

FRAMEWORK DE COMUNICACIONES Y CONTEXTO PARA LA CONSTRUCCIÓN DE JUEGOS MULTIUSUARIO

Andrés Felipe Castaño Henao

(acastan4@eafit.edu.co)

Juan David Hincapié Ramos

(jhincap3@eafit.edu.co)

Asesor

Edwin Nelson Montoya Múnera

Departamento de Ingeniería de Sistemas

Universidad EAFIT

Medellín – Antioquia

2007

TABLA DE CONTENIDO

RESUMEN.....	4
ANOTACION.....	5
Agradecimientos.....	5
1.INTRODUCCION.....	6
1.1. Motivación.....	7
1.2. Definición del Problema.....	8
1.3. Limitación del Alcance.....	9
1.4. Guía al Lector.....	9
2. MARCO TEORICO.....	11
2.1. Conceptos Generales.....	11
2.1.1. Juegos Multiusuario.....	12
2.1.2. Contexto.....	16
2.1.3. Sistemas P2P.....	19
2.1.4. Tecnologías de Comunicaciones Móviles	24
2.1.5. Framework.....	32
2.1.6. Protocolos y Especificación de Protocolos.....	36
2.2. Estado del Arte.....	41
2.2.1. Peer2ME (REFERENCIA).....	41
2.2.2. JSR 259.....	42
2.2.3. JXTA y JXME.....	43
3. GA P2P NETWORK FRAMEWORK.....	44
3.1. Requerimientos.....	44
3.1.1. Requerimientos Funcionales.....	45
3.1.2. Requerimientos No Funcionales.....	46
3.2. Diseño del Framework.....	46
3.2.1. Diagrama Conceptual.....	47
3.2.2. Arquitectura de Alto Nivel.....	49
3.2.3. Consideraciones de Diseño.....	51
3.2.4. Patrones de Diseño.....	55
3.2.5. Servicios del Framework.....	58
3.2.6. Componentes Estructurales del Framework.....	76
3.2.6. Relaciones Entre Servicios.....	77
3.2.7. Protocolos del Framework.....	81
3.3. Implementación en J2ME con Bluetooth.....	96
3.3.1. Requerimientos Funcionales Cubiertos.....	96
3.3.2. Estadísticas del Código.....	97
3.3.3. Ejemplos de Código.....	100
3.3.4. Entorno de Desarrollo.....	106
5. GUIA PARA EL DESARROLLADOR.....	107
5.1. Validación del Framework Mediante Un Ejemplo.....	109
5.1.1. Mi primer MIDlet con el GA P2P Network Framework.....	110

5.1.2. Utilización de los Servicios.....	112
5.1.3. Envío de Mensajes.....	113
5.1.4. Ejecución.....	113
5.2 Comparativo entre el desarrollo de juegos con este Framework y sin el Framework.....	113
6. DISCUSION (o conclusiones?).....	120
6.1. Problemas Encontrados.....	120
6.2. Conclusiones.....	121
6.3. Trabajo Futuro.....	122
7. REFERENCIAS.....	124

RESUMEN

Este documento presenta el API de juegos para desarrollo de juegos P2P Multiusuario (*Game API Peer to Peer Network Framework* o *GA P2P Network Framework*), es decir, un framework para el desarrollo de juegos multiusuario. Los principales productos del framework son la especificación y el diseño del framework, los cuales, son independientes de la plataforma de desarrollo y la tecnología de comunicaciones que se implementó.

La plataforma de desarrollo usada para la implementación del framework fue Java 2 Micro Edition (J2ME) y la tecnología de comunicaciones fue Bluetooth. Este framework cumple con todos los requisitos necesarios para el desarrollo de este tipo de aplicaciones, algunos de estos requisitos son el descubrimiento de sesiones de juego, el manejo de grupo de peers (notificación de unión y salida de un peer hacia o del grupo), manejo de la continuidad de la sesión de la aplicación en su entorno, el paso de mensajes confiables entre peers, la recuperación de la sesión de la aplicación y la sincronización de esta.

Por último, cabe resaltar que el framework hace un manejo eficiente de los recursos (memoria, tamaño del binario), y lo más importante es que es de fácil utilización a la hora de desarrollar una aplicación o juego multiusuario.

ANOTACION

Este es el proyecto de grado que Andrés Felipe Castaño Henao y Juan David Hincapié Ramos realizado desde Enero de 2006 hasta Enero de 2007. Este proyecto de grado se hace dentro del marco del grupo de investigación en redes y sistemas distribuidos de la Universidad EAFIT (GIRSD EAFIT) y aspira a ser la base para trabajos futuros.

Agradecimientos

Queremos agradecer a Edwin Montoya en su papel como asesor y líder de este proyecto, por compartir su experiencia e invaluable conocimiento al desarrollo del mismo.

Igualmente, queremos agradecer a todos aquellos que sufrieron por nuestra ausencia en estas largas jornadas de trabajo, a nuestras novias Diana Carolina Arcila Naranjo y Pernille Christensen y por supuesto, a nuestras familias.

Medellín y Copenhague, Enero 2007.

1. INTRODUCCION

El desarrollo de los dispositivos móviles se ha visto acelerado por el mejoramiento de los componentes de hardware, la aparición de nuevas tecnologías y el aumento del acceso a los mismos por el público en general dado sus bajos precios. Entre estos podemos destacar dispositivos como los asistentes personales digitales (PDA) y los teléfonos celulares, los cuales cuentan cada vez con más servicios como conexiones WiFi, conexiones Bluetooth, redes de datos celulares de alta velocidad, pantallas de alta resolución, sistemas operativos de alto desempeño, etc. Existiendo tasas de penetración hasta del 109% y 108% en países como Suecia e Italia respectivamente y tendencia a cubrimientos similares en otros países [13].

A su vez, debemos destacar cómo las tecnologías de comunicaciones inalámbricas también han venido aumentando sus capacidades, tasas de transferencia y sus rangos de cobertura, a la vez que se disminuyen los costos de los componentes de hardware haciendo que dichas tecnologías sean más accesibles [14]. Encontramos entonces propuestas de comunicación como WiFi, Bluetooth, WiMax, GSM/GPRS, UMTS y muchas más.

Las innovaciones en los campos de los dispositivos móviles y las redes inalámbricas han conducido a investigaciones sobre el desarrollo de nuevos tipos de aplicaciones dados los nuevos escenarios. Resaltan entonces campos como la "Pervasive Computing"¹ o "Ubiquitous Computing", los cuales se concentran en utilizar todos estos dispositivos no convencionales, móviles y con conectividad de algún tipo, para la prestación de nuevos servicios y aplicaciones [6].

Entre los impulsores de esta nueva área de investigación y desarrollo de aplicaciones, tenemos una vez más, las aplicaciones de entretenimiento a la vanguardia. Esto lo podemos ver en aplicaciones de descarga de música a dispositivos móviles, compartimiento de recursos, elementos de personalización como wallpapers y ringtones y principalmente, son los juegos los pioneros en hacer uso extensivo de las nuevas posibilidades que brindan estos dispositivos [15].

Esta industria del contenido para móviles, especialmente el desarrollo de juegos ha crecido desde la primera aparición de "Snake" hasta juegos 3D en los últimos meses, proporcionando los negocios para impulsar el avance de los dispositivos y sus capacidades de procesamiento, interactividad y conectividad. Los primeros juegos móviles mono-usuario fueron desarrollados para el "Casual Gamer"²,

¹Pervasive Computing es la utilización de muchos dispositivos de computación pequeños, en entornos de usuario, ya sea en su casa u oficina (Esta tendencia también es conocida como Ubiquitous Computing).

²Casual Gamer denota aquellos jugadores de juegos de video que no dedican gran cantidad de

dejando afuera a los “Hardcore Gamers”³ debido a la poca capacidad de procesamiento e Figurate de los primeros dispositivos. La gran variedad de dispositivos móviles ahora disponible, *su gran capacidad de procesamiento, presentación y sus nuevas interfaces de comunicaciones* han traído a este campo a los “Hardcore Gamers” los cuales buscan explotar las características de movilidad, ubicuidad y al mismo tiempo no perder las facultades que este tipo de usuario tiene en otras consolas; por ejemplo, los entornos multiusuario.

La construcción de juegos multiusuario en estos nuevos escenarios tecnológicos presenta retos. En algunos casos existirá la disponibilidad de conectarse a Internet con canales de alta velocidad y hacer uso de servidores centrales para el procesamiento [10], en otras ocasiones cierta cantidad de dispositivos estarán aislados de la red central y sin embargo, poseerán la capacidad de procesamiento necesaria para establecer sesiones de juego multiusuario.

Existen también nuevas capacidades en los dispositivos y tecnologías involucradas. *Nos interesan especialmente aquellas que permiten a los dispositivos obtener información acerca del contexto*⁴ en un momento dado, que influye en las condiciones de juego y del jugador.

Debido a los nuevos retos planteados es importante la construcción de frameworks para acelerar el desarrollo de aplicaciones que hagan uso de estos nuevos escenarios tecnológicos (comunicaciones y contexto). En un principio impulsando el desarrollo de juegos y aplicaciones de entretenimiento, lo que luego, por la misma inercia del mercado, resultará en aplicaciones empresariales, educativas, de seguridad, entre otras, y no sólo restringido a dispositivos móviles⁵.

1.1. Motivación

Gran cantidad de prototipos de aplicaciones y servicios se han realizado para estos nuevos escenarios tecnológicos y estos han demostrado el potencial de estas tecnologías emergentes. El desarrollo de aplicaciones de entretenimiento en estos escenarios y en general de desarrollo de aplicaciones de red requiere una gran cantidad de esfuerzo de parte de los desarrolladores en lo que concierne al entendimiento de las tecnologías, las arquitecturas de red y el diseño e implementación de protocolos de comunicaciones. Cada nuevo proyecto bien sea

tiempo a los mismos, que además prefieren juegos relativamente fáciles y que no impliquen grandes inversiones de tiempo y dinero.

³Hardcore Gamer denota aquellos jugadores de juegos de video cuyo tiempo libre es dedicado en su gran mayoría a los juegos o a la lectura acerca de los mismos.

⁴En la ingeniería de sistemas se entiende por contexto las circunstancias en las que un dispositivo está siendo utilizado.

⁵De acuerdo a la consultora Gartner para 2005 la inversión en tecnologías inalámbricas de comunicación y aplicaciones estaba en el TOP3 de las estrategias tecnológicas de las compañías europeas - http://www.gartner.com/press_releases/asset_125194_11.html.

comercial o de investigación invierte grandes cantidades de tiempo en solucionar los problemas antes mencionados, dado que no existen frameworks que puedan facilitar el trabajo permitiendo una mayor concentración en el desarrollo de las funcionalidades únicas de cada proyecto.

Al referirnos a framework acudimos a la definición del libro "Design Patterns"⁶ con un toque nuestro: *Un framework es un conjunto de componentes que cooperan entre sí y que juntos hacen un diseño reusable para un tipo específico de software.*

Un framework permite a la comunidad de desarrolladores de aplicaciones en un mismo dominio, reducir el TTM (Time To Market) de las aplicaciones gracias a su reutilización. Esto, a pesar de que la construcción del framework es una tarea tediosa, es también un esfuerzo único.

Existen una buena cantidad de ejemplos en casi todos los dominios de desarrollo de software. Si tomamos el caso de las aplicaciones Web, podemos encontrar frameworks para la solución de casi cualquier problema desde el acceso a datos (hibernate, JDO, ADO.NET) hasta el control del flujo de eventos (Struts, Spring) y la creación de las interfaces Web (JSP, JSF, ASP.NET). Esto muestra claramente la gran ventaja que representaría el tener un framework para el desarrollo de juegos multiusuario.

La construcción de un framework encargado de la infraestructura de comunicaciones entre los nodos pertenecientes a una sesión de juego y que provee una interfase al juego, reduciría significativamente el tiempo de desarrollo de los juegos u otras aplicaciones. Estas aplicaciones pueden ser de tipo investigativo o juegos y aplicaciones comerciales, que a largo plazo ayudarán al perfeccionamiento y mejoramiento del framework mismo.

1.2. Definición del Problema

Este trabajo de grado define y diseña, en términos de servicios y protocolos, de un framework para la construcción de juegos multiusuario. Este framework prestará los servicios de comunicaciones y monitoreo del contexto de acuerdo al alcance propuesto. El diseño será general dejando las limitaciones de cada plataforma específica a la implementación en dicha plataforma.

Adicionalmente, se realizará una implementación de la especificación sobre la plataforma J2ME/Bluetooth, realizando pruebas por medio de aplicaciones de ejemplo que se ajusten a distintos escenarios.

⁶La definición publicada en el libro "Design Patterns" dice: Un framework es un conjunto de clases que cooperan entre si y que juntas hacen un diseño reusable para un tipo específico de software.

1.3. Limitación del Alcance

Dado que el proyecto hace una definición general de un framework sin tener consideraciones de una plataforma específica, cada implementación deberá resolver de la mejor forma posible, lo que sea necesario para su construcción.

El foco de este proyecto es netamente técnico, por lo tanto, las discusiones conceptuales son mínimas, y incluyen conceptos tales como trabajo colaborativo, aplicaciones dependientes del contexto, computación ubicua, computación pervasiva, P2P, etc.

En cuanto a la implementación Bluetooth, dado es una prueba de concepto del framework, no se discuten las diferentes opciones en cuanto a seguridad, optimización y especialmente soporte para scatternet.

1.4. Guía al Lector

El contenido de este documento puede ser de interés para los diferentes tipos de audiencias. Por lo tanto y como herramienta para una mejor comprensión se presenta un breve resumen de cada una de las partes y lo que podría ser tenido en cuenta de acuerdo al interés del lector.

Desarrollo de aplicaciones: Parte 2.2, parte 5 y los ejemplos de uso.

Dominio del problema: Partes 1 y 2.

Desarrolladores que busquen mejorar el framework: Se recomienda leer todo el contenido del trabajo e incluso las referencias que se consideren necesarias.

