



**UNIVERSIDAD DE CHILE  
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS  
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN**

**MODELO DE COMUNICACIÓN PUNTO A PUNTO PARA APLICACIONES  
COLABORATIVAS EN DISPOSITIVOS MÓVILES**

**MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN  
COMPUTACIÓN**

**LUIS FERNANDO AVELLO HUALA**

**PROFESOR GUÍA:  
LUIS GUERRERO BLANCO**

**MIEMBROS DE LA COMISIÓN:  
NELSON BALOIAN TATARYAN  
ALEXANDRE BERGEL**

**SANTIAGO DE CHILE  
ENERO 2010**

# Agradecimientos

Muchas gracias a mi familia que me ha apoyado durante todo este camino, especialmente a mis padres y hermanos.

Luis

RESUMEN DE LA MEMORIA  
PARA OPTAR AL TÍTULO DE  
INGENIERO CIVIL EN COMPUTACIÓN  
POR: LUIS AVELLO H.  
FECHA: 17/01/2010  
PROF. GUÍA: SR. LUIS GUERRERO

## MODELO DE COMUNICACIÓN PUNTO A PUNTO PARA APLICACIONES COLABORATIVAS EN DISPOSITIVOS MÓVILES

El objetivo general del presente trabajo es proponer e implementar un modelo de comunicación punto a punto para aplicaciones colaborativas en dispositivos móviles, con el fin de facilitar el desarrollo de dichas aplicaciones.

Para conseguir este objetivo, el trabajo fue dividido en una etapa de investigación y en una etapa de aplicación. En la etapa de investigación se estudió los sistemas colaborativos, la arquitectura punto a punto, protocolos de comunicación y la plataforma móvil seleccionada (teléfonos *iPhone*). En la etapa de aplicación se utilizaron los conocimientos adquiridos para diseñar y posteriormente desarrollar el modelo de comunicación, junto a dos aplicaciones que hacen uso de este modelo.

El resultado final es una implementación del modelo que posee todas las características necesarias para proveer comunicación punto a punto entre equipos móviles, tales como detección, búsqueda, múltiples conexiones y envío y recepción de datos entre estos equipos. Junto a esto se tiene una aplicación real que opera exitosamente, y una aplicación de ejemplo a modo de apoyar el diseño y construcción de otros sistemas.

Se concluye que actualmente es factible poder desarrollar aplicaciones colaborativas que posean comunicación punto a punto utilizando teléfonos celulares, dados los últimos avances en tecnología, aunque se deben tener en cuenta las limitaciones de los protocolos de comunicación.

# Índice General

1. Introducción.....	6
2. Motivación.....	8
3. Objetivo General.....	10
3.1. Objetivos Específicos.....	10
4. Revisión Bibliográfica.....	11
4.1. Sistemas Colaborativos.....	11
4.1.1. Patrones de Diseño.....	11
4.2. Desarrollo de software en dispositivos móviles.....	13
4.2.1. Sistemas Operativos.....	13
4.3. Tecnologías de comunicación en dispositivos móviles.....	14
4.3.1. Tecnologías de transmisión entre dispositivos.....	14
4.4. Arquitectura Punto a Punto.....	15
4.4.1. Características y Ventajas.....	15
5. Procedimiento.....	17
5.1. Plataforma y Herramientas.....	17
6. Desarrollo y Resultados.....	19
6.1. Modelo propuesto.....	19
6.2. Detección de dispositivos.....	19
6.2.1. Bonjour.....	20
6.3. Establecimiento de conexiones TCP.....	21

6.4. Estados de un peer.....	22
6.5. Establecimiento de conexión entre peers.....	24
6.6. Envío de datos entre peers.....	28
6.7. Modos de Operación.....	29
6.8. Protocolos Soportados.....	30
6.9. Aplicación Communication.....	32
6.10. Aplicación CSphotoshare.....	37
7. Conclusiones.....	46
7.1. Trabajo Futuro.....	47
8. Bibliografía.....	48
A. ANEXO.....	51
A.1. Código fuente aplicación Communication.....	51
A.1.1. CommunicationViewController.h.....	51
A.1.2. CommunicationViewController.m.....	51
A.1.3. P2PCommunication.h.....	54
A.1.4. P2PCommunication.m.....	56

# 1. Introducción

Actualmente y debido al creciente dinamismo de la sociedad actual, una gran cantidad de personas necesitan moverse de un lugar a otro para cumplir con sus asuntos laborales. Con los últimos avances tecnológicos se está haciendo posible que estas personas puedan trabajar incluso cuando viajan en un bus, avión, tren, o cuando simplemente caminan. Esto gracias a dispositivos tales como notebooks, netbooks (introducidos recientemente en el mercado), PDAs y teléfonos celulares, los cuales han experimentado un aumento prácticamente sostenido de sus capacidades (procesamiento, conectividad, almacenamiento y visualización). A esto se suma la reducción de precios observada durante los últimos años de dichos dispositivos, haciendo que éstos penetren con mayor éxito en áreas como la educación, salud, actividades productivas y de servicios.

Gracias a estas nuevas capacidades de los dispositivos móviles, ahora es posible utilizarlos en aplicaciones más sofisticadas que puedan realmente aprovechar las capacidades de movilidad de ellos.

Además de la necesidad de las personas de moverse de un lugar a otro, también es muy habitual que los empleados de una organización deban formar parte de equipos de trabajo. En estos equipos de trabajo es necesaria una comunicación expedita y que se trabaje de manera coordinada. La finalidad es que el equipo sea productivo, que sus integrantes se mantengan altamente motivados y que efectivamente exista colaboración entre los integrantes. Es aquí donde los sistemas colaborativos hacen su aporte.

Un sistema colaborativo (también conocido como *groupware*) es un sistema que apoya a un grupo de personas que trabajan en una meta común. Este sistema que se basa en tecnología (*software* y *hardware*) provee una interfaz a un ambiente compartido [1]. Además provee los medios necesarios para que los usuarios se comuniquen y trabajen de manera coordinada, por lo que la comunicación es un aspecto muy relevante al momento de diseñar una aplicación colaborativa.

Las principales características de los sistemas colaborativos son: proveer un ambiente de colaboración, mantener la información en un sitio común para todos los miembros del grupo y permitir la interacción entre los usuarios, ya sea de manera escrita o utilizando voz y/o video.

Los sistemas colaborativos deben proporcionar tres funciones esenciales para apoyar un grupo de personas [2]:

- La comunicación, que es el medio en que la información es compartida.
- La colaboración, actitud de ayuda y servicio hacia el trabajo, permite resolver problemas de negocios o alguna actividad empresarial.
- La coordinación, que es la acción de asegurar que el equipo está trabajando eficientemente y en conjunto para alcanzar una tarea o meta común.

Algunos ejemplos de herramientas colaborativas son: los grupos de noticias, los espacios de trabajo de grupos, sistemas que soportan la toma grupal de decisiones, y las aplicaciones de aprendizaje mediante escritura colaborativa.

Los sistemas colaborativos se están haciendo cada vez más populares dentro de las empresas, ya que el costo de mantener una red de comunicaciones y comprar un sistema de colaboración es menor que estar financiando el traslado del personal de un lugar a otro [2].

Un dispositivo móvil, como un teléfono celular, tiene un costo mucho menor en comparación con un computador personal. En consecuencia utilizar dispositivos móviles como parte de un sistema colaborativo ampliará el número de escenarios posibles en donde se puede realizar trabajo grupal, trabajo que antiguamente estaba limitado por la necesaria presencia de un computador.

Para apoyar el desarrollo de aplicaciones colaborativas en escenarios móviles se propone un modelo de comunicación para dichas aplicaciones. Además de entregar un diseño e implementación del modelo, se desarrollará una aplicación de ejemplo para facilitar la construcción de otras aplicaciones.

## 2. Motivación

Desarrollar aplicaciones colaborativas para computadores tradicionales es una tarea difícil, debido a que los sistemas colaborativos involucran problemas que no aparecen en los sistemas de un único usuario, tales como la comunicación persona a persona, dinámicas de grupo, roles sociales, memoria grupal, y otros factores sociales [3]. Esto se complica aún más si queremos diseñar aplicaciones colaborativas para dispositivos móviles tales como teléfonos celulares, dado que estos dispositivos poseen restricciones como el tamaño de la pantalla, la limitada capacidad de ingreso de datos, la corta duración de la batería, la limitada cantidad de memoria y el bajo nivel de procesamiento (CPU) [4]. Además estos dispositivos son bastante heterogéneos (arquitectura de *hardware*, sistema operativo, protocolos de comunicación, entre otros aspectos).

A lo anterior se suman las condiciones de comunicación entre los dispositivos. En el escenario móvil en general se tienen bajas tasas de transferencia, inestabilidades en la conexión y cambios repentinos en la topología de la red. Por lo anterior, un modelo *cliente/servidor* no se adapta de la mejor forma a las características que posee un escenario móvil, especialmente si se trata de teléfonos celulares.

Un dispositivo puede tener fortalezas y debilidades dependiendo del contexto de trabajo en que se utiliza. Luego es necesario efectuar una evaluación para determinar si el dispositivo es adecuado, dado un escenario específico [5].

Otro aspecto a tener en cuenta es que al desarrollar aplicaciones colaborativas no sólo se debe considerar el análisis y la especificación de requerimientos funcionales, también hay que considerar los requerimientos no funcionales. Estos últimos aseguran la aplicabilidad y la usabilidad de una solución de *software* en un ambiente de trabajo determinado [6].

Siendo que ya existen herramientas colaborativas en PDAs (*Personal Digital Assistants*) [7], actualmente no existe una adecuada solución al problema de comunicación de las aplicaciones colaborativas en un ambiente con alta movilidad con teléfonos celulares, ya que estos aparatos paulatinamente están incorporando protocolos de comunicación presentes en



PDA's, como es el caso de IEEE 802.11 (*Wi-Fi*).

Por las razones antes expuestas es que el desarrollo de aplicaciones colaborativas en dispositivos móviles se convierte en una ardua labor.

Este trabajo de memoria tiene como meta especificar un modelo de comunicación punto a punto (*peer to peer*) para dispositivos móviles. Es decir, en vez de tener clientes y servidores fijos, se tendrán nodos que se conectan entre ellos. Esto facilitará el desarrollo de aplicaciones colaborativas en dispositivos móviles. Los dispositivos móviles que se consideraran para esta memoria serán equipos celulares, específicamente teléfonos *iPhones*.

## 3. Objetivo General

La intención de esta memoria es especificar un modelo de comunicación *punto a punto* para aplicaciones colaborativas en dispositivos móviles, más específicamente en equipos celulares. Además de especificar el modelo, se deberá presentar una implementación de éste, y desarrollar y probar una aplicación colaborativa que lo utilice.

### 3.1. Objetivos Específicos

Los objetivos específicos son los siguientes:

- Conocer qué problemas resuelven los sistemas colaborativos y de qué manera lo hacen. Es decir, investigar el estado del arte.
- Saber qué aspectos hay que considerar a la hora de diseñar una aplicación colaborativa, ya sean tecnológicos o de índole social.
- Diseñar un modelo de comunicación entre dispositivos móviles considerando las tecnologías actuales en dispositivos móviles.
- Diseñar, implementar, implantar y probar una aplicación colaborativa en un ambiente real, basada en el modelo de comunicación propuesto.

## 4. Revisión Bibliográfica

### 4.1. Sistemas Colaborativos

Los sistemas colaborativos son sistemas basados en computadores que apoyan a un grupo de personas en la realización de una tarea común. Estos sistemas además proveen una interfaz para un ambiente compartido.

Las aplicaciones colaborativas se pueden clasificar según el espacio o el tiempo. Según el espacio estas aplicaciones pueden estar en el mismo lugar (por ejemplo una sala) o de manera distribuida. Según el tiempo las aplicaciones pueden ser síncronas o asíncronas. En las aplicaciones asíncronas se permite que los usuarios interactúen en tiempos distintos. Ejemplos de esta aplicaciones son los calendarios y sistemas de aprendizaje basados en escritura colaborativa. Otros ejemplos de herramientas colaborativas de este tipo son los grupos de noticias, los espacios de trabajo de grupos, sistemas que apoyan la toma de decisiones grupales.

