioteca Digital. Tesis de Maestría, ITESM, Monterrey Campus, Diciembre 2005.
28. Building Graphical User Interfaces with the MVC Pattern. Available at: http://csis.pace.edu/~bergin/mvc/mvcgui.html. Last visit: December 1.