Parte 1:

Introducción al problema y alcance del proyecto.

Parte 2:

Presentación del marco teórico en términos de los conceptos generales en el dominio del problema y estado del arte de proyectos que tratan la misma problemática.

Parte 3:

Presenta los requerimientos para un diseño general del framework, el diseño general del mismo en términos de servicios y protocolos y el diseño de la implementación J2ME/Bluetooth.

Parte 4:

Guía para utilizar el framework en la construcción de juegos y aplicaciones. Es muy práctica y no ahonda en los detalles del diseño y construcción del mismo. Está basada en la implementación sobre J2ME/Bluetooth.

Contiene las validaciones de la efectividad del framework para reducir los tiempos y esfuerzos en el desarrollo proveyendo software reusable de calidad.

Parte 5:

Se presentan los elementos iniciales para una discusión en torno al trabajo realizado. Primero se examinan los problemas encontrados en la realización del diseño general y la posterior implementación J2ME/Bluetooth. Luego se exponen las conclusiones y por último se esquematiza lo que sería el trabajo futuro con base en el framework.

2. MARCO TEORICO

2.1. Conceptos Generales

La necesidad de desarrollar el GA P2P Network Framework ha nacido de la combinación del estudio de varios conceptos en diferentes áreas de investigación. En la primera parte del marco teórico se analizará los conceptos más importantes y la forma como se relacionan o influyen los requerimientos con base en los cuales se diseñó el framework.

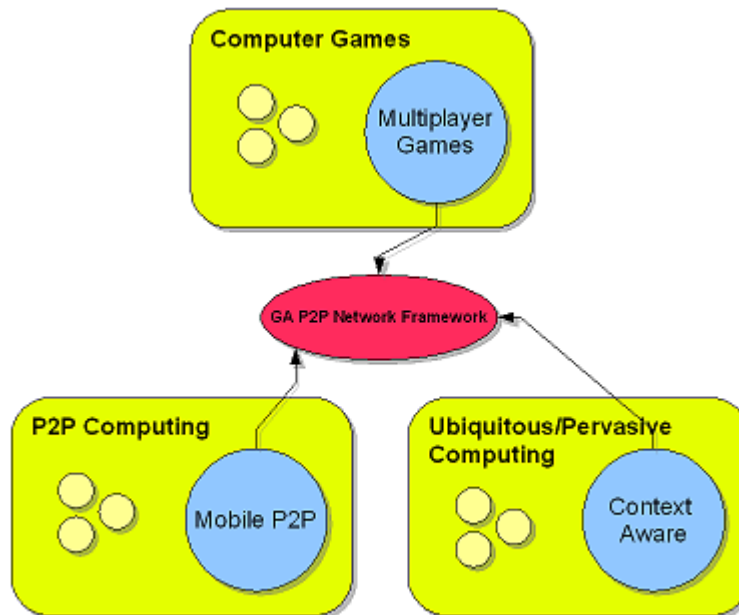


Figura1: Conceptos centrales relacionados con el proyecto.

La figura anterior muestra los principales conceptos relacionados con el proyecto. Las redes P2P, los juegos de video y la computación ubicua y pervasiva son el marco de trabajo general. A nivel específico, el GA P2P Network Framework es una solución para la creación de juegos multiusuario sobre redes P2P móviles y con dependencia del contexto.

Además de los conceptos centrales descritos anteriormente existen otros conceptos que también determinan en un grado menor el diseño y construcción del framework. Estos conceptos son las tecnologías de comunicaciones y su evolución, los protocolos de comunicaciones y como hacer una definición formal de los mismos y por último, de la ingeniería de software, framework y sus características.

2.1.1. Juegos Multiusuario

Los juegos multiusuarios son aquellos que permiten jugar simultáneamente a varios usuarios. Una definición un poco más extensa dice que son juegos que involucran la competencia entre varios usuarios, con o sin el concepto de vencedor/ vencido y que está regulado por un conjunto de reglas. Además, es importante resaltar que para nosotros son aquellos en los que la interacción tiene lugar en al menos un dispositivo de computo.

2.1.1.1. Clasificación

Existen diferentes clasificaciones para los juegos en general. En primera instancia, y la más conocida por el público, está la clasificación de acuerdo al género⁷: acción, aventuras, estrategia, primera persona, de rol, carreras, combate, deportes, rompecabezas, cartas, músicas, de mesa, etc.

Un primer acercamiento a una clasificación técnica y/o teórica de los juegos multiusuario se compone de la siguiente manera⁸:

1. **Juegos mono-usuario (Solo Games):** Estos son juegos organizados en rondas en los que cada jugador juega sólo e independiente de los demás. Sus puntajes son comparados al final de la ronda y se determina un ganador.
2. **Juegos basados en turnos (Turn-Based Games):** En este tipo de juegos la sesión procede en turnos discretos en lugar de tener que hacerlo de forma continua. Los usuarios pueden actuar de dos formas durante un turno: cada uno espera su turno para actuar o todos actúan durante un mismo turno.
3. **Juegos de interacción permanente (Act-Whenever Games):** Estos son juegos que pueden durar largos períodos de tiempo en donde cada jugador puede ingresar al juego en cualquier momento y ejecutar una serie de acciones.
4. **Juegos de actualización lenta (Slow-Update Games):** En estos juegos los agentes configurados por el jugador permanecen durante largos períodos de tiempo interactuando en el juego con base en las reglas dadas por el jugador.

⁷Dado que esta clasificación es importante sólo desde el punto de vista del jugador y no da cuentas del comportamiento a nivel de comunicaciones de red, no se definirán cada una de las categorías. Además no discrimina entre interacciones mono y multiusuario.

⁸Esta clasificación es poco rigurosa, además no tiene en cuenta las situaciones en las que se produce una sesión de juego.

Para este trabajo se tomara una clasificación más rigurosa que tiene en cuenta tanto las características del uso de la red como las situaciones posibles en que una sesión de juego tiene lugar [15]. Esta clasificación está enfocada en juegos multiusuario sobre dispositivos móviles, los cuales hacen parte del dominio del problema y son de naturaleza más compleja que los juegos en consolas o Pcs tradicionales.

Esta es una clasificación bidimensional; la primera dimensión determina la forma en que los usuarios interactúan en la red, la segunda determina la forma en que las redes son usadas. A continuación presentaremos las categorías de la primera dimensión:

- **I1 Controlled:** Uno de los dispositivos involucrados toma el rol de maestro controlando todas las interacciones, por ejemplo, dando turnos.
- **I2 User Interaction:** Los intercambios de información entre dispositivos son consecuencia de acciones realizadas por el usuario explícitamente.
- **I3 Automatic Triggered:** El dispositivo se encuentra en continuo monitoreo del entorno físico en busca de dispositivos. Cuando otro jugador (o dispositivo relevante para el juego) es encontrado, el dispositivo alerta al usuario y espera por instrucciones sobre cómo proceder. Cuando otro jugador sale del área de cobertura del dispositivo el usuario también es alertado y esperará instrucciones sobre como proceder.
- **I4 Automatic:** Los dispositivos interactúan sin intervención de los jugadores. Todo esto se hace con agentes previamente configurados y entrenados.

La segunda dimensión, forma en que las redes son usadas, tiene las siguientes categorías:

1. **U1 Asynchronous:** La actualización se hace cuando sea posible y no es necesario tener datos completamente actualizados para continuar el juego.
2. **U2 Synchronous:** El juego requiere que exista una constante actualización de pequeñas cantidades de información.
3. **U3 Real Time:** El juego requiere altas cantidades de intercambio continuo de datos para poder brindar una buena experiencia de juego al usuario.

Si se hace un cruce entre esta clasificación y la categorización tradicional de los juegos basada en el género, se encontrara algo de la siguiente forma⁹:

⁹Al presentarse casos en donde la misma categoría de acuerdo a la categorización por géneros está en más de una casilla, se demuestra por qué la primera clasificación no es conveniente tratar temas técnicos como la construcción de un framework, dado que no da cuenta de los requerimientos y características de los juegos multiusuario.

	I1	I2	I3	I4
U1	Rompecabezas /Estrategia	Estrategia/Rol		
U2	Deportes/Simulación	Aventura/Simulación	Simulación	Rol/Simulación
U3	Deportes/Carreras	Aventura/Acción	acción	

Tabla 1: Clasificación de juegos multiusuario.

Un framework para la construcción de juegos multiusuario debe cubrir una o más de las combinaciones entre estas dos dimensiones.

2.1.1.2. Arquitectura de Dispositivos

Los dispositivos que intervienen en la ejecución de juegos multiusuario pueden estar organizados en varios esquemas:

- **Servidor Dedicado:** Existe un servidor o una lista de servidores a los que el software cliente de juego se conecta para proveer la funcionalidad multiusuario. El servidor puede presentar varios niveles de procesamiento pasando desde el control de usuarios, inicialización de sesiones y paso de mensajes hasta el control total de la lógica del juego, la administración de recursos gráficos y demás.
- **Cluster de Servidores:** Este esquema es similar al anterior, aunque en este los servidores están organizados en forma de cluster, manejan la lógica del juego y permiten al proveedor actualizar constantemente los mundos virtuales¹⁰.
- **Servidor Ad-Hoc:** Cada aplicación cliente es completamente capaz de jugar como servidor o host y a su vez ser un cliente. De esta forma los jugadores de una sesión se conectarán a una de las aplicaciones en juego. El usuario de la aplicación servidor a su vez puede tomar parte del juego. La próxima sesión cualquiera de las demás aplicaciones puede hacer el papel de servidor.
- **Redes P2P:** Las redes P2P se basan en el concepto de agrupar y utilizar el poder de procesamiento y ancho de banda de los dispositivos participantes en la red, eliminando la distinción entre servidores y clientes (ver 2.1.3 Sistemas P2P). En este modelo todos los jugadores son iguales y pertenecen a un grupo que es la red P2P. Una vez en la red los jugadores intercambian mensajes directamente y dependerá de la construcción del juego el ejercer control sobre las interacciones.

¹⁰Un mundo virtual es un ambiente simulado por computadores en dónde se busca que sus usuarios lo habiten e interactúen por medio de "avatars" o personajes virtuales.

2.1.1.3. Comunicaciones

La capa de red, encargada del direccionamiento y transporte de los mensajes de acuerdo a la arquitectura de dispositivos, es en la actualidad implementada sobre diversas tecnologías de comunicación. Los siguientes son algunos ejemplos de tecnologías de comunicaciones utilizadas por juegos multiusuarios:

- **Serial:** Conexiones físicas uno a uno entre dos dispositivos, realizadas vía MODEM o puerto serial. Este tipo de comunicaciones ya no es tan popular como antes debido a las limitaciones de ancho de banda, los costos asociados a la comunicación vía MODEM y las dificultades de escalabilidad de las redes sobre conexiones seriales¹¹.
- **Infrarrojo (IrDA):** Conexiones inalámbricas uno a uno entre dos dispositivos con movilidad reducida. Este tipo de comunicaciones no es ampliamente utilizado por los fabricantes de juegos debido a las limitaciones de ancho de banda y en alcance de la señal y principalmente la necesidad de línea de vista entre los dispositivos¹².
- **UDP:** Permite conexiones no orientadas a la conexión, no confiables (el mensaje enviado puede que llegue una vez, que no llegue o llegue en orden diferente al enviado), las cuales disminuyen el overhead de la red.
- **TCP:** Permite conexiones orientadas a la conexión, confiables (se garantiza que el mensaje llega en un orden específico), las cuales pueden disminuir el rendimiento de la red. La comunicación se mantiene durante la duración de la conexión.
- **HTTP Polling:** Estas son peticiones http repetitivas a un 'Game Server'. Cada petición lleva opcionalmente un mensaje sobre el movimiento del jugador y espera el nuevo estado del juego en la respuesta.
- **Bluetooth:** Permite conexiones con múltiples dispositivos inalámbricamente con gran movilidad dentro de un alcance muy limitado.
- **SMS/MMS:** Permite conexiones asíncronas entre múltiples dispositivos o entre varios dispositivos y un 'Game Server', cuyo envío normalmente sólo tarda unos segundos, pero al ser un protocolo no confiable, debe tener en cuenta que dependiendo de la congestión de la red pueden tardar hasta un día para ser enviados o pueden no ser enviados nunca, también existen limitaciones a la capacidad de un mensaje SMS, aproximadamente 160 caracteres (1600 bits).

¹¹Las conexiones por cable serial alcanzan una velocidad máxima de 112Kbps.

¹²A pesar de que la especificación de Irda contempla velocidades de transferencia de hasta 16 Mbps, las interfaces generalmente disponibles en el mercado implementan sólo la categoría SIR (serial infrared) con velocidades equivalentes a una conexión serial de máximo 112kbps.

Encontramos una cantidad de combinaciones posibles de las tecnologías de comunicación y la arquitectura de dispositivos. Cada juego además de pertenecer a una de las categorías permite acceder al mismo a través de una o varias de las combinaciones arquitectura/ comunicaciones. La combinación arquitectura/ comunicaciones a utilizar dependerá del dispositivo siendo utilizado por el cliente y del modelo de negocio del desarrollador.

2.1.1.4. Problemas Comunes

Los juegos multiusuario presentan ciertos problemas comunes que deben intentar controlarse al máximo para proveer una satisfactoria experiencia de juego. Estos problemas son:

- **Trampa:** De la misma forma que en cualquier otro juego, algunos jugadores recurren a hacer trampa para obtener ventajas en los juegos multiusuario. Esto se hace explotando errores o limitaciones de diseño en el software, a lo que las compañías de software tratan de hacer frente de numerosas maneras. En primera instancia por medio de sistemas de monitoreo y detección de jugadores haciendo trampa y su modo de operación. Segundo corrigiendo y actualizando el software de tal manera que se evite la trampa. Y por último aplicando penalidades a los jugadores que incurren en estas acciones como borrando la cuenta de dicho usuario.
- **Tiempos de respuesta:** La heterogeneidad de las redes de los diferentes usuarios en términos de sus capacidades y tiempos de respuesta (lag) determina la calidad o continuidad de las interacciones entre un jugador y los demás.
- **Pérdida u obsolescencia de paquetes:** De forma similar a los tiempos de respuesta, la pérdida de paquetes reduce de forma considerable la calidad de las interacciones entre un jugador y los demás.
- **Salida inesperada:** En juegos que dependen de la sincronización de estados entre los jugadores, la salida inesperada no notificada de alguno de sus miembros puede causar un bloqueo al juego e incluso reiniciar la sesión completamente.