#### 4.1.1. Patrones de Diseño

En la literatura se pueden encontrar algunos patrones de diseño los cuales pueden estar presentes en el desarrollo de sistemas colaborativos [8]:

- *Sesiones*: Las sesiones mantienen información acerca de los usuarios que interactúan (sincrónicamente o asincrónicamente) al utilizar una aplicación colaborativa. Los usuarios de una sesión específica no pueden ver el trabajo de otros usuarios que utilizan la misma aplicación pero que están en una sesión diferente.
- *Usuarios*: Los usuarios son los miembros de una sesión. Cada usuario mantiene información personal y además se puede especificar los permisos sobre algún recurso dependiendo del rol del usuario.

- *Roles*: Los usuarios no tienen necesariamente los mismos permisos o accesos a los recursos compartidos. Los roles establecen el nivel de acceso de los usuarios a los recursos compartidos.
- *Mensajes*: Es necesario que durante una sesión de trabajo los usuarios necesiten enviar mensajes entre ellos. Los mensajes pueden ser notificaciones, comandos o eventos. Un usuario puede enviar un mensaje a todos los usuarios, a una parte o solo a un usuario específico.
- *Meta-Objetos*: Los meta-objetos contienen información sobre los objetos compartidos por los usuarios. Generalmente esta información corresponde a atributos tales como el dueño, la fecha y hora de creación y modificación, versiones previas, etc.
- *Repositorios*: Los usuarios al realizar trabajo colaborativo generan datos, y tales datos deben ser almacenados y estar disponibles para que los usuarios puedan compartirlos.
- *Vistas*: Las vistas permiten establecer de qué modo los datos almacenados en el repositorio sean visibles a los usuarios. La vista se ajusta para mostrar una porción mayor o menor de los datos compartidos dependiendo del rol del usuario.
- *Ambientes*: Los ambientes organizan y coordinan las múltiples sesiones de trabajo que se generan en por el uso de aplicaciones colaborativas.
- *Control de Flujo*: El control de flujo se refiere a la manera de administrar el acceso a los recursos compartidos. De este modo se establece el comportamiento cuando 2 o más usuarios acceden al mismo recurso compartido.

## 4.2. Desarrollo de software en dispositivos móviles

Durante los últimos años ha crecido enormemente la oferta de equipos celulares. También ha mejorado sustancialmente las capacidades de estos equipos posibilitando la creación de nuevas aplicaciones, y ahora es posible ejecutar aplicaciones desarrolladas por terceros, y no solamente por los mismos fabricantes.

### 4.2.1. Sistemas Operativos

Los principales sistemas operativos para equipos celulares son: Symbian, Windows Mobile, iPhone OS, Android (Linux, Google) y Windows CE. De todos ellos Symbian es uno de los más utilizados especialmente en equipos Nokia. Windows Mobile y Linux están ganando popularidad en los llamados Smartphones, que son equipos celulares que tienen características muy similares a una PDA. En el caso de las PDAs, los sistemas operativos más utilizados son Palm OS (en el caso de equipos Palm) y Windows Mobile, pero últimamente también existen algunos equipos con linux. Cabe destacar que el uso de las PDA ha disminuido paulatinamente, y están siendo reemplazadas por equipos celulares con mayores capacidades, por ejemplo el *iPhone* de Apple o la Blackberry de RIM, y por notebooks diseñados específicamente para escenarios móviles (llamados Netbooks), que han irrumpido fuertemente este último tiempo.

En general, es posible desarrollar aplicaciones nativas dependiendo del sistema operativo que utilice la aplicación, y de la disponibilidad del respectivo SDK. Pero esto tiene el inconveniente de que, dependiendo del equipo a utilizar, se tiene que reescribir el código (inclusive utilizar otro lenguaje), perdiendo la portabilidad de la aplicación.

En la actualidad la mayoría de los equipos cuentan con Java ME (Java Micro Edition)[9], permitiendo desarrollar aplicaciones casi sin importar en qué dispositivo van a ser ejecutadas. Sin embargo esto no es del todo cierto, ya que el tamaño de la pantalla y el layout de los teclados difieren de un equipo a otro, inclusive si son de la misma empresa. Al existir estas diferencias pueden aparecer ciertos problemas, por ejemplo, de visualización o de funcionamiento debido a teclas mal mapeadas.

### 4.3. Tecnologías de comunicación en dispositivos móviles

El protocolo más extendido y utilizado en la actualidad para dispositivos celulares es GPRS (*General Packet Radio Service*), que corresponde a una extensión de la tecnología de comunicaciones móviles GSM (*Global System for Mobile Communications*). En este protocolo, la transmisión de datos es realizada vía paquetes, en donde se alcanzan velocidades de transferencia de 56 a 114 kbps.

GPRS se puede utilizar para servicios tales como WAP, servicio de mensajes cortos (SMS), servicio de mensajería multimedia (MMS), servicio de correo electrónico, servicio de mensajería instantánea y banda ancha móvil. Esto le permite a las operadoras ofrecer un sinnúmero de servicios asociados a sus clientes de telefonía móvil.

Otras tecnologías y protocolos de comunicación más recientes, similares a GPRS, son EDGE, EVDO y HSPA [10].

#### 4.3.1. Tecnologías de transmisión entre dispositivos

- *Wi-Fi*: Es una de las tecnologías de comunicación inalámbrica más utilizada en la actualidad. Para establecer una red usando este protocolo, es necesario un punto de acceso (por lo general un router inalámbrico) y un dispositivo *Wi-Fi* que se conecte a dicho punto. *Wi-Fi* es el nombre comercial con que se conoce a la familia de estándares 802.11. Los estándares más comunes son el 802.11b y 802.11g de la IEEE [11].
- *Bluetooth* [12]: Este protocolo de transmisión inalámbrica de datos está especialmente diseñado para dispositivos de bajo consumo eléctrico, permitiendo conectar dispositivos como teléfonos celulares, auriculares, teclados, etc. Tiene una tasa de transferencia mucho menor que *Wi-Fi*, además de tener menos alcance, pero su bajo costo y poco consumo lo hacen ideal para equipos móviles. Otra ventaja que posee es que no requiere de una infraestructura de comunicación, permitiendo que 2 dispositivos establezcan comunicación directamente.

## 4.4. Arquitectura Punto a Punto

El concepto punto a punto (*peer to peer, P2P*)[13] hace referencia a una arquitectura distribuida, en que los recursos son compartidos entre *peers*, los que actúan tanto como clientes y servidores a la vez. Un *peer* corresponde a un nodo en una red P2P. En una arquitectura P2P pura, estos nodos tienen la misma responsabilidad, en donde no existe una entidad central que controle y coordine el acceso a los recursos en la red. En el caso que un nodo sea removido de la red, esta última no debe perder funcionalidad ni consistencia de la información.

### 4.4.1. Características y Ventajas

Las características y ventajas que se pueden identificar en esta arquitectura son las siguientes [14]:

- *Capacidad*: Se aprovechan recursos tales como el ancho de banda, almacenamiento y poder de procesamiento de cada nodo de la red
- *Independencia*: Se puede implementar una arquitectura distribuida independiente de los recursos centralizados
- *Descentralización*: La funcionalidad y los servicios están ubicados en cualquier lugar de la red, lo que evita que se produzcan cuellos de botella, que son característicos en la clásica arquitectura cliente-servidor.
- *Extensibilidad*: Un nodo puede ingresar a una red P2P, e inmediatamente proveer nuevos servicios o recursos a los demás nodos.
- *Tolerancia a fallas*: Ante la falla de un nodo, es posible implementar un esquema de replicación
- *Escalabilidad*: Dado que se evita la centralización, esto permite que la red pueda crecer con mayor facilidad

Al considerar un escenario móvil, la comunicación de los nodos en una red P2P estaría implementada considerando las siguientes opciones:

- *Sin una infraestructura:* La comunicación se efectúa a través de una red ad-hoc, sin que se utilice o sea necesaria una infraestructura existente. Un ejemplo de esto sería el uso del protocolo *Bluetooth*.
- *Con una infraestructura:* La comunicación es realizada a través de puntos de acceso (*Wi-Fi*) o de la red celular (GSM, GPRS). Si no existe la infraestructura (routers inalámbricos, puntos de acceso, antenas de celular) no es posible establecer comunicación.



## 5. Procedimiento

Para desarrollar el trabajo propuesto, lo he dividido en las siguientes etapas:

- Aprender el lenguaje de programación y las herramientas de desarrollo para la plataforma móvil que se utilizará.
- Investigar las librerías o APIs disponibles en la actualidad para realizar transmisión de datos entre los dispositivos móviles.
- Diseñar un modelo de comunicación punto a punto entre dispositivos móviles
- Realizar una implementación del modelo anteriormente descrito.
- Desarrollar una aplicación muy simple que utilice la capa de comunicación, a modo de prueba de concepto.
- Desarrollar una aplicación real que integre la capa de comunicación, demostrando así su utilidad.

### 5.1. Plataforma y Herramientas

Los equipos móviles que se utilizarán son teléfonos *iPhone*, que poseen características como pantalla táctil, cámara de fotos, reproductor de música, correo electrónico, navegador web, además de las funcionalidades propias de un teléfono celular. También incluyen conectividad *Wi-Fi* y *Bluetooth*.

Una de las características más llamativas que posee este dispositivo es el servicio *App Store*, que le permite al usuario descargar e instalar aplicaciones gratuitas o de pago, expandiendo notablemente las funcionalidades que posee el equipo. Actualmente hay más de 100.000 aplicaciones disponibles para los usuarios.



Figura 01. iPhone

Para desarrollar aplicaciones para el *iPhone* [15], primero se debe contar con el kit de desarrollo (*SDK*). Para efectos de este trabajo, se utilizó el *SDK* para iPhone OS 3.0, el que incluye las herramientas *Xcode* (IDE para el código fuente) e *InterfaceBuilder* (ambiente visual para el desarrollo de las interfaces de usuario). El equipo para realizar el desarrollo cuenta con Mac OSX Leopard 10.5.

Los equipos móviles con que se cuenta para realizar el desarrollo y el testing son los siguientes:

- 1 iPhone 2G
- 1 iPhone 3G
- 1 iPhone 3GS

Todos estos equipos cuentan con el sistema operativo iPhone OS 3.0. El motivo de contar con equipos diversos es garantizar la compatibilidad de la aplicación que se desarrolle.

El lenguaje de programación que se utiliza es *Objective-C* [16], que corresponde a un lenguaje orientado a objetos creado como un superconjunto de *C*, y que provee sintaxis para definir clases, métodos, además de incluir conceptos tradicionales de los lenguajes orientados a objetos, como herencia, encapsulación, polimorfismo y tipado dinámico.

## 6. Desarrollo y Resultados

### 6.1. Modelo propuesto

El modelo que se plantea es uno basado en capas. Las capas inferiores implementan el establecimiento y manejo de múltiples conexiones *peer to peer*. Una para *Bluetooth*, y otra para *Wi-fi* (via sockets TCP). De este modo se logra una abstracción del protocolo de comunicación (*P2P Session*). La serialización de los objetos a transmitir queda a cargo de *File Transfer* y *Message Transfer*. Como última capa se encuentra la aplicación móvil (*Mobile App*).

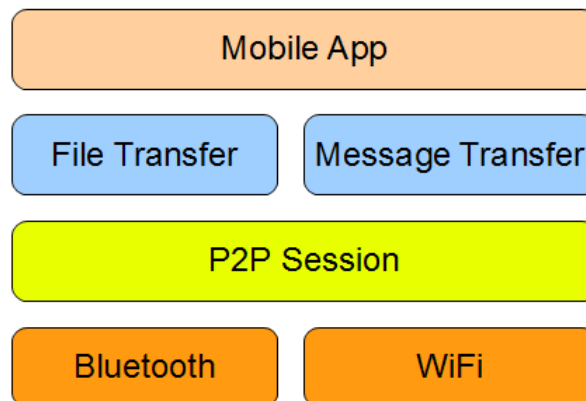


Figura 02: Modelo de comunicación propuesto

Como objetivo principal se busca compartir información o datos a través del uso de este modelo de comunicación, además de tener una única interfaz, independiente del protocolo a utilizar.

### 6.2. Detección de dispositivos

El primer problema que aparece al intentar establecer conexiones *peer to peer* por *Wi-Fi*, es el de la detección de los demás dispositivos que se encuentran presentes en la red, además de determinar que direcciones IP poseen y sus respectivos puertos de escucha.