2.1.2. Contexto

Para hablar de contexto, hay que ubicarse en el área temática de la cual éste como concepto hace parte. Por lo tanto, en esta sección se hablara de la

computación ubicua o omnipresente para luego introducir la computación dependiente del contexto (context aware computing).

Computación Pervasiva es un término para una corriente naciente de investigación enfocada en:

- *Dispositivos de cómputo numerosos, algunas veces accesibles y pocas veces visibles.*
- *Frecuentemente móviles o embebidos en el entorno.*
- *Conectados a una, cada vez más grade, red ubicua compuesta de una base cableada y terminaciones inalámbricas.*

2.1.2.1. Ubiquitous/ Pervasive Computing

Cada vez son más pequeños los dispositivos con capacidad de procesamiento, teléfonos y PDAs tan poderosos como los computadores personales nos han liberado de la utilización del tradicional computador de escritorio, haciendo posible llevarlos a todo lado en todo momento. A su vez, las tecnologías de conexión inalámbricas permiten la conexión de estos dispositivos con el mundo electrónico. La distinción entre tecnologías de comunicación y tecnologías de cómputo es cada vez más borrosa y no sólo a nivel de los dispositivos sino también en la variedad de opciones en las que permiten comunicarse: email, chat, voz y video.

En una escala diferente la computación está yendo aún más allá de los dispositivos de uso personal. Diferentes dispositivos interconectados, grandes y pequeños, junto con varias tecnologías de sensores que incluyen sensores de movimiento, sistemas de etiquetas hasta cámaras de video, se están utilizando para hacer salones e incluso edificios “inteligentes”. Pronto **la interacción con dispositivos de cómputo será “ambiental” y común en vez de ser privada y virtual**. A través de estos desarrollos, los dispositivos de computo estan invadiendo las entrañas de nuestras actividades personales y sociales, además de nuestros entornos.

Estamos siendo llevados en esta dirección por varias ramas de investigación, comenzando con el trabajo de **Weiser** (1991) [31] en donde establece su visión sobre “computación ubicua” (ahora conocida como computación pervasiva). Nuevas áreas de investigación han sido conformadas en torno a nociones como “realidad aumentada”, “interfaces tangibles”, “wearable computers” [30], “edificios cooperativos” y otras más.

Lo que estas tecnologías tienen en común es que ellas mueven el sitio y el estilo de interacción más allá de los computadores de escritorio a un mundo real más grande, que es donde vivimos y actuamos. Desde el punto de vista de los usuarios es un gran logro. Desde el punto de vista de los diseñadores de sistemas y software presenta muchos retos. El computador de escritorio es un mundo que conocemos muy bien y que está bien controlado. El mundo real sin embargo, es

complejo y dinámico. El reto más importante es hacer sistemas usables en la cantidad de situaciones que presenta el mundo real: el siempre cambiante entorno de uso.

2.1.2.2. Context Aware Computing

Contexto es un concepto muy poderoso que ha sido bastante estudiado en el área de interacción humano-computador. La interacción con los sistemas se hace a través de actos explícitos de comunicación (señalar con el puntero del Mouse) y el contexto es implícito (las preferencias del usuario). El contexto es usado entonces para interpretar actos explícitos, haciendo que la comunicación sea mucho más eficiente, por lo tanto, al incluir sistemas de computo en el contexto de nuestras actividades cotidianas, este contexto podría ser interpretado y utilizado a nuestro favor requiriendo un esfuerzo mínimo de nuestra parte.

La comunicación puede además no requerir ningún esfuerzo para estar completamente integrada en todo lo que hacemos. **Yendo un poco más allá, lo que se hace deja de tener sentido como esfuerzos por comunicarse,** al contrario, son actividades completamente normales en donde la comunicación se ha hecho invisible (Norman 1998) [32]. Anteriormente, las discusiones acerca del contexto se concentraban en cómo se debería entender la forma de uso en situaciones contextuales sociales en que las herramientas son utilizadas para diseñarlas de forma que fueran entendibles, usables y significativas.

La noción de contexto se aprecia de una forma mucho más amplia actualmente. El término “computación dependiente del contexto” (context-aware computing) es generalmente entendido por aquellos que trabajan en computación pervasiva como clave para sus esfuerzos de dispersar y hacer que los sistemas de computo estén inmersos en nuestras vidas. El contexto se refiere a la situación física y social en las que los sistemas de cómputo se encuentran. Una de las metas de la computación dependiente del contexto es recoger información acerca del contexto de un sistema/ dispositivo y proveer servicios apropiados a la gente, el lugar, el tiempo, el evento, etc. Por ejemplo, un teléfono celular podría siempre vibrar en vez de sonar en un concierto si pudiera saber el lugar y fecha del mismo. Sin embargo, no se reduce tampoco a la simple recolección de información, el tener más información no es necesariamente mejor. Además, el recoger información acerca de nuestras actividades es una intrusión a nuestra privacidad. Lo que lleva a que la información del contexto es útil sólo cuando puede ser útilmente interpretada y, debe ser tratada con especial cuidado.

La teoría sobre dependencia del contexto es bastante refinada, los esfuerzos de investigación están centrados en cómo llevarla a la práctica. Además, los problemas de interacción entre humanos y dispositivos de cómputo son especialmente complejos y la computación dependiente del contexto redefine las nociones básicas sobre interfase e interacción. Las preguntas de investigación son abundantes: Cuál es el papel del contexto en nuestra experiencia cotidiana? Cómo

puede extenderse esto al dominio de la tecnología? Qué puede la computación dependiente del contexto realmente hacer por nosotros? Cómo podemos interactuar con tecnologías “invisibles” y aún así mantener control? Cómo podemos conservar la seguridad en cuestiones tales como la privacidad?

2.1.3. Sistemas P2P

Los sistemas P2P son una alternativa a los sistemas cliente servidor. De acuerdo con la teoría [18] los sistemas de cómputo pueden ser clasificados en centralizados o distribuidos. Los sistemas distribuidos pueden a su vez clasificarse en modelo cliente/ servidor y modelos P2P. En un modelo cliente/servidor el servidor es la entidad central y el único proveedor de servicios y contenidos. De acuerdo con los modelos P2P los recursos son compartidos entre peers que actúan tanto como clientes como servidores. Este modelo P2P puede darse bien sea de forma pura o de forma híbrida.

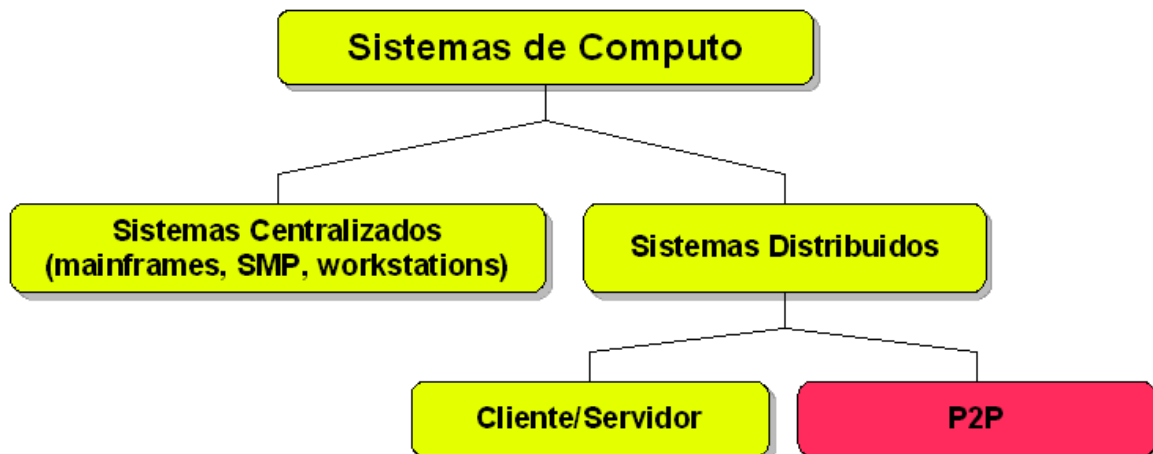


Figura2: Taxonomía de los sistemas de cómputo.

El termino **peer** encuentra sus raíces en francés antiguo **peer** y este a su vez del latín **aequālis** que se traduce **igual** en español. Es necesario entender el término peer para luego pasar a Sistemas P2P. Un peer es un nodo en una red P2P. Es la unidad fundamental de procesamiento de cualquier solución P2P. De acuerdo a [19] un peer es:

Cualquier entidad capaz de realizar cualquier trabajo útil y comunicar los resultados de ese trabajo a otra entidad a través de una red, bien sea directa o indirectamente.

Aunque la interacción de los peers en una red P2P es independiente de la existencia de alguna entidad central, la comunidad de trabajo en el área de P2P no se ha puesto de acuerdo en una definición que cubra cada aspecto de la

misma. Una forma de definir los sistemas P2P es **encontrada en [20]**:

Los sistemas P2P son aplicaciones que permiten a los usuarios formar redes lógicas encima de alguna infraestructura de transporte, compartir e intercambiar contenido digital.

Mats Thoresen presenta un conjunto de diferentes **definiciones en [21]** y de este conjunto sugiere 5 requerimientos para sistemas P2P:

1. Los sistemas P2P se componen de dispositivos con cualidades de servidor. Los Peers funcionan muy bien tanto como clientes que como servidores.
2. Los sistemas P2P tienen sistemas de direccionamiento propios, independientes de los de la tecnología y de otros estándares como DNS.
3. Los sistemas P2P son capaces de trabajar con conectividad variable. Los Peers pueden salir debido a fallas técnicas o por razones naturales y nuevos peers pueden ingresar en cualquier momento.
4. Un peer puede acceder otros peers y sus recursos de manera directa, sin pasar por entidades intermediarias, una vez la conexión es establecida entre dos peers estos dos se comunican directamente.
5. Los sistemas P2P utilizan los recursos en el borde de la red (the edge of the network) los cuales poseen más capacidad que la realmente utilizada.

Este modelo de sistemas distribuidos ofrece algunas características técnicas muy distintas de aquellas de los sistemas cliente/ servidor clásicos. Algunas de las ventajas son [21]:

1. **Capacidad:** Utilización de recursos baldíos como ancho de banda, almacenamiento y capacidad de procesamiento al borde de la red.
2. **Independencia:** Una arquitectura distribuida independiente de entidades centrales.
3. **Configuración:** La red es autónoma y autoconfigurable.
4. **Descentralización:** No hay cuellos de botella debido a que los recursos y servicios pueden encontrarse en cualquier lugar de la red.
5. **Extensibilidad:** Capacidad de agregar nuevos servicios y recursos con simplemente agregar un peer a la red.
6. **Tolerancia a fallos:** No existe un punto central cuyo fallo comprometería a toda la red.

A pesar de que los sistemas P2P proveen una cantidad posible de ventajas, algunos problemas importantes deben ser resueltos. Los tres mayores problemas son [22]:

1. **Descubrimiento y enrutamiento:** Cómo encontrar recursos y servicios en la red y cómo hacer el enrutamiento de mensajes a través de la red? No existe una entidad central en una red P2P, esto significa que los peers tienen que buscar entre los demás peers por los recursos o servicios que necesitan. Los Peers que busquen exactamente lo mismo se podrán comunicar con diferentes peers por diferentes rutas obteniendo resultados diferentes.
2. **Administración de recursos:** Contribución, asignación, replicación, etc. de recursos. Al no existir una entidad central cada peer debe decidir que tipos de recursos y servicios debe proveer, y cómo estos deben ser asignados una vez disponibles en la red. La naturaleza autónoma de un peer le brinda la posibilidad de no contribuir con nada en la red y simplemente tomar ventaja de lo ofrecido por los demás peers.
3. **Seguridad y privacidad:** Prevenir peers malintencionados y la protección de información personal, especialmente desarrollando mecanismos de confianza y confidencialidad, también es necesario que cada peer pueda controlar de que forma se usa la información personal de los mismos.

2.1.3.1. Arquitecturas P2P

Existen dos tipos de arquitecturas para sistemas P2P llamadas híbrido y puro. La distinción principal entre estas dos arquitecturas es que en la arquitectura híbrida existen una o más entidades centrales.

- **P2P puro:** En una arquitectura pura todos los peers tienen las mismas responsabilidades, no existe una entidad central responsable de administrar, coordinar y controlar los servicios y recursos en la red, todos los peers tienen las mismas responsabilidades, cualquiera de los peers puede ser removido de la red sin perder funcionalidades. Al no existir entidades centrales, los protocolos de enrutamiento, descubrimiento y asignación de recursos a construir son mucho más complejos, sin embargo, dada la naturaleza descentralizada, aplicaciones de alta disponibilidad, tolerancia a fallos y buena escalabilidad pueden ser construidos.

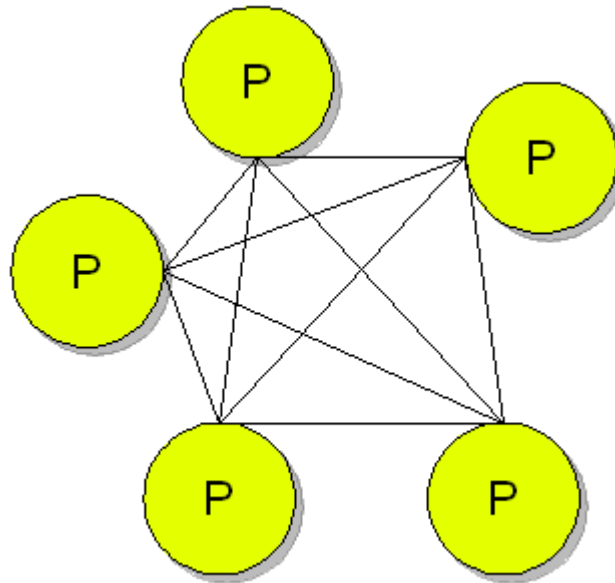


Figura3: Arquitectura P2P puro.

- **P2P híbrido:** En esta arquitectura existen entidades centrales responsables de proveer servicios e intercambiar información con los demás peers de la red. El sistema P2P debe escoger autónomamente estas entidades centrales y proveer los mecanismos para garantizar características como la tolerancia a fallos y evitar los cuellos de botella.

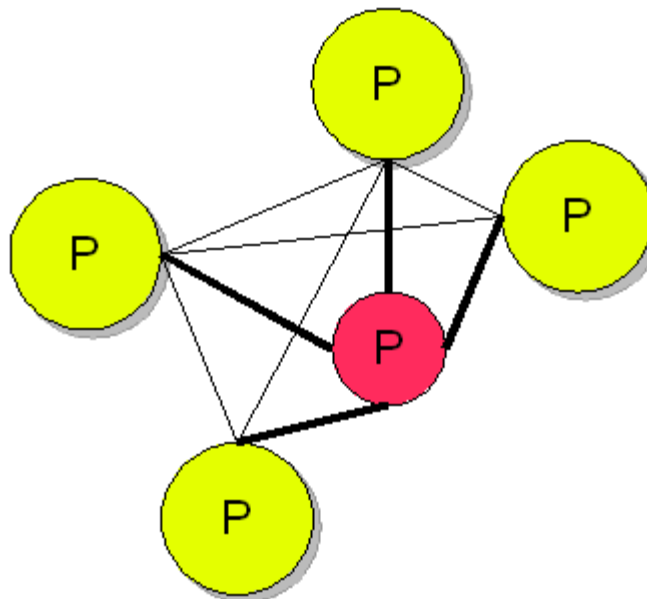


Figura4: Arquitectura P2P híbrido.

Los sistemas P2P presentan un paradigma completamente nuevo para la creación de sistemas distribuidos. El desarrollo utilizando el nuevo paradigma a resultado

en nuevos tipos de aplicaciones y en la creación de diferentes acercamientos a arquitecturas P2P. Las aplicaciones más famosas son aquellas de compartimiento de archivos (LimeWire, Bittorrent, eMule, Napster) y mensajería instantánea (MSN, ICQ, Skype). En [23] se identifican cinco diferentes categorías para aplicaciones P2P: Mensajería instantánea, compartimiento de contenido digital, “grid computing”, trabajo colaborativo y servicios web.

Los requerimientos del framework exigen considerar las situaciones en las que la red es inalámbrica con todas sus características (discontinuidad, obstáculos, fallas de alcance, etc). Estas características, en el marco de P2P, brindan aún más posibilidades para hacer nuevos tipos de aplicaciones en los nuevos escenarios de uso. Algunas características y consideraciones a este respecto se discuten a continuación.

2.1.3.2. P2P Mobile

En [24] los sistemas P2P Mobile o sistemas P2P móviles son definidos de la siguiente forma:

Un sistema P2P Mobile es un sistema distribuido móvil que consiste en dispositivos móviles, que continuamente cambian su posición y establecen relaciones con otros con base en la proximidad entre los involucrados.

Estos sistemas pueden ser implementados de dos maneras:

1. Sin infraestructura¹³: Utilizando alguna tecnología para comunicaciones móviles ad-hoc como bluetooth o IrDA.
2. Con infraestructura¹⁴: Utilizando Internet a través de una red celular de datos como GSM/GPRS, UMTS WI FI.

Existen diferencias claras entre los sistemas P2P móviles y los normales. Los sistemas P2P móviles son influenciados por los mismos fenómenos que la computación móvil en general. Las categorías principales de estos fenómenos son: comunicación, movilidad y portabilidad. A nivel de comunicación se refiere a interrupciones de las transferencias, problemas con tasas de transferencias bajas y cambiantes, integración de redes heterogéneas y cuestiones de seguridad. En

¹³Existen algunas implementaciones conocidas de este tipo utilizando interfaces bluetooth. En ellas cada peer hace uso de interfaces Bluetooth para conectarse con otros estableciendo una red LAN inalámbrica también conocida como PAN (Personal Area Network). Aquí los peers pertenecientes a una sesión se organizan autónomamente garantizando el paso de mensajes entre todos los peers y maximizando la tolerancia a fallos individuales de los peers.

¹⁴En donde cada dispositivo hace uso de la red de datos GPRS/UMTS/WiFi para mandar paquetes a un servidor de paso de mensajes disponible en Internet, el cual almacenará los paquetes recibidos y los entregará a los demás peers miembros de la sesión cuando estos realicen peticiones de actualización. En este caso encontramos varias funciones en el servidor: entrega de mensajes, lobby, recuperación de sesión, timeout de sesión, entre otras.

cuanto a la categoría de movilidad están los problemas de direccionamiento de dispositivos e información dependiente de la localización. Por último, a nivel de portabilidad se trata de reducir la cantidad de energía a utilizar, los problemas de interfase de usuario, la baja capacidad de almacenamiento y el bajo poder de procesamiento. Para los sistemas P2P los siguientes problemas son especialmente importantes:

- Descubrimiento de Recursos: La naturaleza dinámica de los sistemas P2P móviles requiere mecanismos más dinámicos para el descubrimiento de dispositivos y recursos.
- Envío de datos y sincronización: La alta disponibilidad es deseable para poder tener peers autónomos. Para obtener esa alta disponibilidad es necesario implementar algún esquema de sincronización. Esto introduce problemas complejos de consistencia entre peers.

Por lo tanto, las condiciones de movilidad traen un conjunto de posibilidades pero también conllevan dificultades. Por ejemplo, los dispositivos móviles están en constante movimiento, haciendo fácil su interacción en sistemas pervasivos. Sin embargo, el mismo movimiento de los dispositivos hará que con el tiempo los enlaces de transmisión fallen y sean interrumpidos. Es responsabilidad de quien diseñe software para este tipo de sistemas mantener un balance entre estas contradicciones y crear sistemas funcionales.

2.1.4. Tecnologías de Comunicaciones Móviles

Las tecnologías de comunicaciones móviles son:

Las que permiten establecer enlaces de comunicación sobre medios inalámbricos para conectar dos o más dispositivos, en donde se asume que uno o más de los dispositivos involucrados pueden cambiar de ubicación en cualquier momento.

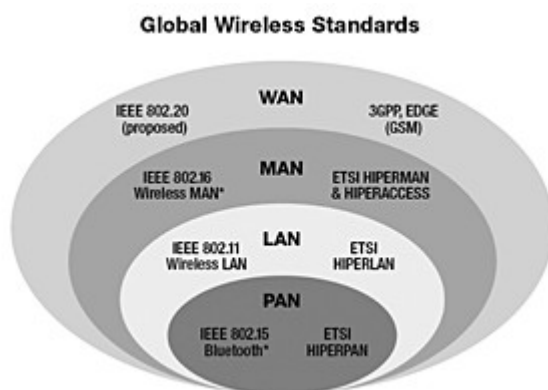


Figura5: Tecnologías de Comunicaciones Móviles.

La Figura5 muestra como diferentes tecnologías de comunicaciones móviles se sobreponen unas a otras en términos de alcances. Se tienen las PANs (Bluetooth, ZigBee, etc) como las de menor alcance con apenas generalmente 10 metros de diámetro hasta las redes WAN (WiMax) con radios hasta de 48 kilómetros.

Estas tecnologías están en casi todos los entornos. Se puede ver como el número de teléfonos celulares y usuarios de Internet inalámbrico ha crecido de manera significativa en los últimos años, alcanzando en algunos casos un cubrimiento de más del 100% de la población¹⁵:

¹⁵Fuente www.oecd.org/sti/ICTindicators.

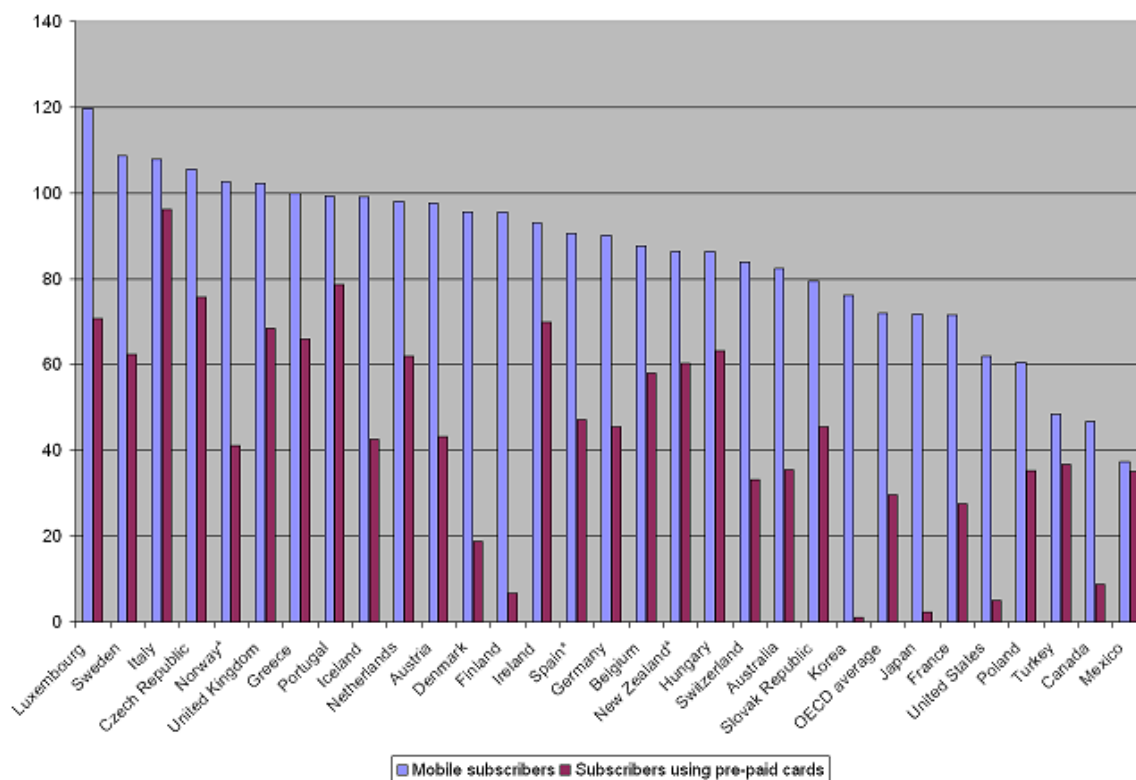


Figura6: Penetración de teléfonos móviles en los países OECD.

Los teléfonos móviles prestan sus servicios de transmisión de voz y de datos haciendo uso de la red celular. La red celular comprende una gran infraestructura física fija compuesta por antenas, conexiones de alta velocidad, enrutadores y conexiones con la PTNS. Además, establece conexiones con los teléfonos a través de tecnologías como GSM, GPRS y UMTS.

Por otro lado, el creciente despliegue a nivel mundial de la banda ancha, impulsó el desarrollo de tecnologías inalámbricas de red de área local (WLAN), especialmente 802.11a, 802.11b y 802.11g, para compartir el acceso a la red entre varios dispositivos en lugares como oficinas, sitios públicos y hogares. El gran potencial de las tecnologías inalámbricas, las recientes investigaciones y los deseos de reducir los costos asociados al despliegue de la banda ancha han influido en gran manera en el desarrollo de una nueva tecnología de banda ancha inalámbrica conocida como WiMax.

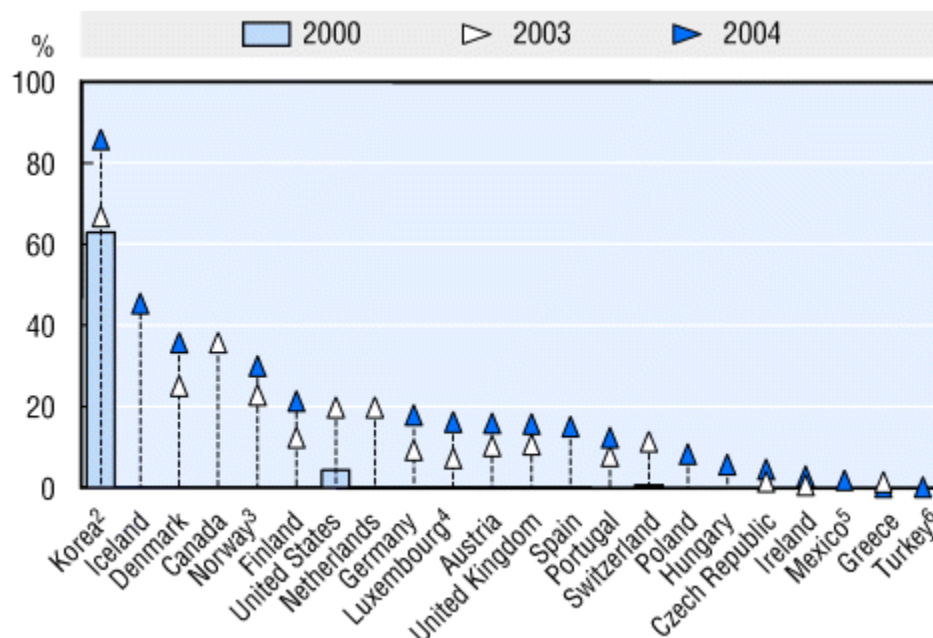


Figura7: Penetración de Internet en banda de ancha 2000 a 2004 según OECD.

Existen también una gran cantidad de dispositivos, entre ellos los teléfonos móviles, que han permitido el despliegue a gran escala de muchos avances tecnológicos, entre los que se destacan, las tecnologías inalámbricas de red de área personal como bluetooth. Estas tecnologías ofrecen interfaces baratas que permiten la comunicación entre dispositivos en áreas de pocos metros de cobertura. Este tipo de redes se conocen como redes de área personal o PAN (Personal Area Network).