Una opción es implementar esta detección por medio de la realización de broadcast UDP, cosa que es posible de realizar en el *iPhone*, pero después de investigar se encontró una tecnología que resuelve este problema: *Bonjour* [17].

### 6.2.1. Bonjour

*Bonjour* permite descubrir dispositivos o servicios en una red local, sin la necesidad de contar con algún servidor de por medio. Estos dispositivos tienen asociado un nombre, el que además tiene asociado su IP y puerto, datos que se obtienen luego del proceso de resolución, algo similar al sistema de *DSN (Domain Name System)*. Además de la búsqueda de dispositivos, también permite su publicación para que el dispositivo pueda ser visible y detectado por los demás. *Bonjour* corresponde a la implementación de Apple del protocolo *ZeroConf* [18].

Aunque habitualmente *Bonjour* es utilizado en el contexto de una red fija (computadores de escritorio, impresoras), este protocolo está disponible en el *iPhone* [19] y es ampliamente utilizado por las aplicaciones (principalmente juegos) que establecen conexiones via *Wifi* entre estos teléfonos.

Para identificar y diferenciar a los peers que se encuentran presentes, se utiliza un identificador único: *peerID*. En la práctica este identificador corresponde al nombre del teléfono, el que es retornado por *Bonjour* una vez que pudo publicarse exitosamente. Además *Bonjour* asegura que sea único en la red, por lo que si ocurre alguna colisión (dos equipos poseen el mismo nombre), se modifica este identificador antes de publicarse.

Al momento que se detecta un dispositivo, se notifica indicando su identificador. Lo mismo ocurre cuando ya no está disponible, es decir, en el caso que el *peer* desactive su publicación.

Para la identificación de la aplicación se utiliza el *sessionID*. Esto permite que dentro de una red distintas aplicaciones no interfieran entre ellas, ya que sólo serán visibles por *Bonjour* los dispositivos que posean el mismo *sessionID*. Este identificador se configura tanto al momento de buscar dispositivos, como en la publicación.

### 6.3. Establecimiento de conexiones TCP

Dado el uso de *Bonjour*, el problema de la detección se encuentra solucionado. El siguiente paso corresponde al establecimiento de conexiones TCP. Una alternativa es utilizar directamente sockets BSD (lenguaje C), pero el trabajo sería bastante arduo, ya que se deben solucionar problemas básicos como el bloqueo al escribir o leer por el socket, y la dificultad para manejar múltiples conexiones. Apple provee APIs de alto nivel para el uso de *http*, pero lamentablemente en la actualidad carece de una para el manejo directo de sockets en lenguaje *Objective-C*, y que tenga resuelto los problemas anteriormente mencionados.

Por tal razón, se escogió el uso de la librería open source *AsyncSocket* [20], la que tiene los siguientes beneficios:

- Interfaz orientada al uso de objetos en *Objective-C*.
- Colas de escritura y lectura no bloqueantes.
- Al recibir una nueva conexión, se crea automáticamente un nuevo socket, por lo que esta adaptada para recibir múltiples conexiones.
- Uso del patrón de diseño *Delegate* para notificar envío de datos, recepción de datos, nuevas conexiones, errores y desconexiones.
- Funcionamiento asíncrono.

## 6.4. Estados de un *peer*

Con la utilización de la librería antes descrita, ya se logra establecer conexiones *TCP* y enviar y recibir datos por sockets. Ahora corresponde establecer un protocolo para que un *peer* pueda conectarse a otro *peer*, de modo de lograr que sólo exista una conexión *TCP* entre ambos *peers*. Más adelante se detalla el caso en que sucede esta situación.

Para esto se han definido una serie de estados en que un *peer* se podrá encontrar, los que se detallan a continuación:

- *Disconnected*: Corresponde al primer estado en que se encuentra un *peer*. Luego puede pasar al estado *Available* si el *peer* se encuentra disponible para recibir conexiones, o puede pasar al estado *Connecting Server* si el *peer* corresponde a una conexión entrante.
- *Connected*: Estado que indica que el *peer* se encuentra completamente conectado. En esta situación solo puede pasar al estado *Disconnected* si se pierde la conexión.
- *Connecting Client*: Cuando el componente de cliente desea establecer conexión con un *peer* se llega a este estado.
- *Connecting Server*: Al recibir un intento de conexión por parte de un *peer* se pasa a este estado, ya que corresponde a la componente de servidor.
- *Available*: Un *peer* esta disponible cuando es visible por los demás *peers*, quienes dependiendo de caso podrán conectarse.
- *Unavailable*: Cuando un *peer* no desea recibir conexiones es que se encuentra en estado no disponible.

Cuando se establece una conexión entre dos *peers* , en un caso entra en operación la componente de cliente, y en el otro *peer* la componente de servidor. Debido a esto es que existen dos estados para expresar esta diferencia, *Connecting Client* y *Connecting Server*. Sin embargo, una vez que la conexión es establecida, esta situación es transparente a la capa superior, ya que solo se le notifica que tiene un nuevo *peer* conectado, independiente de la forma en que se llevo a cabo esta conexión.

Las transiciones entre los diversos estados pueden observarse en el siguiente diagrama de estados:

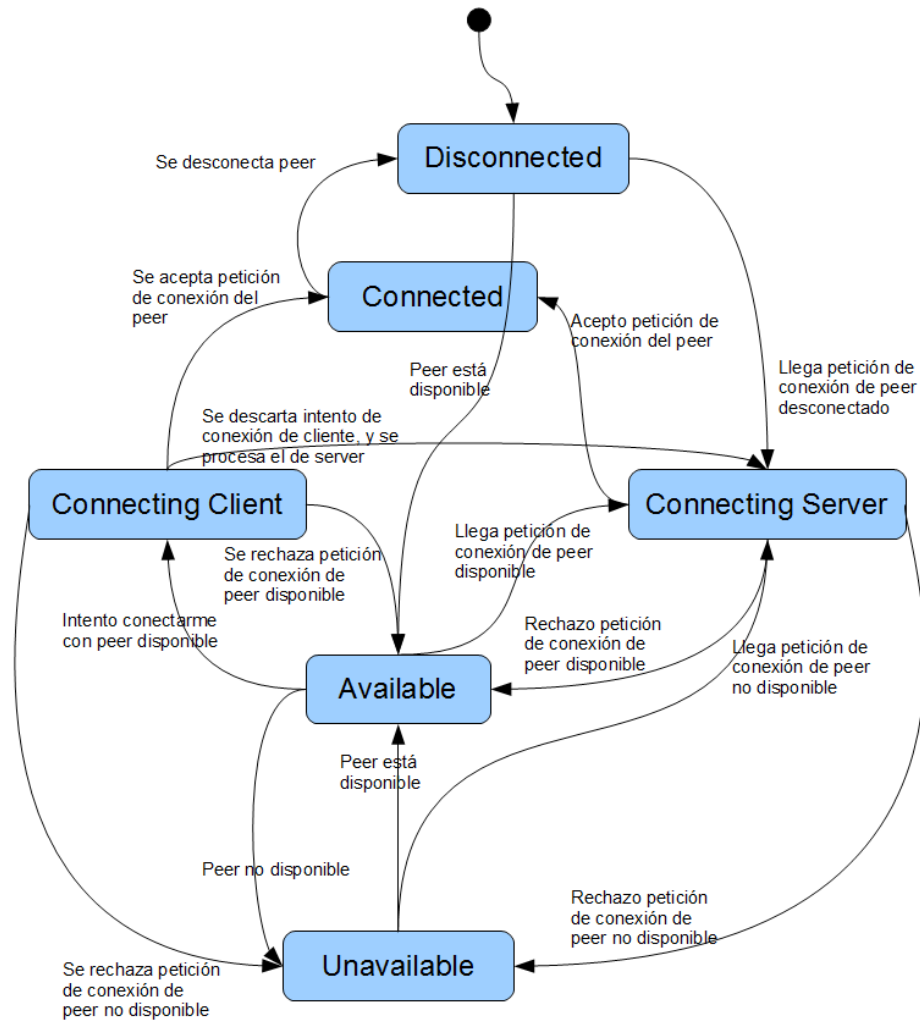


Figura 03: Diagrama de Estados de un *peer*

## 6.5. Establecimiento de conexión entre *peers*

En el siguiente esquema se pueden ver los pasos y los estados de los *peers* al momento de establecer una conexión, que en este caso es aceptada por la capa superior. Previamente se establece una conexión TCP, para luego comenzar con el envío de los mensajes.

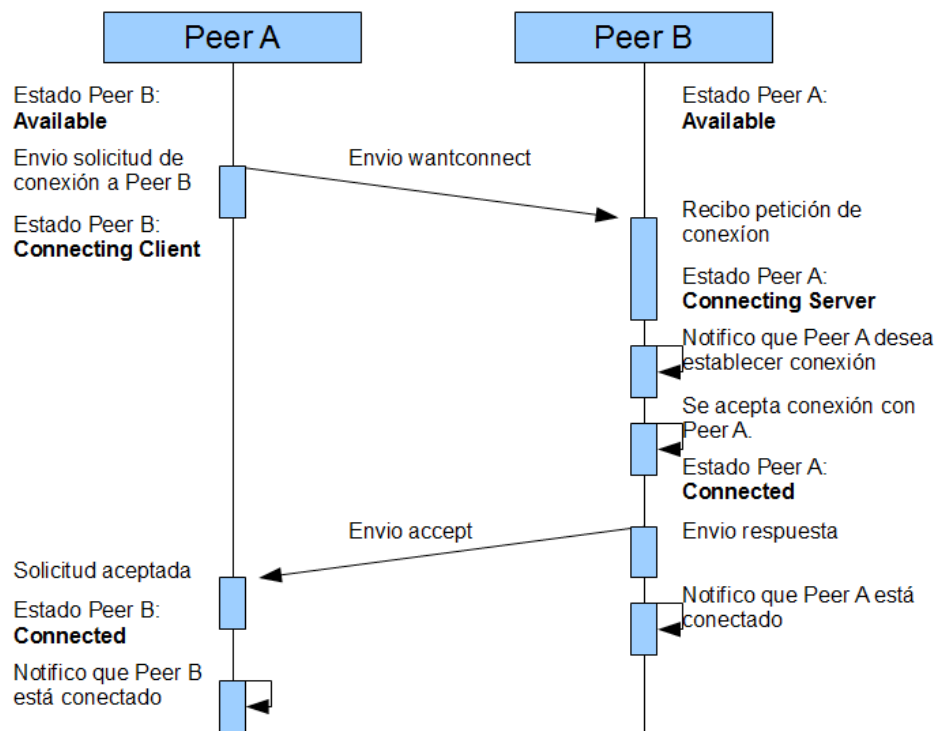


Figura 04: Esquema de establecimiento de conexión. Caso aceptada.



En el siguiente esquema, se observa el caso en que se desea establecer conexión, pero el *peer* no la acepta. Como resultado ambos *peers* no se conectan y mantienen el estado anterior dependiendo si están visibles o no.

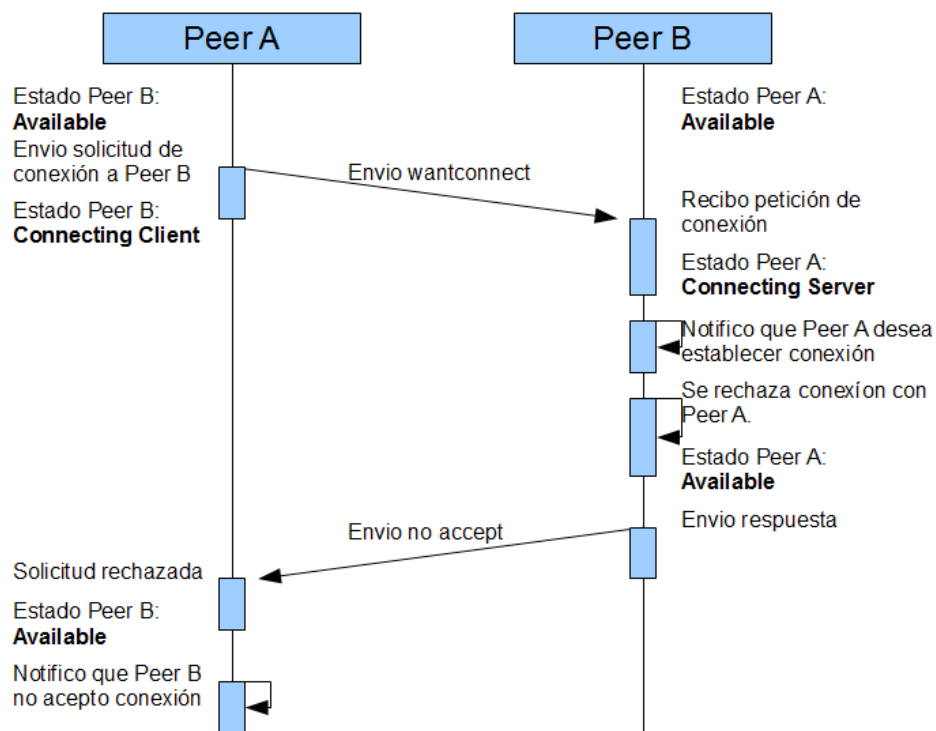


Figura 05: Esquema de establecimiento de conexión. Caso rechazada.