Las redes celulares y la banda ancha inalámbrica también presentan modelos de comunicaciones móviles con infraestructura. Mientras que las redes de área personal no tienen infraestructura dado que el único condicionante para la interacción de dos dispositivos es la distancia entre ellos.

A continuación se presentarán de una forma un poco más detallada las mencionadas tecnologías de comunicaciones móviles.

2.1.4.1. Redes móviles Celulares

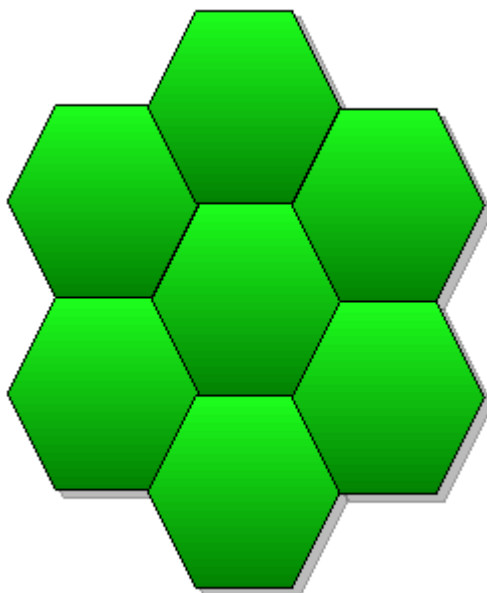


Figura8: Redes Celulares.

Estas redes han evolucionado en aproximadamente tres décadas desde la primera generación (1G) hasta la cuarta (4G), la cual se encuentra apenas en proceso de discusión y diseño. La siguiente tabla presenta una historia breve de las redes móviles celulares.

tecnología	1G	2G	2.5G	3G	4G
diseño	1970	1980	1985	1990	2000
Implementación	1984	1991	1999	2002	2010?
Servicio	Voz análoga, datos sincrónicos a 9.6 kbps.	Voz digital, SMS.	Más capacidad, paquetes de datos.	Más capacidad, ancho de banda de hasta 2 Mbps.	Capacidad más alta y completamente basada en IP, multimedia y ancho de banda de cientos de Mbps.
Estándares	AMPS, TACS, NMT, etc.	TDMA, CDMA, GSM, PDC.	GPRS, EDGE, 1xRTT.	WCDMA, CDMA2000.	Un solo estándar.
Ancho de Banda	1.9 kbps	14.4 kbps	384 kbps	2 Mbps	200 Mbps
Multiplexación	FDMA	TDMA, CDMA	TDMA, CDMA	CDMA	CDMA?
Red Base	PSTN	PSTN	PSTN, red de datos.	Red de datos.	Internet

Siglas y definiciones:

- AMPS = Advanced Mobile Phone Service
- CDMA = Code Division Multiple Access
- 1xRTT = 2.5G CDMA data service up to 384 kbps
- EDGE = Enhanced Data for Global Evolution
- FDMA = Frequency Division Multiple Access
- GPRS = General Packet Radio System
- GSM = Global System For Mobile
- NMT = Nordic Mobile Telephone
- PDC = Personal Digital Cellular
- PSTN = Public Switched Telephone Network
- TACS = Total Access Communications System
- TDMA = Time Division Multiple Access
- WCDMA = Wideband CDMA
- UMTS = Universal Mobile Telecommunications System

2.1.4.2. Banda Ancha Inalámbrica Metropolitana

De acuerdo con el WiMax Forum WirelessMan o IEEE 802.16, como es oficialmente reconocida esta tecnología, se define como:

Una tecnología basada en estándares permitiendo la implementación de canales de banda ancha de última milla por medios inalámbricos como una alternativa al cable-MODEM o DSL.

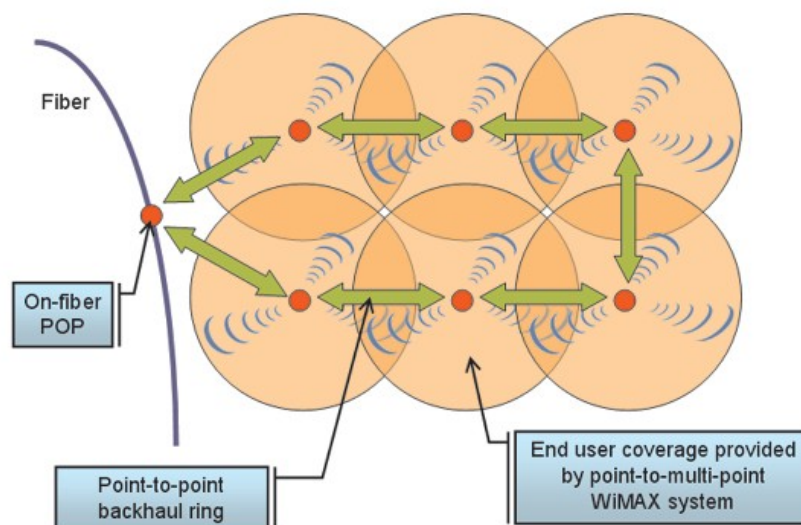


Figura9 Red Inalámbrica WiMax.

WiMax¹⁶ permite altas tasas de transmisión (hasta 70Mbps) en áreas del rango de kilómetros de radio (48 kilómetros), con interfaces baratas, aprovechando los desarrollos en materia de seguridad hechos para WLAN y reduciendo notablemente los costos de despliegue.

La capacidad y alcance de WiMax hacen que sea posible el despliegue de las siguientes aplicaciones:

- Interconexión de nodos WiFi entre ellos y con el Internet.
- Proveer una alternativa al cable MODEM y al DSL.
- Redes móviles de comunicaciones y datos de alta velocidad.
- Redes inalámbricas, parte del plan de continuidad para una empresa.
- Conectividad Nómada: donde los dispositivos están en continuo movimiento en una región bastante grande como para hacer uso de WLAN y PAN.

WiMax es una alternativa muy conveniente para áreas en donde no existe infraestructura previa de cableado o líneas telefónicas (en donde la banda ancha no ha sido económicamente viable).

2.1.4.3. PAN

Las redes de área personal (Personal Area Networks) se definen como:

*“Son aquellas que tiene un **factor** de forma pequeña y de bajo costo, que permite la creación de enlaces de radio de corto alcance entre dispositivos móviles, dispositivos estacionarios y entre combinaciones de los dos anteriores. La relación entre estos dispositivos existe dado que todos tienen una relación de distancia con una o varias personas.” **Cual es la referencia***

¹⁶WiMax no es la tecnología como tal, sin embargo, este término se utiliza comúnmente para referirse a WirelessMan o IEEE 802.16. WiMax es un sello de certificación para dispositivos que implementan el IEEE 802.16 de acuerdo a los parámetros definidos por el WiMax Forum.



Figura10: Diferentes dispositivos que intervienen en una PAN.

A continuación se mostrara algunas de las tecnologías PAN disponibles en las que se puede potencialmente hacer una implementación del framework:

- Bluetooth: Esta es la tecnología PAN más ampliamente utilizada especialmente por teléfonos móviles y accesorios relacionados. En 2002 fue ratificada la especificación Bluetooth 1.0 y en Noviembre de 2004 la versión 2.0. Bluetooth tiene un rango de comunicaciones de aproximadamente 10 metros y tasas de 768Kbps.

A noviembre 14 de 2006¹⁷ el número de dispositivos utilizando Bluetooth como tecnología PAN alcanzo los mil millones¹⁸. De acuerdo con el Bluetooth SIG (Grupo de Interés Especial) son los teléfonos móviles los que continúan dominando esos números seguidos por los dispositivos del tipo “manos libres”. De igual manera contribuyen en gran forma los audífonos, vehículos, reproductores multimedia, computadores portátiles, teclados y ratones. Se espera que para el 2010 se pueda llegar a la cifra de los dos mil millones de nuevos dispositivos producidos solamente en ese año. Para ese entonces, se espera que sean dispositivos médicos, de entretenimiento, de video, proyectores y cámaras digitales los que impulsen estos números.

Cada fabricante de interfaces bluetooth provee una librería para interactuar con la misma, las cuales son generalmente construidas en C. De acuerdo

¹⁷ Fuente: http://www.bluetooth.com/Bluetooth/Press/SIG/BLUETOOTH_WIRELESS_TECHNOLOGY_SURPASSES_ONE_BILLION_DEVICES.htm

¹⁸ En ingles moderno “un billón” significa “mil millones”, sin embargo, en ingles antiguo y en español “un billón” significa “un millón de millones”.

con la filosofía de Java “write once run anywhere”, los fabricantes de teléfonos y otros dispositivos con capacidad de ejecutar aplicaciones construidas por el usuario, al implementar la máquina virtual de Java, también implementan el JABWT (JSR-82). Lo que permite desarrollar aplicaciones en una plataforma común independiente del dispositivo.

- ZigBee: Es una tecnología de comunicaciones móviles PAN con rangos de hasta 30 metros y velocidad de hasta 250Kbps. ZigBee fue originalmente diseñada como una red para control y procesamiento en ambientes industriales. En diciembre de 2004 se ratificó la primera versión de ZigBee, la cual se realizó en trabajo conjunto con más de 100 empresas a nivel mundial [28].
Su despliegue a nivel mundial ha sido lento y enfocado en aplicaciones industriales y de control del hogar o domótica.

A medida que tecnologías como VoIP y los servicios de navegación en Internet para dispositivos móviles maduran, más y más fabricantes han integrado la conectividad WLAN en sus dispositivos (entre estos podemos encontrar Nokia [E60, E61, E70], Samsung [SGH-P200], Siemens [P51], Motorola [A910], etc). Debemos contar también con los procesadores Intel Centrino y AMD microprocesadores los cuales llevan la conectividad WLAN a casi la totalidad de los computadores de escritorio y portátiles.

La programación sobre esta tecnología está completamente soportada en las plataformas Symbian [27] y Windows Mobile (a través del .NET Compact Framework). Sin embargo, no existe soporte en la plataforma J2ME para conexiones ad-hoc [27].

2.1.5. Framework

De acuerdo con “Design Patterns” [25] un framework es “un conjunto de clases cooperantes que componen un diseño reusable para un tipo específico de software”. Pero según se presentará más adelante, un framework no es sólo clases, por lo cual se definirá framework de la siguiente forma:

Un framework es un conjunto de elementos cooperantes que componen un diseño reusable para un tipo específico de software.

Entre estos elementos se puede encontrar grandes cantidades de documentación (como este trabajo), herramientas, ejemplos de uso y por supuesto clases (asumiendo que se esta trabajando en una plataforma o lenguaje orientado a objetos).

2.1.5.1. Justificación de los Frameworks

Existen muchas razones para construir un framework, a continuación mostraremos las 5 más importantes:

1. **Los frameworks ofrecen soluciones a problemas comunes en áreas específicas:** Esto significa que otras personas se vieron enfrentadas a problemas similares en la construcción de diferentes sistemas. Por lo tanto, nace la necesidad de agrupar todas estas funcionalidades comunes y de bajo nivel en un mismo lugar que sea fácil de mantener, mejorar y por supuesto reutilizar. Gracias a esto los usuarios de un framework se pueden concentrar en la implementación de requerimientos específicos a determinada solución.
2. **Los frameworks son libres de errores:** Lo mas deseado es que esto fuera completamente cierto, pero no lo es. Lo que es cierto es que los desarrolladores de frameworks invierten grandes cantidades de tiempo estabilizando, manteniendo, mejorando y corrigiendo los errores en sus frameworks. Este proceso requiere que las empresas inviertan sus mejores recursos en el proceso debido a los niveles de abstracción y al rol estratégico de un framework. En el largo plazo, estos esfuerzos resultan en frameworks bastantes estables y con un claro conocimiento (documentación) de sus alcances y fallos.
3. **Los frameworks pueden ser adaptables:** Deben ser adaptables y esto se hace por medio de extensión y herencia de sus clases y demás elementos.
4. **Los frameworks están basados en patrones de diseño:** Como los frameworks han de ser compartidos y estudiados por todos sus usuarios, es necesario que sean diseñados de acuerdo a patrones de diseño ampliamente conocidos en la industria.
5. **Los frameworks son bien documentados:** Una vez más, los usuarios de un framework deben tener las herramientas para entender todo lo relacionado con el framework, y además de los patrones de diseño es posible tener información en forma de modelos, ejemplos e incluso código fuente. Es importante tener en cuenta que todos deben estar sincronizados con la última versión del framework.

También existen características no tan buenas de los frameworks:

- **Código en exceso:** Cuando un framework cubre un dominio muy amplio o cuando una aplicación no usa completamente las funcionalidades de un framework existirán cantidades no despreciables de código sin usar, que podrá estar cargado en la memoria principal y en últimas tener un impacto en el desempeño de la aplicación o la disponibilidad de recursos, etc.

- **Inversión de tiempo en la evaluación de frameworks:** El proceso de evaluación y adopción de un framework toma cantidades no despreciables de tiempo. Este tiempo es invertido en la evaluación y pruebas de diferentes frameworks en un mismo dominio. Algunos autores dicen que este tiempo es el mismo invertido en hacer el diseño propio de una solución desde cero. Sin embargo, después de un corto análisis de la materia se encontró:
 - Este tiempo de evaluación será más corto cada vez y la decisión tenderá por el mismo framework, al tiempo que la compañía cliente se especializa en proveer software en determinado dominio.
 - El desarrollo continuado del framework por parte de su proveedor y la comunidad de sus usuarios es un intangible valioso para todos los usuarios, lo que significa obtener mejoras en el framework sin hacer la inversión necesaria (en algunos casos cooperación con el proveedor del framework), que tomaría el mismo desarrollo in-house y teniendo en cuenta las mejoras desarrolladas a petición de otros usuarios del framework, que a pesar de desarrollar aplicaciones en el mismo dominio podrán tener diferentes problemas.