El siguiente caso es especial, ya que ocurre cuando ambos *peers* desean establecer conexión simultáneamente. Al estar en estado *Connecting Client*, significa que ya fue enviada una solicitud de conexión al otro *peer*, y se espera su respuesta. Como se observa en el siguiente esquema, puede darse el caso en que estando en el estado *Connecting Client*, llegue una solicitud de conexión del mismo *peer*.

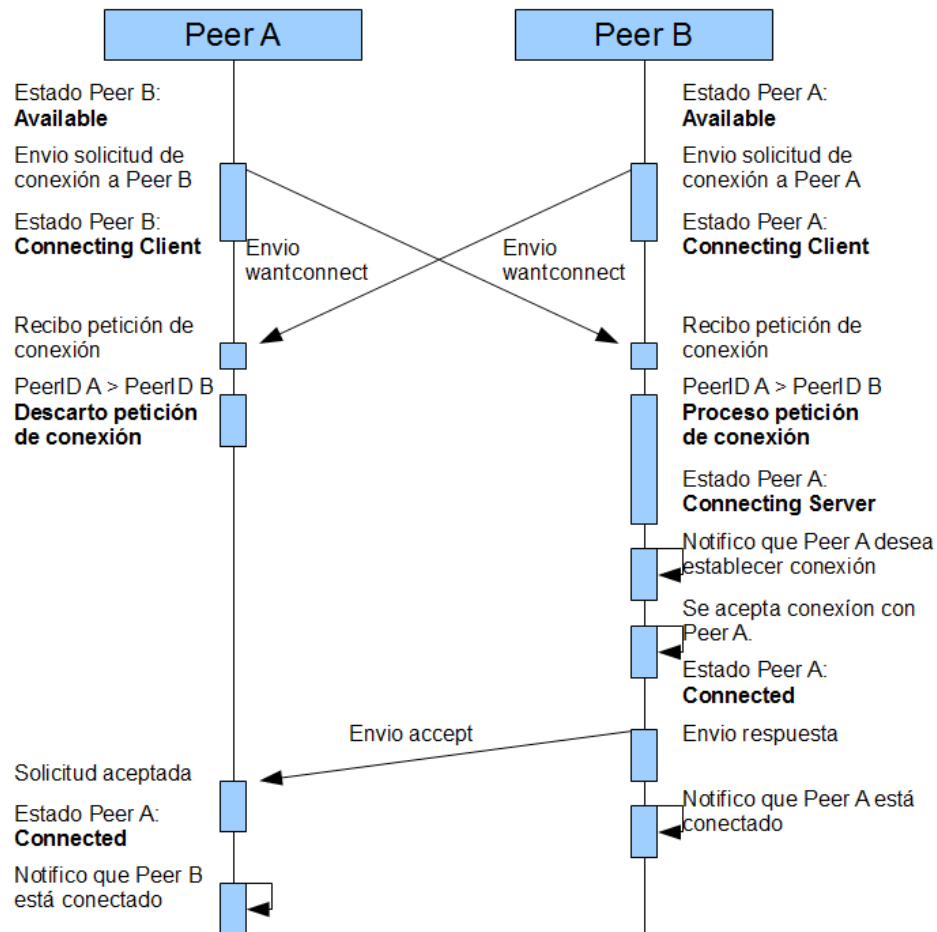


Figura 06: Esquema de establecimiento de conexión simultanea.

Tal situación se da análogamente con el otro *peer*, en donde se puede optar por dos caminos. Uno es descartar la solicitud de conexión recién llegada, y esperar por la solicitud enviada antes, lo que genera que ambos se queden esperando y no se conecten. El otro camino es descartar la solicitud enviada anteriormente y procesar la recién llegada, dando como consecuencia que se generen dos conexiones entre los *peers*, resultado que tampoco se desea.

Para evitar las situaciones anteriormente descritas, se utilizan los peerIDs para determinar que intento de conexión es el que finalmente se procesa, y cuál es el que se descarta. De este modo, se obtiene como resultado sólo una conexión entre los dos *peers*.

Si ocurre la pérdida de la conexión TCP durante el establecimiento de la conexión con un *peer*, dicho *peer* quedará en estado *Disconnected*.

Todas las notificaciones existentes, ya sea cuando se conecta un *peer* o se reciben datos, se realizan utilizando el patrón de diseño *delegate*.

## 6.6. Envío de datos entre *peers*

Una vez que se establece la conexión entre ambos *peers*, estos se quedan en espera hasta que lleguen datos. Al comenzar la transferencia, se envía el tamaño de los datos a enviar, de este modo el *peer* que recibe sabe exactamente cuantos bytes necesita leer para posteriormente notificar la llegada de datos.

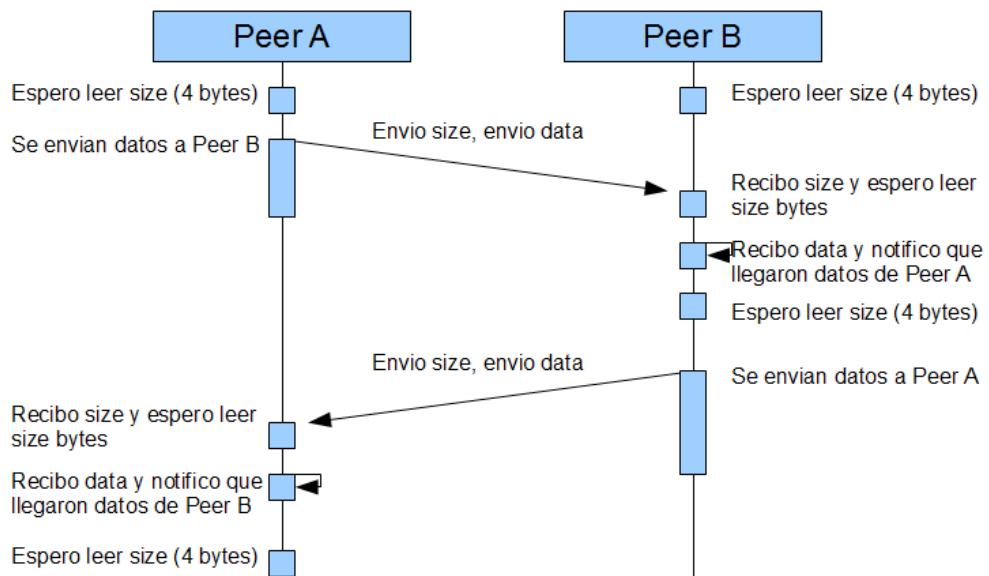


Figura 07: Esquema de envío de datos

El formato de los datos a enviar queda determinado por la capa superior.

## 6.7. Modos de Operación

Dado que un *peer* internamente posee un cliente y un servidor operando al mismo tiempo, también se dejó la alternativa que pueda operar ya sea como cliente o como servidor solamente, por lo que se tienen los siguientes modos de operación.

- Modo *Peer*: Permite recibir conexiones de otros *peer*, además de la posibilidad de solicitar conexiones a otros *peers*. Visibilidad y búsqueda de *peers* activadas.
- Modo *Cliente*: Posee desactivada la visibilidad, pero activada la búsqueda de *peers* disponibles, por lo que puede solicitar conexiones a otros *peers*.
- Modo *Servidor*: Posee desactivada la búsqueda de *peers* disponibles, pero activa la visibilidad, por lo que puede recibir conexión de otros *peers*.



Figura 08: Modo Servidor y Modo Cliente

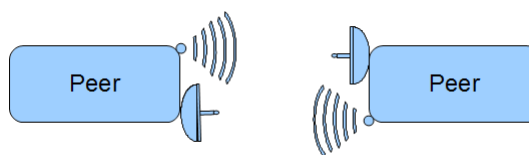


Figura 09: Modo Peer

## 6.8. Protocolos Soportados

Al iniciar el diseño de esta capa de comunicación *P2P*, sólo se tenía contemplado el uso de *TCP/IP* via *Wi-Fi*, pero al poco tiempo se liberó la versión 3.0 del sistema operativo del iPhone, la que incluye el soporte de *Bluetooth* a través del framework *GameKit* [21], además de proveer conectividad *peer to peer*, muy similar a lo diseñado para *Wi-Fi*.

Por esta razón se incluyó el protocolo *Bluetooth*, pero como se detallará más adelante, no posee soporte nativo para la transmisión de archivos, debido al tamaño que estos pueden tener (sobre los 100 kbytes), ya que esta más orientada al envío de mensajes. Sin embargo, se desarrolló un método de envío que permite utilizando *GameKit*, transmitir archivos por *Bluetooth*. Esto está implementado en la clase *P2PSessionBluetooth*.

Para la integración se utilizó una clase abstracta, permitiendo así hacer que la interfaz quedara independiente de la implementación del protocolo.

La clase *TransfersManager* se encarga de entregar métodos adecuados para el envío y recepción de mensajes (NSString) e imágenes (UIImage). Internamente realiza la serialización de estos objetos, y hace uso de *P2PCommunication* para realizar las transferencias. En resumen, entrega una interfaz muy similar a *P2PCommunication*, salvo que posee métodos para transmitir mensajes e imágenes. La razón de esto es que la aplicación a desarrollar se basa en transmisión de fotos del usuario.

La clase abstracta *P2PSession* unifica la interfaz utilizada por los protocolos.

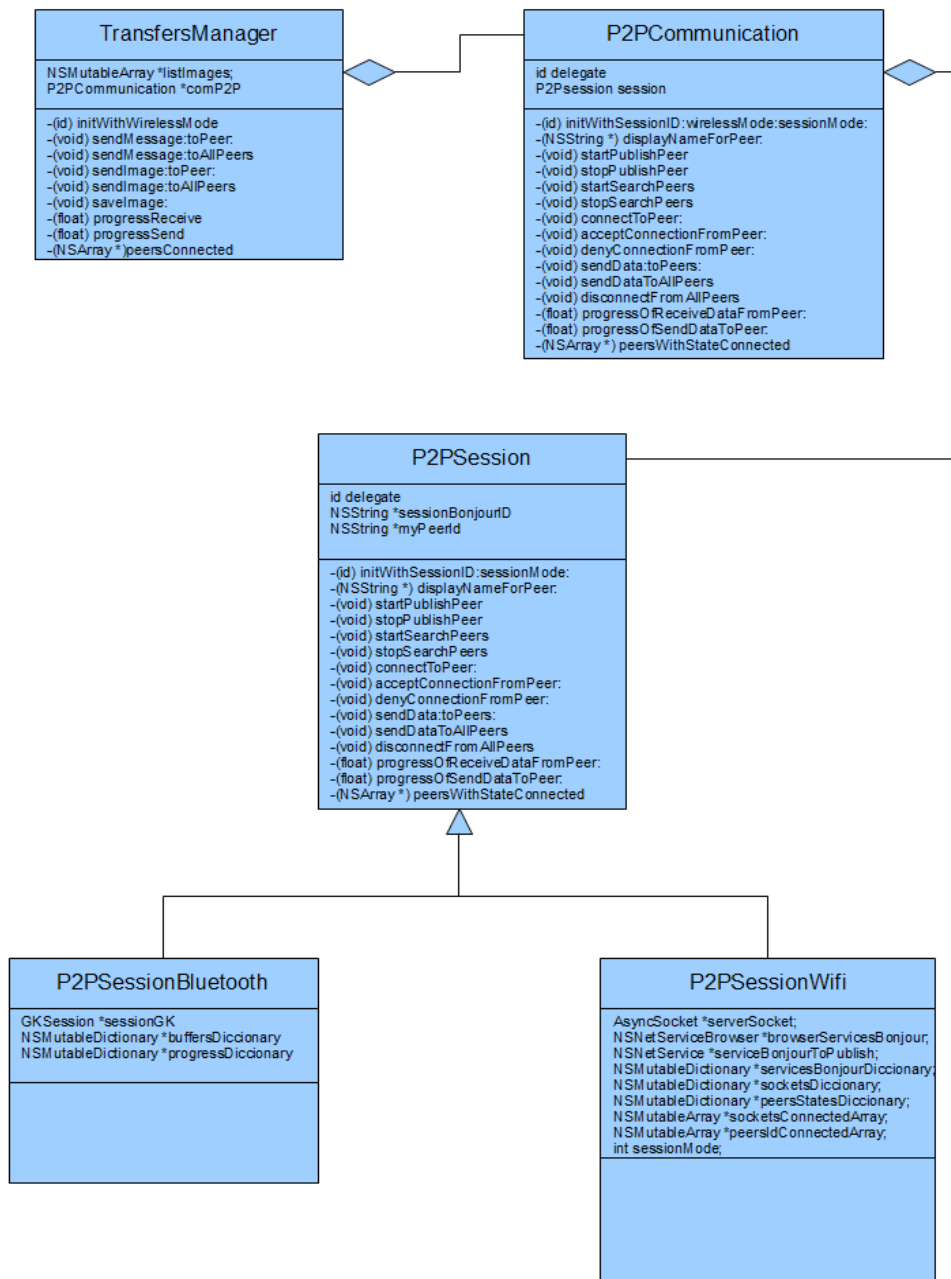


Figura 10: Diagrama de clases de la capa de comunicación diseñada

## 6.9. Aplicación Communication

Esta aplicación que desarrollé, permite probar si la capa de comunicación esta operando correctamente, a modo de prueba de concepto. Además de tener una interfaz muy sencilla, posee las siguientes características:

- Muestra el listado de *peers* conectados.
- Se indica cuando un *peer* se acaba de conectar
- Permite enviar un mensaje predefinido a un *peer* con sólo seleccionarlo en la pantalla. También puede enviar un mensaje simultáneamente a todos los *peers* conectados. Al *peer* receptor se le avisara la llegada del mensaje.
- Botón para desconectarse de todos los *peers*.
- Se indica cuando un *peer* se desconecta.

Al iniciar la aplicación, se muestra un cuadro de opción para que el usuario seleccione el protocolo a utilizar. Una vez realizada esta selección, la capa de comunicación parte en *Modo Peer*, lo que significa que los *peers* serán visibles y podrán buscar *peers* disponibles. En el momento que se detecte un *peer* disponible, inmediatamente se comenzará con el proceso de establecimiento de conexión. A su vez, cuando se notifique que un *peer* desea establecer conexión, esta se aceptará.

Como resultado de esta configuración, todos los *peers* se conectarán entre ellos, formando una red *peer to peer* pura, como se indica en la figura 11.



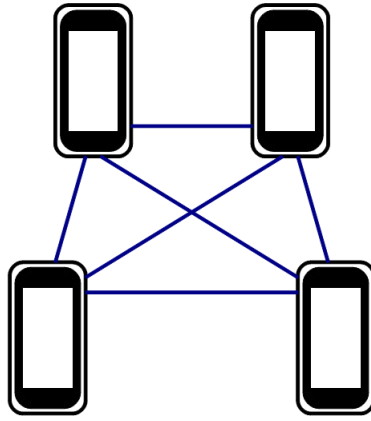


Figura 11: Red *P2P* pura

A medida que los *peers* se conectan o desconectan, automáticamente actualizan el listado que se muestra en pantalla. Esto hace que la aplicación sea bastante dinámica y que los cambios que ocurren dentro de la red se reflejen inmediatamente.

Pantalla de selección de protocolo:



Figura 12: Inicio de aplicación Communication

Iniciando búsqueda, publicación y conexión con otros *peers*. Este proceso se realiza automáticamente.

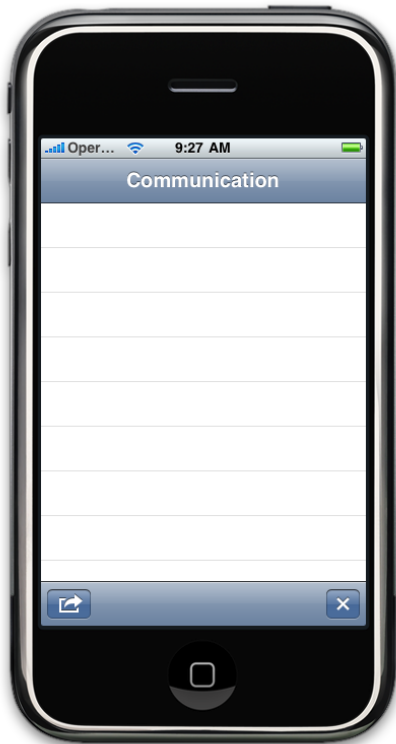


Figura 13: Inicio de aplicación



Figura 14: Conexión con *peers*

Recepción de mensaje de un *peer*, y recepción de mensaje enviado a todos los *peers* conectados. Para enviar un mensaje, basta con presionar el nombre del *peer* que se muestra en pantalla. Para enviar un mensaje a todos los *peers*, se presiona el botón inferior izquierdo.



Figura 15: Inicio de aplicación



Figura 16: Conexión con *peers*

Listado de *peers* conectados y desconexión de un *peer*. Para desconectarse de todos los *peers*, se presiona el botón inferior derecho.



Figura 17: Listado de *peers*



Figura 18: Desconexión de un *peer*

## 6.10. Aplicación CSphotoshare

Esta aplicación que desarrollé tiene como finalidad permitir a los usuarios compartir las fotos que poseen en su álbum de fotos del teléfono. También permite la edición previa de las fotos antes de ser enviadas. Se encuentra disponible en la *App Store* desde su primera versión.

La primera versión de CSphotoshare no contaba con conexiones *peer to peer* propiamente tal, sino que se asemejaba más a un esquema *cliente-servidor*. Al enviar una imagen a otro teléfono, primero se debe resolver el nombre del equipo entregado por *Bonjour*. Una vez que ya se tiene la IP y el puerto, se procede a establecer la conexión TCP. Cuando la conexión esta lista, se manda la imagen y una vez que la transferencia esta completa, la conexión se termina. Si se desea enviar otra imagen, se debe realizar el mismo proceso. En este esquema la aplicación no era capaz de recibir ni enviar imágenes simultáneamente de varios *iPhones*.

Para la segunda versión de esta aplicación, se integro la capa de comunicación propuesta en este trabajo, la que además incluye transmisión por *bluetooth*. Tal como la aplicación Communication, Photoshare esta configurada para formar una red *peer to peer* pura.

Dado el nuevo diseño de la capa de comunicación de la aplicación, esta permite recibir y enviar fotos simultáneamente, recibir fotos de múltiples iPhones, y enviar fotos a todos los equipos al mismo tiempo, lo que permite que dos o más usuarios que estén ejecutando la aplicación puedan compartir sus fotos con mucha facilidad.

Para efectos de su distribución en la App Store, la aplicación se encuentra en idioma ingles y español, el que se determina dependiendo del idioma en que se encuentre el teléfono.

El diseño de la aplicación sigue el patrón *MVC (Modelo Vista Controlador)*. La interfaz de usuario esta determinada por 4 vistas y sus correspondientes 4 controladores, que se describirán a continuación:

**PhotoBrowser:** Esta corresponde a la primera vista que se muestra el usuario al inicial la aplicación. Con el botón *Help* se puede acceder al about y a la vista de *Settings*. El botón *Photos* se accede al álbum de fotos del teléfono, permitiendo al usuario navegar por las fotos que posee en el teléfono. Una vez que selecciono una, se muestra una serie de alternativas, entre enviar la foto, enviar a todos, o continuar con la selección. Cabe destacar que inmediatamente que la aplicación es iniciada, comienza la búsqueda y establecimiento de conexión con otros *iPhones* que estén ejecutando la aplicación.

En la interfaz también se indica que protocolo se esta utilizando actualmente, ya sea *Wi-Fi* o *Bluetooth*.



Figura 19: Pantalla Inicial



Figura 20: Navegador de fotos

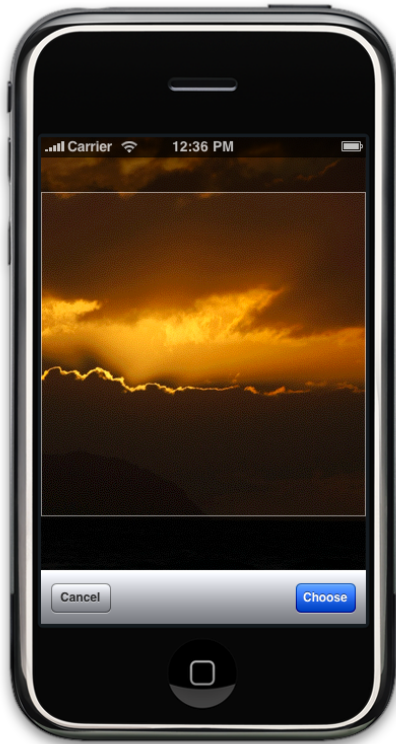


Figura 21: Preview y edición de la imagen

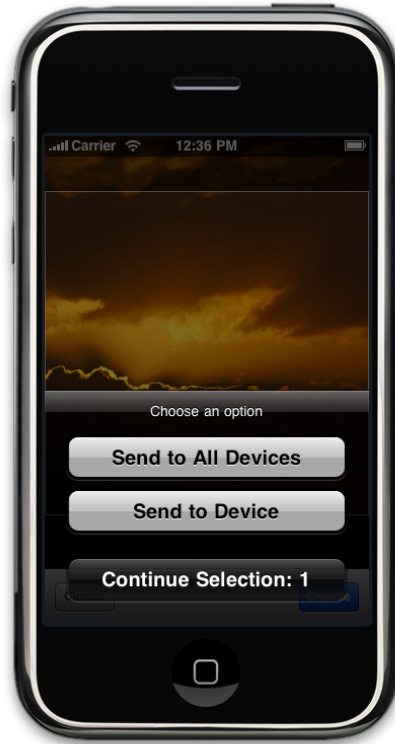


Figura 22: Opciones para enviar

**Settings:** En esta vista se visualizan y establecen las preferencias de la aplicación. Basta con tocar los controles para que estos cambien de valor. Las preferencias se almacenan una vez que se abandona la ventana de *Settings*. Cabe destacar que acá se determina que protocolo utilizará la aplicación. El resto de la aplicación puede consultar los valores de los *settings* sin la necesidad de llamar métodos de este controlador.

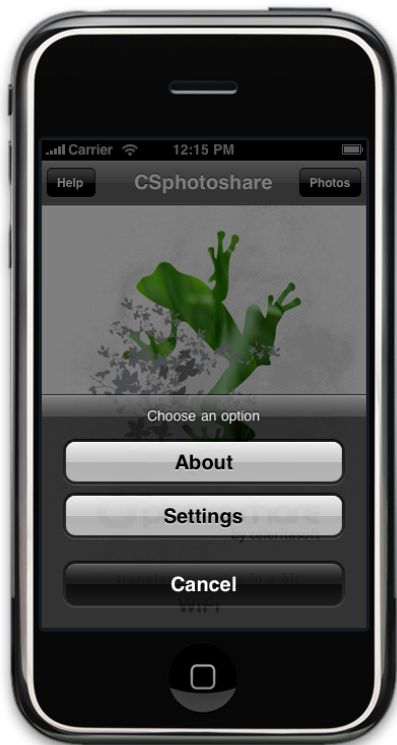


Figura 23: About y Settings

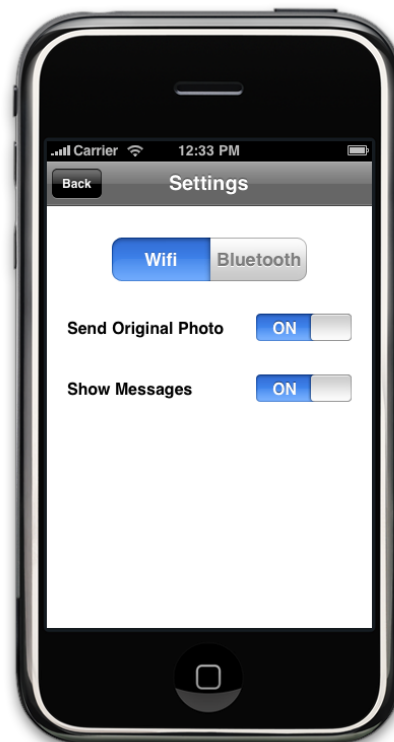


Figura 24: Pantalla Settings



**DeviceBrowser:** En esta vista se muestra el listado de teléfonos que se encuentran conectados. Para enviar la imagen anteriormente seleccionada, basta con elegir a alguno del listado. Inmediatamente se enviará una petición de envío. Si esta es aceptada, el envío es efectuado.