- **Entrenamiento tardío en el framework:** El tiempo inicial invertido en la evaluación del framework nunca es suficiente para conocer los pequeños detalles, entonces se dan casos en donde el framework hace suposiciones en cuanto a lo que se debió haber hecho antes y lo que pasará después de la ejecución de sus rutinas. Este tipo de situaciones no son bien documentadas o son pasadas por alto en las evaluaciones. Por lo tanto, es necesario volver a estudiar y adaptar el código al funcionamiento del framework.

2.1.5.2. Características del Framework

Las siguientes son las características que todo framework debe tener:

- **Está compuesto por varios elementos:** Estos elementos pueden ser abstracciones como clases e interfaces, pero también scripts, archivos de configuración, documentación y otros.
- **Sus elementos cooperan entre sí:** Cada componente está relacionado con uno o más de los otros componentes. Y para poder ejecutar cada tarea algunos de ellos tienen que colaborar.
- **Sus elementos y lo que hacen son reusables:** Un framework provee la lógica de negocio de bajo nivel y estructura (común a varios escenarios y convenientemente adaptable) en la que participan varios elementos del mismo. Esta lógica al igual que los elementos es reusable.

- **Impone una metodología:** La estructura del framework hace necesario trabajar de cierta forma para ponerlo a punto, para obtener referencias de objetos, para compilar, etc. Resultando en que cada framework influencia la metodología de desarrollo de quien lo usa en cierto grado. Especialmente cuando el framework juega un papel completamente central en la estructura de un sistema, la metodología del framework define casi completamente la metodología del proyecto (no completamente por que siempre hay espacio para la disciplina).
- **Está dirigido a un área específica:** Un framework debe estar dirigido a un dominio específico y finito. De otra forma su desarrollo y mantenimiento sería muy costoso o terminaría siendo algo muy generalista que no proporciona ningún valor agregado a sus usuarios. La clave está en limitar el alcance y proveer las herramientas para extender el framework en caso de ser necesario.
- **Permite concentrarse en los requerimientos propios de la aplicación:** Una vez entendido y probado, el framework permite a sus usuarios concentrarse en la implementación de los requerimientos específicos de la aplicación.
- **Provee puntos de extensión:** Este es uno de los mecanismos que permite a un framework ser limitado y a la vez adaptable a las necesidades específicas de un usuario, que no son cubiertas por el framework. Por ejemplo, cuando estas necesidades van un poco más allá del dominio del framework.
- **Se apoya fuertemente en patrones de diseño:** Un framework es usado por varios usuarios y se supone que debe ser fácilmente entendido por quien lo use. Por lo tanto, es conveniente que su diseño esté basado en patrones ampliamente conocidos. De esta forma los programadores usuarios pueden entender mejor las decisiones de diseño y el propósito de los elementos, lo cual es necesario para la adaptabilidad del framework.

2.1.5.3. Framework vs. Librería

Un framework es diferente a una librería (puede ser un jar o un dll) por que una librería es una colección de clases e interfaces (o funciones para ambientes no OO) que no están necesariamente relacionadas unas con otras, que pueden trabajar independientemente y proveer funcionalidades en dominios completamente diferentes.

2.1.5.4. Desarrollo de Frameworks

El desarrollo de frameworks es en gran parte igual al desarrollo de cualquier otro proyecto de software. Sin embargo, existen algunas cosas que adquieren particular importancia:

- **Documentación:** El proceso de desarrollo de software debe enfatizar la creación de documentación fácil de usar dado que esta hace parte de los entregables del framework junto con los binarios y el código fuente. Esta documentación es más extensa y detallada que en otro tipo de proyectos. Y es necesario que este completamente sincronizada con la implementación. Los ejemplos de uso (código fuente de una aplicación que usa el framework) se consideran como parte de la documentación, de ahí que se considere que deben ser parte del proceso de desarrollo del framework.
- **Pruebas:** Como una de las características del framework es que sea libre de errores, la realización de pruebas cobra vital importancia, y estas se hacen de varias formas. En primera instancia un nivel mínimo de calidad es requerido antes de cualquier salida importante de producción, para este propósito se utilizan pruebas unitarias y casos de pruebas como estándares de-facto. También, a medida que el framework comienza a ser usado por diferentes usuarios, es importante establecer mecanismos para obtener retroalimentación de estos. Esta retroalimentación es en forma de reporte de errores, peticiones de nuevas funcionalidades o mejoras e, incluso parches al código.

2.1.6. Protocolos y Especificación de Protocolos

A continuación se presentara el concepto de protocolo y las características de la especificación de un protocolo.

2.1.6.1. Protocolos de Comunicaciones

Un protocolo es definido como [26]:

Un conjunto de reglas que controlan el intercambio de información en un sistema distribuido.

La especificación de un protocolo incluye la definición de un formato válido para los mensajes (sintaxis), las reglas de formación para el intercambio de mensajes (gramática) y define un vocabulario de mensajes válidos que puedan ser intercambiados (semántica).

Un protocolo estandariza la interacción entre dos entidades que usan un canal de comunicación, por eso es importante que el conjunto de reglas que componen un

protocolo estén completas y sean consistentes, para esto es necesario que el protocolo contenga reglas tales como, el inicio y el fin del intercambio de datos, la sincronización entre el emisor y receptor, la detección y corrección de errores en la transmisión, el formateo y la codificación de los datos.

2.1.6.2. Especificación de Protocolos

Una especificación de protocolos se divide en cinco partes:

1. Descripción del Servicio: El servicio que provee el protocolo.
2. Descripción del Entorno: Las suposiciones acerca del entorno en que este se ejecuta.
3. Vocabulario: El vocabulario de los mensajes usados para implementar el protocolo.
4. Codificación: La codificación de cada mensaje en este vocabulario.
5. Reglas de Intercambio: Las reglas que mantienen la consistencia entre el intercambio de los mensajes.

Hay diez reglas básicas para un buen diseño de un protocolo.

1. Definir completamente el problema para el cual se va a diseñar el protocolo. Esta definición debe hacerse en términos de requisitos, restricciones y límite en el alcance.
2. Definir el servicio que soluciona el problema a todo nivel de abstracción, antes de decidir cuales estructuras son requeridas para implementar el servicio. Esta definición, sin embargo esta en términos de agrupaciones de componentes y sus responsabilidades de cada componente con respecto a los requisitos, restricciones y limite en el alcance.
3. Diseñar los componentes que atienden funcionalidades externas antes que los de funcionalidades internas. Es importante considerar la solución como una caja negra y decidir como esta debe interactuar con el entorno. Luego decidir como funciona internamente la caja negra.
4. Hacer un diseño de protocolo simple. Los protocolos muy robustos son difíciles de implementar, verificar y son menos eficientes. El trabajo de los diseñadores es identificar los problemas simples separarlos y resolverlos individualmente.
5. No relacionar lo que es independiente. Esto tiende a ocurrir cuando se desea hacer protocolos compactos, cosa que aumenta la complejidad y

hace compleja la mantenibilidad del protocolo.

6. Un buen diseño es abierto, extensible y permite una mejor resolución de problemas.
7. Antes de implementar el diseño, se debe construir un prototipo de alto nivel y verificar que el diseño sea el correcto.
8. Implementar el diseño y optimizarlo.
9. Verificar que el diseño optimizado es equivalente al diseño de alto nivel que tomó como correcto.
10. No se salte de las reglas 1 a la 7.

Existen sin embargo, diferentes mecanismos para la especificación de protocolos. Estos mecanismos hacen uso de diferentes herramientas para hacer una definición de protocolos clara y fácil de entender. Estas herramientas son del tipo tablas, diagramas, diccionarios, etc. Por lo tanto, es necesario hacer la especificación de protocolos utilizando una metodología y siguiendo las reglas básicas antes mencionadas.

A continuación se mostrara el mecanismo tradicional para la especificación de protocolos: Máquinas de estado finito.

MÁQUINAS DE ESTADO FINITO

Una máquina de estado finito usualmente es especificada con una tabla de transición, como se muestra en la siguiente tabla.

Condición		Efecto	
Estado Actual	Entrada	Salida	Sig Estado
Q0	-	1	Q2
Q1	-	-	Q0
Q2	0	0	Q3
Q2	1	0	Q1
Q3	0	0	Q0
Q3	1	0	Q1

Para cada estado se definen un conjunto de reglas de transición, hay una regla por fila en la tabla y por lo general, es más de una regla por estado. El ejemplo

contiene reglas de transición para los estados Q0, Q1, Q2, Q3. Cada transición está dividida en cuatro partes, cada parte corresponde a cada una de las columnas en la tabla. Las primeras dos son condiciones que se deben satisfacer para que la regla de transición sea ejecutada y especifican:

- El estado en el cual la máquina debe estar.
- Una condición de entorno de la máquina, como un valor de entrada.

Las últimas dos columnas definen el efecto de la regla de transición y especifican:

- Cómo el valor de salida de la máquina cambia.
- El nuevo estado que la máquina alcanzaría si se aplica la regla de transición.

El modelo tradicional de máquinas de estado finito consiste en un conjunto de señales, de entrada y salida, que pueden tomar un conjunto finito de valores. La regla ejecutada con la variable de entrada y el efecto en la transición pueden cambiar los valores de la variable de salida.

El comportamiento de la máquina de estado finito de la anterior tabla se puede ver gráficamente de la siguiente forma:

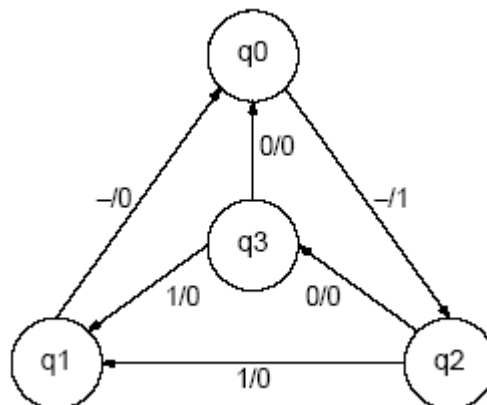


Figura11: Representación grafica de una maquina de estado finito

“Siendo menos formal, lo que se hace es modelar un protocolo como una máquina

de estados finitos o autómeta. Esto significa que el protocolo -o, mejor dicho, la entidad de protocolo- puede estar en un sólo número finito de estados definidos en un instante dado. Por ejemplo, podría estar inactivo a la espera de un mensaje para transmitir o a la espera de recibir una confirmación. Las transiciones entre estados son consecuencia de un suceso entrante; por ejemplo, si un mensaje está listo para ser enviado o se recibe una trama ACK. Al presentarse un suceso entrante casi siempre se genera un suceso saliente asociado; por ejemplo, al recibir un mensaje, enviar por el enlace la trama I creada, o al recibir una trama NAK, retransmitir la trama I pendiente”.

“Por supuesto, algunos sucesos entrantes pueden provocar varios sucesos salientes posibles. El suceso saliente específico que se escoja estará determinado por el cálculo del estado de uno o más predicados (variables booleanas). Por ejemplo, el predicado PI podría ser verdadero si el $N(R)$ de una trama ACK recibida es igual al $N(S)$ de la trama I que espera ser confirmada. De este modo, si PI es verdadero, se liberará el almacenamiento temporal en el que se mantenía la trama I; si es falso, se iniciará la retransmisión de la trama”.

“Por último, además de generar un suceso saliente (y posiblemente un cambio de estado), un suceso entrante también puede tener asociadas una o más acciones locales o específicas. Como ejemplos se puede mencionar *iniciar un temporizador* e *incrementar la variable de secuencia de transmisión*”.

2.2. Estado del Arte

A continuación se mostrara un pequeño resumen de proyectos relacionados o similares al GA P2P Network Framework que han sido estudiados, evaluados y en algunos casos referencia para la concepción y diseño del presente trabajo.

2.2.1. Peer2ME (REFERENCIA)

Peer2ME es proyecto que busca construir un framework exploratorio para la construcción de aplicaciones sobre redes PAN sobre la plataforma J2ME. El framework es desarrollado como un proyecto de maestría del Departamento de Ciencias de la Computación e Información (IDI) de la Universidad Noruega de Ciencia y tecnología (NTNU). El proyecto fue desarrollado con un módulo para conectividad sobre interfaces Bluetooth y aplicaciones de ejemplo para verificar la conveniencia del framework en el dominio del problema.

Las conclusiones del proyecto dicen:

- Es posible la construcción de un framework de esta naturaleza sobre teléfonos móviles en la plataforma J2ME. Lo que implica que los teléfonos móviles y la plataforma J2ME son una plataforma adecuada para la construcción de aplicaciones colaborativas móviles.
- Concluyen que a pesar de presentar algunos inconvenientes (mencionados posteriormente) Bluetooth es una tecnología de transporte adecuada para la construcción de aplicaciones colaborativas móviles.
- Concluyen que la utilización de un framework como el peer2me hará fácil el desarrollo y reducirá el tiempo del mismo una vez los programadores estén debidamente entrenados en la utilización del framework.
- De acuerdo con la clasificación de aplicaciones utilizada en el proyecto (fue la clasificación base sobre la cual se diseñó la que utilizamos en el proyecto actual), la gran mayoría de aplicaciones se pueden construir existiendo limitaciones como las impuestas por J2ME/Bluetooth.

La siguiente es la lista de inconvenientes encontrados en la implementación sobre Bluetooth:

- La implementación J2ME en diferentes dispositivos varía sustancialmente, resultando en ajustes necesarios para cada dispositivo.
- Las opciones de multitasking en los diferentes dispositivos utilizados son bastante pobres.

- Dispositivos Bluetooth ejecutándose como maestros no pueden aceptar conexiones entrantes desde otros maestros. Además, los esclavos no pueden ser esclavos en más de una piconet al tiempo.
- A pesar de que la especificación Bluetooth se pueden tener de 7 conexiones simultáneas, no todos los teléfonos cumplen con este requerimiento existiendo en algunos casos teléfonos que soportan sólo una conexión.
- Los tiempos de descubrimiento de los interfaces Bluetooth son lentos y los tiempos de timeout por desconexión son largos.