Figura 25: Pantalla DeviceBrowser

**Loading:** Esta vista se despliega cada vez que se este realizando una transferencia de imagen, ya sea al enviar o al recibir. Además de indicar si se trata de un envío o recepción, también se indica el estado de avance, lo que da mayor información al usuario.



Figura 26: Envío de imágenes



Figura 27: Recepción de imágenes

Cabe destacar que la aplicación es capaz de recibir una foto independiente de la pantalla en que se encuentre. Pero antes de efectuar el envío, se le informa al usuario del teléfono receptor si desea recibir una foto. Si acepta, se lleva a cabo la transferencia.

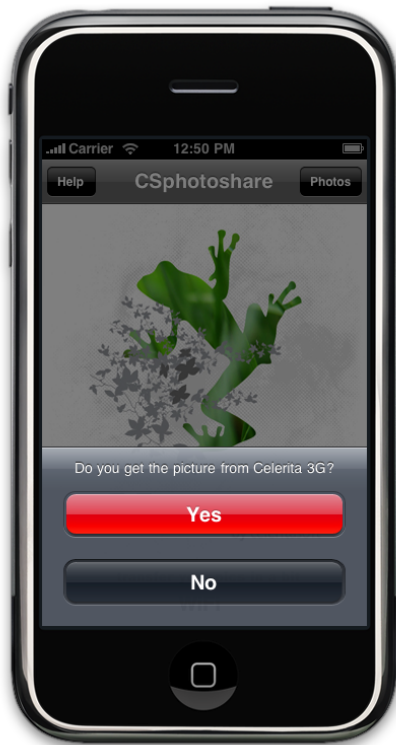


Figura 28: Consulta al usuario

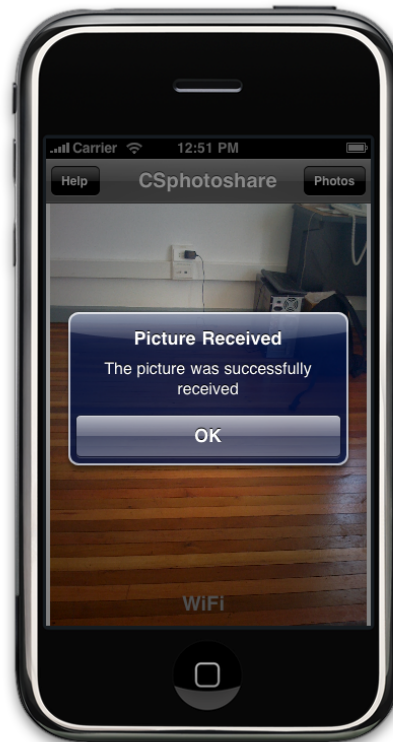


Figura 29: Imagen recibida

Uno de los primeros problemas que se presentaron al utilizar *bluetooth*, es que se tenían problemas de transmisión al intentar enviar fotos, debido al tamaño del archivo a enviar. Para hacer posible el envío de fotos vía *bluetooth*, previamente se divide los datos en paquetes de tamaño 500 bytes aproximadamente, los que son enviados consecutivamente. En el receptor se espera hasta que llegue el último paquete, para luego notificar que el dato fue recibida. De este modo se mejoró la funcionalidad que posee el framework *GameKit* en lo que respecta a transmisión de datos.

En el diagrama de clases de la aplicación se pueden observar las clases de los 4 controladores anteriormente descritos. La clase *SwitchViewsController* se encarga del manejo de las transiciones entre las diversas vistas de la aplicación. *TransfersManager* se encarga de establecer las conexiones y de realizar las transferencias, además de notificar el status de avance de cada una de ellas.

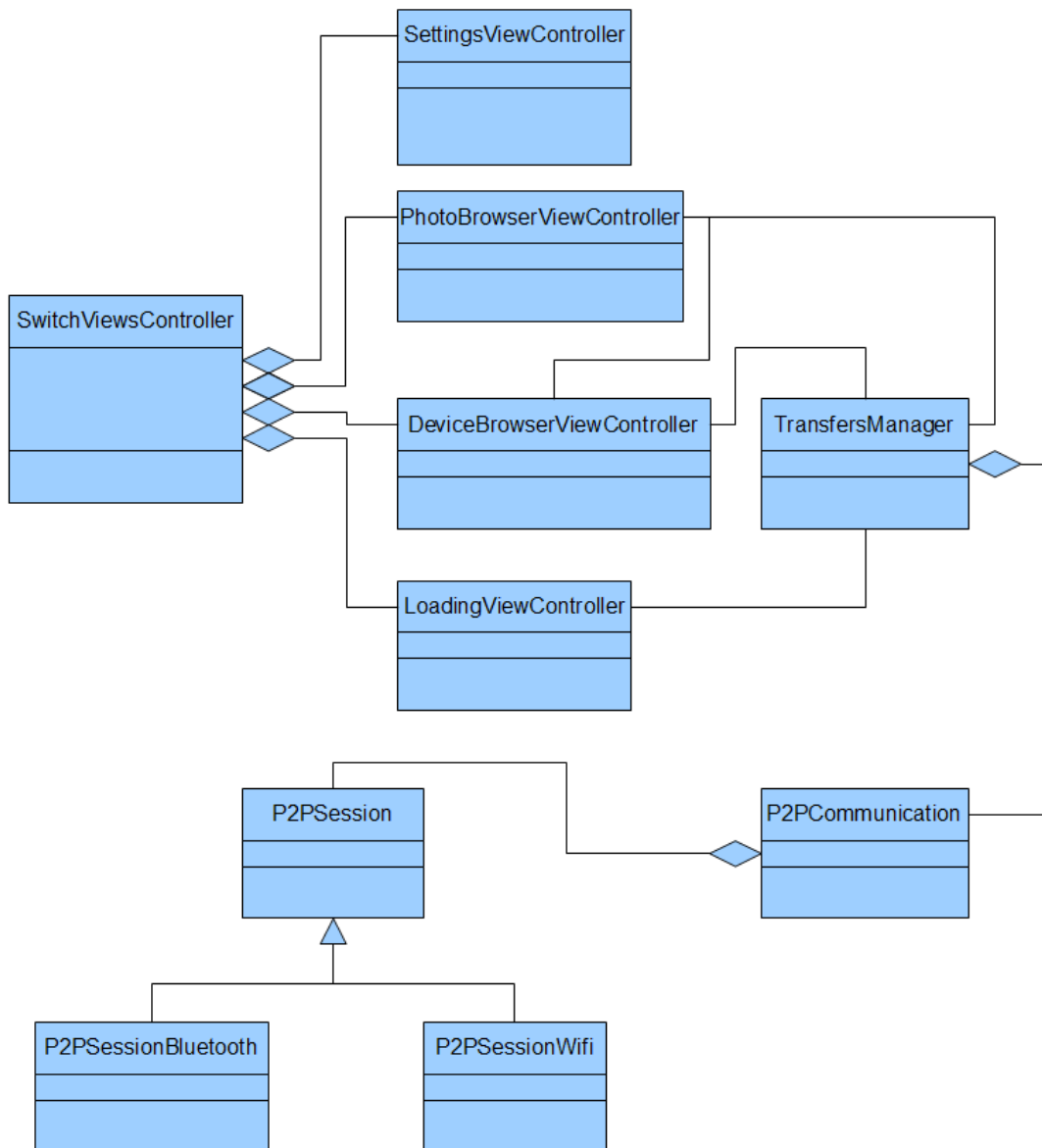


Figura 30: Diagrama de clases de la aplicación CSphotoshare

Como la primera versión de Csphotoshare se desarrollo utilizando directamente sockets, quedan notoriamente en evidencia las ventajas de utilizar la capa de comunicación *peer to peer*:

- En la segunda versión se cuenta con conexiones permanentes, haciendo que las transferencias de fotos sean más rápidas.
- Identificación del usuario que envía la imagen.
- Se pueden realizar envíos simultáneos de fotos a todos los teléfonos que tengan ejecutando la aplicación. También se puede recibir simultáneamente.
- La integración del protocolo *Bluetooth* a la aplicación tuvo un esfuerzo mínimo, ya que se utiliza una única clase para el manejo de la comunicación.
- Existe una clara separación entre la lógica de la aplicación, y el manejo de las conexiones y la transmisión de datos. En la primera versión esta separación es difusa, debido a que se utilizaron sockets directamente.
- Las horas de trabajo para desarrollar la segunda versión fueron menores, ya que mucha funcionalidad de red se movió a la capa de comunicación.

## 7. Conclusiones

Se ha logrado diseñar e implementar un modelo de comunicación *peer to peer*, el que puede ser configurable dependiendo de los requerimientos de la aplicación. Fácilmente se puede desarrollar un cliente, un servidor, o tener la funcionalidad de ambos en el caso de *peer to peer*. Esto le permite al desarrollador enfocarse en la lógica de la aplicación, ahorrando esfuerzo especialmente si la aplicación maneja múltiples conexiones.

Además de realizar una implementación para *Wi-Fi*, se integró la funcionalidad entregada por *GameKit* para el protocolo *Bluetooth*. De este modo, dependiendo del escenario móvil, la aplicación puede ofrecer al usuario el protocolo de comunicación a utilizar. Si no se cuenta con acceso a red *Wi-fi*, los usuarios de la aplicación pueden utilizar *Bluetooth*. En el caso contrario, es preferible utilizar *Wi-Fi*, dado su mayor alcance de señal, mayor tasa de transferencia, y mayor número de conexiones simultáneas que puede manejar.

Los principales inconvenientes nacen con el uso del *Bluetooth*. Primero la limitación con el tamaño de datos a enviar, que afortunadamente se pudo solucionar. También ocurren problemas de conexión al tener más de 3 *peers* conectados. Es de esperar que Apple solucione estos inconvenientes en las próximas versiones del sistema operativo del *iPhone*.

En el caso del uso de *Wi-Fi*, los problemas se originaban con el uso de *Bonjour*, ya que en ciertos momentos no se detectaban a los equipos, o eran detectados siendo que ya no estaban disponibles. De todos modos estos episodios fueron muy poco frecuentes y ocurrían en un solo equipo celular.

A pesar de estos inconvenientes, es factible poder desarrollar aplicaciones colaborativas que posean comunicación punto a punto. Se logró desarrollar una aplicación real para la App Store (*CSphotoshare*), cumpliendo con los estándares de calidad que se exigen para pasar las pruebas de validación y estar disponible para el público.

También se desarrollo una aplicación de ejemplo (*Communication*), que puede apoyar el diseño y construcción de otros sistemas que utilicen esta implementación.

## 7.1. Trabajo Futuro

Una posible mejora al modelo es integrar la implementación de un servidor *Wi-Fi* de modo de establecer conexiones *Ad-Hoc*, logrando así operar en escenarios carentes de infraestructura adecuada (routers inalámbricos o puntos de acceso), siendo una alternativa mucho más conveniente que la utilización de Bluetooth, debido a las limitaciones tanto de alcance, ancho de banda y de conexiones simultáneas que presenta este protocolo.

## 8. Bibliografía

- [1]: C.A. Ellis, S.J. Gibs, G.L. Rein. *Groupware, some issues and experiences*. Communications of the ACM. 2001
- [2]: G. Gerónimo, V. Canseco. *Breve introducción a los sistemas colaborativos*. Temas de Ciencias y Tecnologías. 2002
- [3]: Luis A. Guerrero, David A. Fuller. *A pattern system for the development of collaborative applications*. Information and Software Technology. 2001
- [4]: Luis A. Guerrero, Jose A. Pino, Cesar Collazos. *Mobile support for collaborative work*. Proceedings of 10th International Workshop on Groupware. 2004
- [5]: Luis A. Guerrero, Sergio F. Ochoa, Jose A. Pino, Cesar Collazos. *Selecting computing devices to support mobile collaboration*. Group Decision and Negotiation. 2006
- [6]: Rosa Alarcon, Luis A. Guerrero, Sergio F. Ochoa, Jose A. Pino. *Analysis and design of mobile collaborative applications using contextual elements*. Computing and Informatics. 2006
- [7]: Paredes M., Fernandez I., Ortega M. . *Escritura Colaborativa y PDAs: una Propuesta de Aprendizaje Basada en Resolución de Problemas*. . 2004
- [8]: Luis A. Guerrero, David A. Fuller. *Design patterns for collaborative systems*. Proceedings of 5th International Workshop on Groupware. 1999
- [9]: Sun Microsystems, Inc. *Java ME Technology APIs and Docs*. <http://java.sun.com/javame/reference/apis.jsp>. 2009
- [10]: GSM Association. *GSM Technology*. <http://www.gsmworld.com/technology/index.htm>. 2009
- [11]: Wi-Fi Alliance. *In-depth Wi-Fi Information* . [http://www.wi-fi.org/knowledge\\_center\\_overview.php](http://www.wi-fi.org/knowledge_center_overview.php). 2009
- [12]: Jaap C. Haartsen, Ericsson Radio System B.V. *The Bluetooth radio system*. <https://eprints.kfupm.edu.sa/69017/1/69017.pdf>. 2000



- [13]: Detlef Schoder, Kai Fischbach, Christian Schmitt. *Core Concepts In Peer-To-Peer Networking*. <http://www.idea-group.com/downloads/excerpts/Subramanian01.pdf>. 2005
- [14]: Stephanos Androutsellis, Theotokis and Diomidis Spinellis . *A Survey of Peer-to-Peer Content Distribution Technologies*. ACM Computing Surveys. 2004
- [15]: Apple Inc.. *iPhone Application Programming Guide*..  
<http://developer.apple.com/iphone/library/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/index.html>. 2009
- [16]: Apple Inc.. *Introduction to The Objective-C Programming Language*.  
<http://developer.apple.com/iphone/library/documentation/Cocoa/Conceptual/ObjectiveC/Introduction/introObjectiveC.html>. 2009
- [17]: Apple Inc.. *Introduction to Bonjour Overview*.  
<http://developer.apple.com/mac/library/documentation/Cocoa/Conceptual/NetServices/Introduction.html>. 2006
- [18]: Erik Guttman. *Autoconfiguration for IP Networking: Enabling Local Communication*.  
<http://www.zeroconf.org/w3onwire-zeroconf.pdf>. 2001
- [19]: Apple Inc. *NSNetServices and CFNetServices Programming Guide*.  
<http://developer.apple.com/iphone/library/documentation/Networking/Conceptual/NSNetServiceProgGuide/Introduction.html>. 2009
- [20]: Dustin Voss. *AsyncSocket Documentation*.  
[http://homepage.mac.com/d\\_j\\_v/.Public/AsyncSocket4.3/AsyncSocket%20Documentation.html](http://homepage.mac.com/d_j_v/.Public/AsyncSocket4.3/AsyncSocket%20Documentation.html). 2009
- [21]: Apple Inc.. *Game Kit Programming Guide*.  
[http://developer.apple.com/iphone/library/documentation/NetworkingInternet/Conceptual/GameKit\\_Guide/index.html](http://developer.apple.com/iphone/library/documentation/NetworkingInternet/Conceptual/GameKit_Guide/index.html).
- [22]: Apple Inc.. *iPhone Human Interface Guidelines*.  
<https://developer.apple.com/iphone/library/documentation/UserExperience/Conceptual/MobileHIG/index.html>. 2009
- [23]: Luis A.Guerrero, Sergio F.Ochoa, Oriel A.Herrera, Davis A.Fuller. *Un modelo de comunicacion para aplicaciones colaborativas*. Actas del Workshop de Investigadores en Ciencias de la Computación. 1999

[24]: Alf Inge Wang, Tommy Bjornsgard, Kim Saxlund. *Rapid Application Framework for Mobile Peer to Peer Applications*. Department of Computer and Information Science Norwegian University of Science and Technology Trondheim, Norway. 2007

[25]: César A. Collazos, Rosa Alarcón, Luis A. Guerrero. *An Architectural Model to Support Collaborative Work through Mobile Devices*. XXXI Conferencia Latinoamericana de Informática. 2005

[26]: Andrés Neyem, Sergio Ochoa, José A. Pino, Luis A. Guerrero. *Sharing Information Resources in Mobile Ad-hoc Networks*. Proceedings of 11th International Workshop on Groupware. 2005

# A. ANEXO

## A.1. Código fuente aplicación Communication

### A.1.1. CommunicationViewController.h

```
//
// CommunicationViewController.h
// Communication
//
// Created by Luis Avello on 8/17/09.
// Copyright __MyCompanyName__ 2009. All rights reserved.
//

#import <UIKit/UIKit.h>

@class P2PCommunication;

@interface CommunicationViewController : UIViewController
<UISheetDelegate> {
    IBOutlet UITableView *peersTable;
    NSMutableArray *peersArray;
    P2PCommunication *com;
}
@property (nonatomic, retain) UITableView *peersTable;
@property (nonatomic, retain) NSMutableArray *peersArray;
@property (nonatomic, retain) P2PCommunication *com;

-(IBAction) sendAll:(id)sender;

-(IBAction) disconnectAll:(id)sender;

@end
```

### A.1.2. CommunicationViewController.m

```
//
// CommunicationViewController.m
// Communication
//
// Created by Luis Avello on 8/17/09.
// Copyright __MyCompanyName__ 2009. All rights reserved.
//

#import "CommunicationViewController.h"
#import "P2PCommunication.h"
#import <sys/utsname.h>

@implementation CommunicationViewController

@synthesize peersTable;
@synthesize peersArray;
@synthesize com;
```

```

- (void)viewDidLoad {

    NSLog(@"P2P: Comunicacion Iniciada");
    peersArray = [[NSMutableArray alloc] init];
    UIActionSheet *xactionSheet = [[UIActionSheet alloc]
initWithTitle:@"Seleccione un protocolo" delegate:self
 cancelButtonTitle:nil destructiveButtonTitle:nil otherButtonTitles:nil ];
    [xactionSheet addButtonWithTitle:@"Wifi"];
    [xactionSheet addButtonWithTitle:@"Bluetooth"];
    xactionSheet.cancelButtonIndex = 1;
    xactionSheet.destructiveButtonIndex = 0;
    xactionSheet.tag = 1;
    xactionSheet.actionSheetStyle = UIActionSheetStyleAutomatic;
    [xactionSheet showInView:self.view];
    [xactionSheet release];
    [super viewDidLoad];
}

- (void)actionSheet:(UIActionSheet *)actionSheet didDismissWithButtonIndex:
(NSInteger)buttonIndex {
    if (buttonIndex == 0) { // Wifi
        NSLog(@"MODO WIFI");
        com = [[P2PCommunication alloc]
initWithSessionID:@"p2pXT" wirelessMode:P2PWirelessModeWifi
sessionMode:P2PModePeer];
        com.delegate = self;
    }
    if (buttonIndex == 1 ) { //Bluetooth
        NSLog(@"MODO BLUETOOTH");
        com = [[P2PCommunication alloc]
initWithSessionID:@"p2pXT" wirelessMode:P2PWirelessModeBluetooth
sessionMode:P2PModePeer];
        com.delegate = self;
    }
}

-(IBAction) sendAll:(id)sender {
    NSLog(@"SEND ALL");
    NSData *datos_mensaje = [@"Les debe llegar a todos"
dataUsingEncoding:NSUTF8StringEncoding];
    [com sendDataToAllPeers:datos_mensaje];
}

-(IBAction) disconnectAll:(id)sender {
    NSLog(@"DISCONNECT ALL");
    [com disconnectFromAllPeers];
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
}

- (void)dealloc {
    [super dealloc];
}

```

```

}

/* Delegates */

-(void) communication:(P2PCommunication *)communication peerIsAvailable:
(NSString *)peerID {
    NSLog(@"P2P: Se encontro PeerID: %@", peerID);
    [communication connectToPeer:peerID];
}

-(void) communication:(P2PCommunication *)communication peerIsUnavailable:
(NSString *)peerID {
    NSLog(@"P2P: Se elimino PeerID: %@", peerID);
}

-(void) communication:(P2PCommunication *)communication
didReceiveConnectionRequestFromPeer:(NSString *)peerID {
    NSLog(@"P2P: Se recibio peticion de coneccion de PeerID: %@",
peerID);
    [communication acceptConnectionFromPeer:peerID];
}

-(void) communication:(P2PCommunication *)communication peerIsConnected:
(NSString *)peerID {
    [peersArray addObject:peerID];
    [peersTable reloadData];
    NSLog(@"P2P: Se conecto PeerID: %@", peerID);
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:[com
displayNameForPeer:peerID] message:@"Se ha conectado con Peer" delegate:nil
 cancelButtonTitle:@"OK" otherButtonTitles:nil];
    [alert show];
    [alert release];
}

-(void) communication:(P2PCommunication *)communication peerIsDisconnected:
(NSString *)peerID {
    [peersArray removeObject:peerID];
    [peersTable reloadData];
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:[com
displayNameForPeer:peerID] message:@"El peer se ha desconectado"
delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil];
    [alert show];
    [alert release];
}

-(void) communication:(P2PCommunication *)communication didNotConnectPeer:
(NSString *)peerID {
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:peerID
message:@"No se pudo conectar con el peer" delegate:nil
 cancelButtonTitle:@"OK" otherButtonTitles:nil];
    [alert show];
    [alert release];
}

-(void) communication:(P2PCommunication *)communication didReceiveData:

```

```

(NSData *)data fromPeer:(NSString *)peerID {
    NSLog(@"ME LLEGARON DATOS");

    NSString *contenido = [[NSString alloc] initWithData:data
encoding:NSUTF8StringEncoding];
    NSString *mensaje = [[NSString alloc] initWithFormat:@"Llego
mensaje: %@", contenido];
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:peerID
message:mensaje delegate:nil cancelButtonTitle:@"OK"
otherButtonTitles:nil];
        [alert show];
        [alert release];
}

-(void) communication:(P2PCommunication *)communication connectionError:
(NSString *)error fromPeer:(NSString *)peerID {
    NSLog(@"ERROR EN COMMUNICATION peer: %@ error: %@", peerID,
error);
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:
(NSInteger)section {
    return [self.peersArray count];
}

- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    static NSString *SimpleTableIdentifier = @"SimpleTableIdentifier";
    UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:SimpleTableIdentifier];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc]
initWithFrame:CGRectMake(0,0,0,0)

                reuseIdentifier:SimpleTableIdentifier ] autorelease];
    }
    NSUInteger row = [indexPath row];
    cell.textLabel.text = [com displayNameForPeer:[peersArray
objectAtIndex:row]];
    return cell;
}

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:
(NSIndexPath *)indexPath {
    NSUInteger row = [indexPath row];
    NSString *peerID = [peersArray objectAtIndex:row];
    NSLog(@"Se selecciono el peer: %@", peerID);
    NSData *datos_mensaje = [@"Hello peer iphone!!!"
dataUsingEncoding:NSUTF8StringEncoding];
    [com sendData:datos_mensaje toPeer:peerID];
    NSLog(@"SE MANDARON DATOS");
}

@end

```

### A.1.3. P2PCommunication.h

```

//
// P2PCommunication.h
// Communication
//
// Created by Luis Avello on 8/17/09.
// Copyright 2009 __MyCompanyName__. All rights reserved.
//

#import <Foundation/Foundation.h>

@class P2PSession;

typedef enum {
    P2PWirelessModeWifi,
    P2PWirelessModeBluetooth,
} P2PWirelessMode;

typedef enum {
    P2PModeClient,
    P2PModeServer,
    P2PModePeer,
} P2PMode;

@interface P2PCommunication : NSObject {
    id delegate;
    P2PSession *session;
}

@property (nonatomic, assign) id delegate;
@property (nonatomic, retain) P2PSession *session;

/* Inicio */
-(id)initWithSessionID:(NSString *)sessionID wirelessMode:
(P2PWirelessMode)modeWireless sessionMode:(P2PMode)modeSession;

/* Retorna el nombre asociado a un PeerID */
-(NSString *) displayNameForPeer:(NSString *)peerID;

/* Inicia publicacion, Peer queda visible */
-(void) startPublishPeer;

/* Detiene publicacion, Peer no queda visible */
-(void) stopPublishPeer;

/* Inicia la busqueda de peers */
-(void) startSearchPeers;

/* Detiene la busqueda de peers */
-(void) stopSearchPeers;

/* Conectase con un peer */
-(void) connectToPeer:(NSString *)peerID;

/* Aceptar la coneccion del peer */
-(void) acceptConnectionFromPeer:(NSString *)peerID;

/* Rechazar la coneccion del peer */
-(void) denyConnectionFromPeer:(NSString *)peerID;