2.2.2. JSR 259

El JSR-259 o la especificación para Java de un API para la creación de redes Ad Hoc en J2ME (Java Specification Request on Ad Hoc Networkg API for J2ME). Este trabajo se ha realizado bajo la iniciativa de diversos fabricantes de dispositivos y ha sido encabezado por BENQ Mobile.

El alcance del JSR-259 es el siguiente:

- Un API opcional y genérico que permite a los dispositivos móviles ofrecer y consumir servicios en una red Ad Hoc móvil.
- Permitir a quien implemente el API implementar uno o más módulos de protocolo de forma transparente (Upnp, Webservice, JXTA y ZeroConf).
- Permitir al desarrollador de aplicaciones implementar un nuevo módulo de protocolo y registrarlo en el API. Este nuevo módulo puede ser utilizado por la aplicación específica o por las demás aplicaciones en el dispositivo.

El JSR-259 definirá los siguientes métodos para los desarrolladores de aplicaciones cliente:

- Descubrimiento de servicios.
- Utilización de servicios.
- Consulta acerca del servicio y la capacidad de servicio.

El JSR-259 permitirá a los desarrolladores de servicios, hacer su trabajo en los protocolos incluidos en el framework o independientes de cualquier protocolo. Para ellos se definen principalmente los siguientes métodos:

- Registro de servicios.
- Alertas sobre la disponibilidad del servicio.

La primera versión de la especificación permite la comunicación de dispositivos sobre una red Ad Hoc bajo el modelo P2P. Está orientado a dispositivos con configuraciones CLDC 1.1 y CDC 1.0 sobre MIDP. Y lo que se busca principalmente es esconder la complejidad del problema de los desarrolladores de servicios y aplicaciones. Tampoco tiene consideraciones en cuanto a los protocolos, las implementaciones o la interfase de usuario y todo esto se deja a quien implemente la misma.

El proyecto sigue en su fase de definición y no ha obtenido actualizaciones desde marzo del 2006. Tampoco existen fechas para la liberación del mismo.

2.2.3. JXTA y JXME

JXTA es un framework para la creación de redes P2P¹⁹. Existen dos proyectos diferentes para implementar JXTA en esquemas P2P móviles, ambos sobre la plataforma J2ME:

- JXME con Proxy: Requiere de un peer central que actúa como Proxy entre los peers en la red (modelo P2P híbrido) y no cubre todas las características de un sistema P2P móvil maduro.
- JXME sin Proxy: Es un intento por crear una implementación madura de JXTA sobre J2ME bajo un modelo de sistema P2P puro. Hasta ahora no existe una implementación disponible para redes ad-hoc, sólo para redes TCP/IP.

¹⁹Fuente: <http://jxme.jxta.org/>

3. GA P2P NETWORK FRAMEWORK

El proyecto GA P2P Network Framework busca ofrecer un framework de comunicaciones y contexto para la construcción de juegos multiusuario en entornos P2P. Este documento presenta el diseño del framework en términos de requerimientos, conceptos, protocolos y servicios que prestan todas las funcionalidades identificadas en los requerimientos.

3.1. *Requerimientos*

El GA P2P Network Framework al igual que cualquier otro proyecto de software se diseña y construye con base en un conjunto de requerimientos. Estos requerimientos se dividen en funcionales y no funcionales. Siendo los funcionales los que definen los servicios que el framework será capaz de ofrecer, describiendo las transformaciones o acciones a realizar sobre ciertos parámetros para producir determinadas salidas. Los requerimientos no funcionales tienen que ver con características que de una u otra forma pueden limitar el sistema, como por ejemplo, el rendimiento (en tiempo y espacio), interfaces de usuario, confiabilidad, mantenimiento, seguridad, portabilidad, estándares, etc.

Dada la doble naturaleza de los problemas a tratar por el framework (a saber, problemas de comunicaciones y contexto y los problemas de sesión de juego) es conveniente categorizar los requerimientos como de tipo comunicaciones y contexto (CC), de tipo sección de juego (SJ) y de tipo framework (F).

- Los problemas de comunicaciones son del tipo: encapsulamiento de direcciones físicas, el establecer conexiones de red, monitoreo de peer, entre otros.
- Los problemas de sesión de juego son del tipo: Ingreso y salida de la sesión, asegurar la continuidad de la misma, etc.

El tercer tipo de requerimientos está relacionado con las características generales de un framework.

A continuación se presenta la lista de requerimientos definidos para el diseño y posterior implementación del framework.

3.1.1. Requerimientos Funcionales

Código:	Requerimiento:	Tipo:
RF1	El framework debe permitir establecer y terminar conexiones, de acuerdo a la tecnología de transporte, con otros peers.	CC
RF2	El framework debe permitir a un peer remoto establecer conexiones con el peer local.	CC
RF3	El framework debe proveer al juego independencia de la tecnología de transporte subyacente.	CC
RF4	El framework debe proveer encapsulamiento de las direcciones físicas de los peers al juego y a los servicios de alto nivel del framework mismo.	CC
RF5	El framework debe detectar y controlar los cambios de direcciones físicas e interfaces que un peer remoto pueda utilizar.	CC
RF6	El framework debe hacer notificación sobre las fallas en la capa de red.	CC
RF7	El framework debe reciclar las conexiones de tal forma que se minimice el número de veces que una conexión tiene que ser abierta.	CC
RF8	El framework debe permitir la creación de sesiones de juego.	CC
RF9	El framework debe permitir entrar a una sesión de juego existente.	SJ
RF10	El framework debe permitir a los jugadores de una sesión aceptar nuevos jugadores que desean entrar a hacer parte de la misma.	SJ
RF11	El framework debe permitir a los jugadores de una sesión rechazar nuevos jugadores que desean entrar a hacer parte de la misma.	SJ
RF12	El framework permite el registrar servicios a la capa de transporte.	F
RF13	El framework entrega los mensajes entrantes al servicio que vaya dirigido.	CC
RF14	El framework debe hacer notificación a la aplicación cliente sobre eventos en el mismo.	CC
RF15	El framework debe poder identificar el peer origen de un mensaje.	CC
RF16	El framework debe permitir enviar mensajes a todos los peers.	CC/SJ
RF17	El framework debe permitir enviar mensajes a un peer en particular.	CC
RF18	El framework permite realizar descubrimiento de peers que ejecutan instancias del juego.	CC

RF19	El framework permitir descubrir diferentes sesiones de juego ejecutándose en el momento.	CC
RF20	El framework provee información sobre la sesión local a peers remotos.	CC/SJ
RF21	El framework hace monitoreo de los peers pertenecientes a la sesión en términos de la posibilidad de establecer conexión con cada uno de ellos.	CC/SJ
RF22	El framework realiza notificaciones en cuanto a los cambios de visibilidad hacia los demás peers pertenecientes a la sesión.	CC/SJ
RF23	El framework provee un mecanismo de sincronización de objetos de control y variables de juego.	SJ
RF24	El framework administra y sincroniza la información necesaria para garantizar la continuidad de la sesión de juego.	SJ
RF25	El framework notifica y garantiza la continuidad de la sesión cuando un peer sale de la misma.	SJ

3.1.2. Requerimientos No Funcionales

Código:	Requerimiento:	Tipo:
RNF1	El diseño del framework no tiene requerimientos sobre los tipos de conexiones a utilizar.	CC
RNF2	El framework no tiene requerimientos sobre el lenguaje de programación o la plataforma de ejecución de la implementación.	F
RNF3	El framework debe asumir que la red presenta las características de una red inalámbrica: inestabilidad, obstáculos, interferencias, etc.	CC
RNF4	El framework no permitirá al juego acceder a mensajes que no estén destinados a él.	SJ

3.2. Diseño del Framework

A continuación se presenta el diseño general del framework. Primero se hará una presentación rápida de los conceptos del dominio del problema. Luego se presentarán los protocolos que guiaron el diseño del framework. Por último, se presentará la arquitectura del framework en términos de servicios y protocolos.

3.2.1. Diagrama Conceptual

En este diseño, existen algunos conceptos que son fundamentales para poder comprenderlo. A continuación se provee una descripción de dichos conceptos y las relaciones entre los mismos.

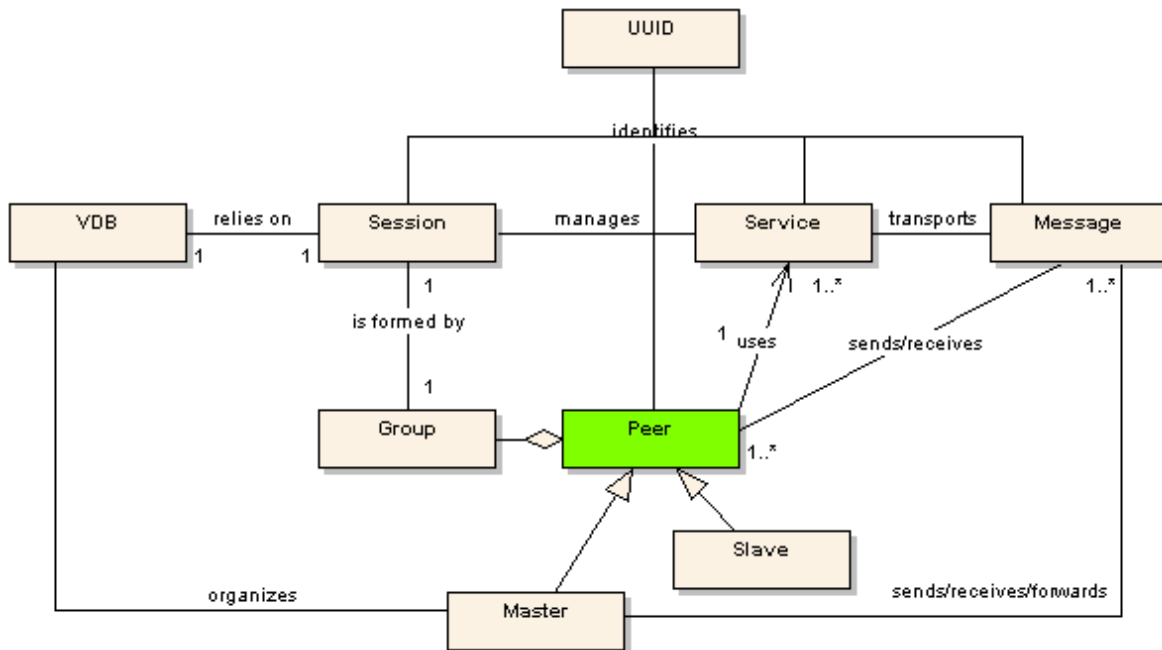


Figura12: Mapa Conceptual.

3.2.1.1. Peer

Es un dispositivo con capacidad de establecer conexiones de red y que implementa la totalidad de los protocolos acá descritos. Cada peer es identificado por un Universal Unique Identifier (UUID). Los peers son autónomos y funcionan independiente y asincrónicamente a los demás peers de la red. Algunos peers pueden tener dependencias hacia algunos otros como en los casos del Maestro de la sesión y el de utilización de un peer Rendezvous que sirve como puente (como los bridges en redes de datos) bien sea para ampliar la cobertura de una red inalámbrica o para proveer acceso a otra tecnología de comunicaciones. Estos peers Implementan exclusivamente los servicios Conexión y Discovery, además de no tomar parte activa en las sesiones de juego.

Un peer permite acceder a la información relativa a la sesión a la que pertenece desde cualquier otro peer. Y puede a su vez consultar información sobre la sesión de otros peers. Un peer debe tener mecanismos para hacer el descubrimiento físico de los otros peers. Y mecanismos para enviar y recibir información hacia y desde otros peers.

3.2.1.2. Sesión

El concepto de la sesión es el eje central del framework y se vera presente en todos los protocolos. Es una entidad lógica que provee una forma de segmentar la red P2P en grupos de peers organizados. Los peers pertenecientes a la misma sesión hacen parte del mismo mundo virtual (comparten valores en variables y objetos) del juego y en este pueden interactuar de la forma en que lo defina la lógica propia del juego. Una sesión de juego debe poseer como mínimo un peer. Los peers de la sesión conforman un grupo que está organizado en los siguientes términos:

- Existe uno y sólo un peer llamado el Maestro. Este peer hace el papel de secuenciador de mensajes (recibe los mensajes desde los demás Esclavos y entrega el mensaje uno a uno a los miembros del grupo de peers de la sesión), sincroniza los objetos de la sesión y las variables de juego y garantiza la máxima continuidad de la sesión a través de la organización (y posterior sincronización) del vector del dueño del balón.
- Existen cero o más peers que juegan el papel de Esclavos. Estos peers ingresan a la sesión haciendo peticiones de ingreso al Maestro. Envían mensajes de juego a través del Maestro y hacen monitoreo continuo de la visibilidad que tienen hacia los demás peers que hacen parte de la sesión. Cada peer conoce a los demás peers que hacen parte de la sesión.

Al existir problemas de comunicación o abandono por parte de algunos miembros de la sesión o del Maestro mismo, los peers Esclavos de la sesión cuentan con la información necesaria para elegir un nuevo Maestro y continuar con la sesión de juego, de tal forma que no existe un punto único de fallo y se garantiza la continuidad del juego al usuario final.

3.2.1.3. Grupo

Es el conjunto de peers que juegan en la misma sesión. Cada peer conoce a todos los integrantes del grupo y hace continuo monitoreo de los mismos para ver si es posible establecer conexiones con ellos.

3.2.1.4. UUID

Son los identificadores generados para distinguir de forma única entidades dentro de la solución. Estos identificadores se utilizan para las entidades *juego*, *servicio*, *sesión*, *peer* y *mensaje*. Los UUID son cadenas de texto generadas de forma tal que la posibilidad de generar otra cadena igual sea reducida al mínimo.