```

```

/* Enviar datos a un peer */
-(void) sendData:(NSData *)data toPeer:(NSString *)peerID;

/* Enviar datos a peers conectados */
-(void) sendData:(NSData *)data toPeers:(NSArray *)peers;

/* Enviar datos a todos los peers conectados */
-(void) sendDataToAllPeers:(NSData *)data;

/* Me desconecto de todos los peers */
-(void) disconnectFromAllPeers;

/* Progreso de recepcion de datos del peer */
-(float) progressOfReceiveDataFromPeer:(NSString *)peerID;

/* Progreso de envio de datos a un peer */
-(float) progressOfSendDataToPeer:(NSString *)peerID;

/* Listado de peers conectados */
-(NSArray *) peersWithStateConnected;

@end

@interface NSObject (P2PCommunicationDelegate)

/* El peer esta disponible */
-(void) communication:(P2PCommunication *)communication peerIsAvailable:
(NSString *)peerID;

/* El peer no esta disponible */
-(void) communication:(P2PCommunication *)communication peerIsUnavailable:
(NSString *)peerID;

/* Llego peticion de coneccion de un peer */
-(void) communication:(P2PCommunication *)communication
didReceiveConnectionRequestFromPeer:(NSString *)peerID;

/* El peer se ha conectado */
-(void) communication:(P2PCommunication *)communication peerIsConnected:
(NSString *)peerID;

/* El peer se ha desconectado */
-(void) communication:(P2PCommunication *)communication peerIsDisconnected:
(NSString *)peerID;

/* No se pudo conectar con peer */
-(void) communication:(P2PCommunication *)communication didNotConnectPeer:
(NSString *)peerID;

/* Se han recibido datos */
-(void) communication:(P2PCommunication *)communication didReceiveData:
(NSData *)data fromPeer:(NSString *)peerID;

/* Error en la comunicacion con peer */
-(void) communication:(P2PCommunication *)communication connectionError:
(NSString *)error fromPeer:(NSString *)peerID;

@end

```

#### A.1.4. P2PCommunication.m



```

//
// P2PCommunication.m
// Communication
//
// Created by Luis Avello on 8/17/09.
// Copyright 2009 __MyCompanyName__. All rights reserved.
//

#import "P2PCommunication.h"
#import "P2PConstants.h"
#import "P2PSession.h"
#import "P2PSessionWifi.h"
#import "P2PSessionBluetooth.h"

@implementation P2PCommunication

@synthesize delegate;
@synthesize session;

/* Inicio */
-(id)initWithSessionID:(NSString *)sessionID wirelessMode:
(P2PWirelessMode)modeWireless sessionMode:(P2PMode)modeSession {
    self = [super init];
    if ( self != nil ) {
        if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION init");
        switch (modeWireless) {
            case P2PWirelessModeWifi:

                switch (modeSession) {
                    case P2PModeClient:
                        session =
[[P2PSessionWifi alloc] initWithSessionID:sessionID
sessionMode:P2PSessionModeClient];
                        break;
                    case P2PModeServer:
                        session =
[[P2PSessionWifi alloc] initWithSessionID:sessionID
sessionMode:P2PSessionModeServer];
                        break;
                    case P2PModePeer:
                        session =
[[P2PSessionWifi alloc] initWithSessionID:sessionID
sessionMode:P2PSessionModePeer];
                        break;
                    default:
                        if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION SESSION MODE
ERROR: Error al inicializar");
                        break;
                }
                break;
            case P2PWirelessModeBluetooth:
                switch (modeSession) {
                    case P2PModeClient:
                        session =
[[P2PSessionBluetooth alloc] initWithSessionID:sessionID
sessionMode:P2PSessionModeClient];
                        break;
                    case P2PModeServer:
                        session =

```

```

[[P2PSessionBluetooth alloc] initWithSessionID:sessionID
sessionMode:P2PSessionModeServer];

                                break;
                                case P2PModePeer:
                                session =
[[P2PSessionBluetooth alloc] initWithSessionID:sessionID
sessionMode:P2PSessionModePeer];

                                break;
                                default:

                                if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION SESSION MODE
ERROR: Error al inicializar");

                                break;

                                }
                                break;
                                default:
                                if(DEBUG_COMMUNICATION)
NSLog(@"P2PCOMMUNICATION WIRELESS MODE ERROR: Error al inicializar");
                                break;
                                }
                                session.delegate = self;
                                }
                                return self;
}

/* Retorna el nombre asociado a un PeerID */
-(NSString *) displayNameForPeer:(NSString *)peerID {
    NSString *peerName = [session displayNameForPeer:peerID];
    if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION displayNameForPeer
id: %@ name: %@", peerID, peerName);
    return peerName;
}

/* Inicia publicacion, Peer queda visible */
-(void) startPublishPeer {
    if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
startPublishPeer");
    [session startPublishPeer];
}

/* Detiene publicacion, Peer no queda visible */
-(void) stopPublishPeer {
    if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION stopPublishPeer");
    [session stopPublishPeer];
}

/* Inicia la busqueda de peers */
-(void) startSearchPeers {
    if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
startSearchPeers");
    [session startSearchPeers];
}

/* Detiene la busqueda de peers */
-(void) stopSearchPeers {
    if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION stopSearchPeers");
}

```

```

        [session stopSearchPeers];
    }

    /* Conectase con un peer */
    -(void) connectToPeer:(NSString *)peerID {
        if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION connectToPeer:
%@", peerID);
        [session connectToPeer:peerID];
    }

    /* Aceptar la coneccion del peer */
    -(void) acceptConnectionFromPeer:(NSString *)peerID {
        if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
acceptConnectionFromPeer: %@", peerID);
        [session acceptConnectionFromPeer:peerID];
    }

    /* Rechazar la coneccion del peer */
    -(void) denyConnectionFromPeer:(NSString *)peerID {
        if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
denyConnectionFromPeer: %@", peerID);
        [session denyConnectionFromPeer:peerID];
    }

    /* Enviar datos a un peer */
    -(void) sendData:(NSData *)data toPeer:(NSString *)peerID {
        if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION sendDatatoPeer");
        NSArray *peers = [NSArray arrayWithObject:peerID];
        [session sendData:data toPeers:peers];
    }

    /* Enviar datos a peers conectados */
    -(void) sendData:(NSData *)data toPeers:(NSArray *)peers {
        if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION sendDatatoPeers");
        [session sendData:data toPeers:peers];
    }

    /* Enviar datos a todos los peers conectados */
    -(void) sendDataToAllPeers:(NSData *)data {
        if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
sendDataToAllPeers");
        [session sendDataToAllPeers:data];
    }

    /* Me desconecto de todos los peers */
    -(void) disconnectFromAllPeers {
        if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
disconnectFromAllPeers");
        [session disconnectFromAllPeers];
    }

    /* Progreso de recepcion de datos del peer */
    -(float) progressOfReceiveDataFromPeer:(NSString *)peerID {

```

```

        if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
progressOfReceiveDataFromPeer: %@", peerID);
        float progress = [session progressOfReceiveDataFromPeer:peerID];
        return progress;
    }

    /* Progreso de envio de datos a un peer */
    -(float) progressOfSendDataToPeer:(NSString *)peerID {
        if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
progressOfSendDataToPeer: %@", peerID);
        float progress = [session progressOfSendDataToPeer:peerID];
        return progress;
    }

    /* Listado de peers conectados */
    -(NSArray *) peersWithStateConnected {
        if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
peersWithStateConnected");
        NSArray *list = [session peersWithStateConnected];
        return list;
    }

    /* Liberar memoria */
    -(void) dealloc {
        if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION DEALLOC");
        [session release];
        [super dealloc];
    }

    /* DELEGATES P2PSESSION */

    /* El peer esta disponible */
    -(void) session:(P2PSession *)session peerIsAvailable:(NSString *)peerID {
        if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
session:peerIsAvailable: %@", peerID);
        if ( delegate && [delegate
respondsToSelector:@selector(communication:peerIsAvailable:)] ) {
            [delegate communication:self peerIsAvailable:peerID];
        }
        else {
            if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
communication:peerIsAvailable: FALTA DELEGATE");
        }
    }

    /* El peer no esta disponible */
    -(void) session:(P2PSession *)session peerIsUnavailable:(NSString *)peerID
    {
        if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
session:peerIsUnavailable: %@", peerID);
        if ( delegate && [delegate
respondsToSelector:@selector(communication:peerIsUnavailable:)] ) {
            [delegate communication:self peerIsUnavailable:peerID];
        }
    }

```

```

    }
    else {
        if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
communication:peerIsUnavailable: FALTA DELEGATE");
    }
}

/* Llego peticion de coneccion de un peer */
-(void) session:(P2PSession *)session didReceiveConnectionRequestFromPeer:
(NSString *)peerID {
    if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
didReceiveConnectionRequestFromPeer: %@", peerID);
    if ( delegate && [delegate
respondsToSelector:@selector(communication:didReceiveConnectionRequestFromP
eer:)] ) {
        [delegate communication:self
didReceiveConnectionRequestFromPeer:peerID];
    }
    else {
        if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
communication:didReceiveConnectionRequestFromPeer: FALTA DELEGATE");
    }
}

/* El peer se ha conectado */
-(void) session:(P2PSession *)session peerIsConnected:(NSString *)peerID {
    if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION peerIsConnected:
%@", peerID);
    if ( delegate && [delegate
respondsToSelector:@selector(communication:peerIsConnected:)] ) {
        [delegate communication:self peerIsConnected:peerID];
    }
    else {
        if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
communication:peerIsConnected: FALTA DELEGATE");
    }
}

/* El peer se ha desconectado */
-(void) session:(P2PSession *)session peerIsDisconnected:(NSString *)peerID
{
    if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
peerIsDisconnected: %@", peerID);
    if ( delegate && [delegate
respondsToSelector:@selector(communication:peerIsDisconnected:)] ) {
        [delegate communication:self peerIsDisconnected:peerID];
    }
    else {
        if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
communication:peerIsDisconnected: FALTA DELEGATE");
    }
}

/* No se ha podido conectar con el peer */
-(void) session:(P2PSession *)session didNotConnectPeer:(NSString *)peerID
{
    if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION didNotConnectPeer:

```

```

%@", peerID);
    if ( delegate && [delegate
respondsToSelector:@selector(communication:didNotConnectPeer:)] ) {
        [delegate communication:self didNotConnectPeer:peerID];
    }
    else {
        if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
communication:didNotConnectPeer: FALTA DELEGATE");
    }
}

/* Se han recibido datos */
-(void) session:(P2PSession *)session didReceiveData:(NSData *)data
fromPeer:(NSString *)peerID {
    if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
didReceiveDatafromPeer: %@", peerID);
    if ( delegate && [delegate
respondsToSelector:@selector(communication:didReceiveData:fromPeer:)] ) {
        [delegate communication:self didReceiveData:data
fromPeer:peerID];
    }
    else {
        if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
communication:didReceiveData:fromPeer: FALTA DELEGATE");
    }
}

/* Error en la comunicacion con peer */
-(void) session:(P2PSession *)session connectionError:(NSString *)error
fromPeer:(NSString *)peerID {
    if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
connectionError:fromPeer: %@", peerID);
    if ( delegate && [delegate
respondsToSelector:@selector(communication:connectionError:fromPeer:)] )
{
        [delegate communication:self connectionError:error
fromPeer:peerID];
    }
    else {
        if(DEBUG_COMMUNICATION) NSLog(@"P2PCOMMUNICATION
communication:connectionError:fromPeer: FALTA DELEGATE");
    }
}

@end

```