3.2.1.5. Servicio

Son componentes de software que se encargan de implementar determinadas funcionalidades y de ofrecer estas funcionalidades tanto a las aplicaciones cliente como a otros componentes del P2P Network Framework. Los servicios generalmente definen e implementan protocolos para comunicarse con otras instancias del servicio en otro peer. Los servicios generalmente utilizarán los mecanismos de transporte provistos por el Connection Wrapper (CW) el cual se explicara más adelante.

3.2.1.6. Vector del dueño del balón

En los ambientes P2P es necesario que las aplicaciones no dependan exclusivamente de ninguna entidad. De acuerdo con el acercamiento aquí propuesto, existirá siempre un peer Maestro que estará a cargo de la sesión durante un tiempo indefinido. Esta consideración de diseño hace que toda la sesión dependa del Maestro y que este sea un punto de fallo para toda la red. Se propone un mecanismo de recuperación basado en el ordenamiento de los demás peers del grupo. Este mecanismo es conocido como el vector del dueño del balón y determina, en caso de fallar el Maestro, quien será el nuevo Maestro a cargo de la sesión. Este vector es organizado por el Maestro actual y cada peer del grupo debe tener una copia actualizada del mismo.

3.2.2. Arquitectura de Alto Nivel

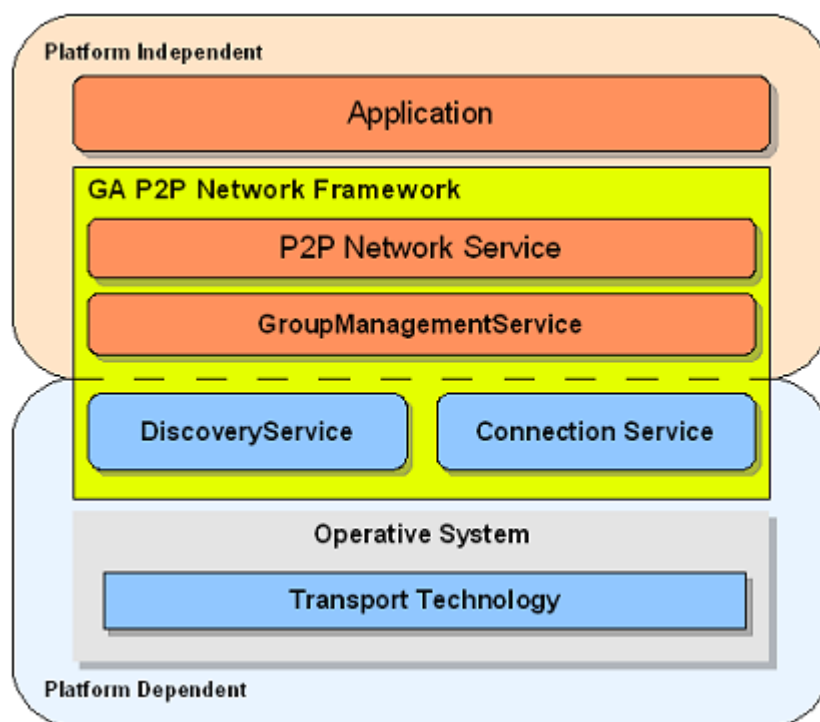


Figura13: GA P2P Network Framework Building Blocks

Este diseño está basado en 4 servicios y cada uno de estos servicios hace uso de uno o varios protocolos. El servicio base es el `ConnectionService` cuya funcionalidad es el envío y recepción de mensajes y respuestas, haciendo uso del `Connection Wrapper (CW)` implementado sobre la tecnología de transporte propia de la plataforma. El `DiscoveryService` presta los servicios de descubrimiento de peers, sesiones y monitoreo de peers. El descubrimiento y monitoreo de peers es un proceso que depende en gran medida de la tecnología de transporte que se esté utilizando y puede variar sustancialmente de una a otra. Por lo tanto, este proceso no se define en esta especificación. Para el descubrimiento de sesiones el `DiscoveryService` utiliza el `Discovery Protocol (DP)` haciendo uso del `ConnectionService`.

El `GroupManagementService` ofrece dos tipos de servicios: 1) Los relativos al manejo de las sesiones (inicio, entrada de un peer a la sesión, salida de un peer de la sesión y recuperación de la sesión) y 2) Los relativos a la sincronización de la información de control de la sesión entre los peers pertenecientes a la misma. Estos dos tipos de servicios utilizan mensajes del `Group Management Protocol (GMP)` los cuales se envían utilizando el `ConnectionService`. Por último, está el `P2PNetworkService` que es el servicio que funciona como punto único de interacción entre el juego y el framework; proveyendo acceso a todas las funcionalidades del framework de manera centralizada, además de facilitar el envío de los mensajes propios del juego.

3.2.3. Consideraciones de Diseño

A continuación se muestra las consideraciones o acercamientos que se toma para plantear la solución al problema.

3.2.3.1. Sincronización

Existen elementos del framework, como el vector del dueño del balón, que deben estar actualizados en todos los peers que hacen parte de ella. Igualmente existen valores dentro de la lógica del juego, que deben ser compartidos por todos los usuarios jugando. El proceso de llevar todos estos valores desde el Maestro hacia los Esclavos para asegurar que todos comparten los mismos valores para las mismas variables es lo que se llama sincronización. **RF23, RF24 y RF25.**

3.2.3.2. Maestros y Esclavos

El acercamiento planteado en el framework plantea la existencia de un peer maestro y de cero o varios peers esclavos. Lo que es claramente un acercamiento P2P Híbrido.

Este peer maestro tiene la característica de ser alcanzable por todos los peers de la sesión. Condición que no es siempre cierta dependiendo de la tecnología de transmisión de red. Las siguientes funcionalidades principales distintivas del peer maestro se muestran en la siguiente tabla de acuerdo a los requerimientos.

Código:	Requerimiento:
RF10	El framework debe permitir a los jugadores de una sesión aceptar nuevos jugadores que desean entrar a hacer parte de la misma.
RF11	El framework debe permitir a los jugadores de una sesión rechazar nuevos jugadores que desean entrar a hacer parte de la misma.
RF16	El framework debe permitir enviar mensajes a todos los peers.
RF17	El framework debe permitir enviar mensajes a un peer en particular.
RF23	El framework provee un mecanismo de sincronización de objetos de control y variables de juego.
RF24	El framework administra y sincroniza la información necesaria para garantizar la continuidad de la sesión de juego.
RF25	El framework notifica y garantiza la continuidad de la sesión cuando un peer sale de la misma.

Los demás peers de la sesión, los peers esclavos, se caracterizan por tener toda su interacción directamente con el Maestro. Estas interacciones en relación con los requerimientos son:

Código:	Requerimiento:

RF16	El framework debe permitir enviar mensajes a todos los peers.
RF17	El framework debe permitir enviar mensajes a un peer en particular.
RF23	El framework provee un mecanismo de sincronización de objetos de control y variables de juego.
RF24	El framework administra y sincroniza la información necesaria para garantizar la continuidad de la sesión de juego.
RF25	El framework notifica y garantiza la continuidad de la sesión cuando un peer sale de la misma.

Dependiendo de la tecnología de transporte utilizada para la implementación es necesario para los Esclavos realizar un monitoreo continuo de los demás peers del grupo para determinar la visibilidad hacia ellos. Esto es importante para la organización del vector del dueño del balón. En caso de que la tecnología de transporte de red garantice la posibilidad de conectar el peer local con cualquiera de los peers del grupo, el Maestro decidirá con base en otros parámetros cual será la organización del vector del dueño del balón en un momento dado.

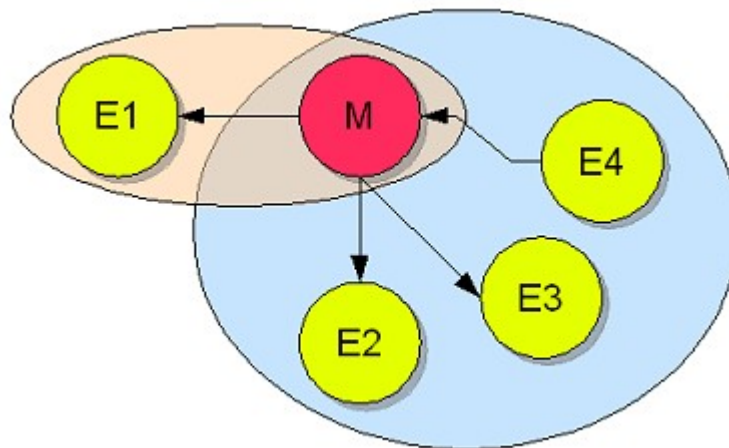


Figura14: Proceso de paso de mensajes 1.

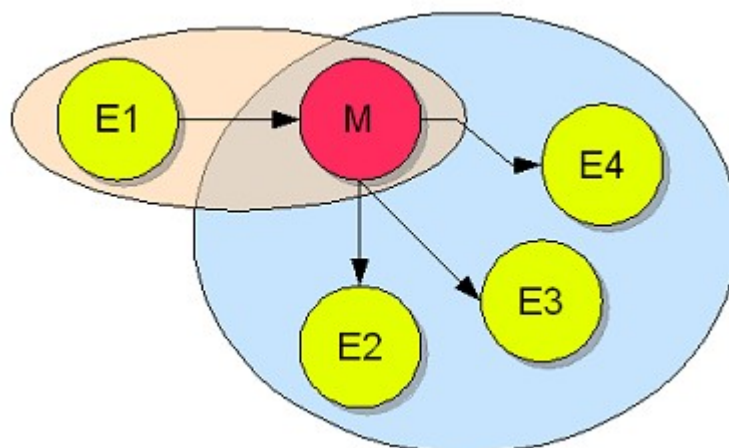


Figura15: Proceso de paso de mensajes 2.

En las figuras 14 y 15 es posible ver de forma clara como funciona el proceso de paso de mensajes desde un peer a los demás en la red haciendo uso del maestro.

3.2.3.3. Continuidad de la Sesión

Durante una sesión de juego puede suceder que para un peer cualquiera el maestro sea inalcanzable bien sea por que está fuera de rango o por que el maestro dejó de funcionar. En ese momento el framework debe comenzar un proceso que le permita al usuario continuar jugando, y a la vez velar por proveer esta continuidad conservando el mayor número de peers pertenecientes a la sesión, también se debe implementar mecanismos por parte del programador, si es necesario, para que esta continuidad sea efectiva, por ejemplo en un juego de damas chinas de dos jugadores, si uno de esto sale de la sesión el programador debería implementar mecanismo que le permita seguir jugando al jugador así el jugador oponente este por fuera de esta. **RF6, RF14, RF20, RF21, RF22, RF24 y RF25.**

De este proceso de reestructuración resulta uno o varios nuevos maestros cada uno con su sesión y un conjunto disyunto de peers. Cabe notar que cada vez que un peer pasa a ser maestro la sesión como tal se transforma en una sesión nueva y su identificador de sesión cambia. Esta reestructuración está guiada por el siguiente algoritmo:

1. Si el siguiente peer en el vector del dueño del balón es el peer local, entonces el peer local pasa a ser Maestro y se genera otro identificador para la sesión. Si ha encolado peticiones como se describe en el paso 3 entonces les responde con respuesta positiva y termina.
2. Por cada peer en el vector del dueño del balón, sí es posible conectarse al peer (esto lo puedo saber gracias al monitoreo de peers) entonces intento

conectarme al peer por medio de un JoinSessionMessage:

- Si la respuesta es negativa y hay más peers vuelve a 1.
 - Si la respuesta es positiva: si ha encolado peticiones como se describe en el paso 3 entonces les responde con respuesta negativa y termina. El nuevo estado de la sesión llegará por sincronización.
3. Si mientras se están ejecutando los pasos anteriores, se recibe una petición de JoinSessionMessage desde otro peer miembro de la sesión original, la encola para darle respuesta posteriormente.
 4. Si el proceso no ha comenzado y no se ha detectado la necesidad del mismo y se recibe una petición de un peer miembro de la sesión del tipo JoinSessionMessage, se asume entonces que el peer ha perdido conexión con el Maestro (bien sea por rango o por que el Maestro dejó de funcionar). Por lo tanto se verifica la conexión con el Maestro:
 - Si el peer local también perdió conexión con el Maestro, encola la petición como se describe en 3 y va a 1.
 - Si la comunicación con el Maestro sigue, responde a la petición del peer con respuesta negativa y termina.

3.2.3.4. Restricciones

A continuación se presenta un conjunto de restricciones directamente relacionadas con los requerimientos no funcionales que se han tenido en cuenta en el diseño y especificación del framework y sus protocolos:

1. Para el problema de los peers protegidos por herramientas como firewall o NATs y su participación en las sesiones de juego, no se ofrece ningún tipo de servicio de enrutamiento o bridging en Internet como soporte al framework. Sin embargo, la interacción con un peer Rendezvous que cumpla con las funciones de traducción de red privada a Internet o de una tecnología de transporte a otra, está contemplado dentro del diseño del framework y los protocolos del mismo. La arquitectura del peer Rendezvous y sus interacciones con otros peers Rendezvous no se contempla en esta especificación. **RNF1 y RNF2.**
2. El descubrimiento de peers, en cuanto a su dirección física de red, es responsabilidad de la implementación del DiscoveryService. Esto se debe a que la capa de transporte a utilizar en determinados casos provee el servicio de descubrimiento, como es el caso de Bluetooth. Para cuando se considere una implementación sobre TCP/IP u otra tecnología de transporte, el descubrimiento de peers en la red local deberá ser definido por la implementación. Igual es el caso para el monitoreo de peers por

medio del monitoreo de sus direcciones físicas. **RNF1 y RNF2.**

3. Este no es un framework para construir aplicaciones P2P de propósito general. En efecto, se concentra en proveer las funcionalidades necesarias para los juegos multiusuario en ambientes P2P en redes de área local. A pesar que la especificación introduce el concepto de un peer Rendezvous como el mecanismo para cubrir varias tecnologías y redes, no se hace una especificación detallada de su funcionamiento en los diversos escenarios. **RNF3.**

3.2.3.5. Descubrimiento y Monitoreo de Peers

Tal como se muestra en la sección anterior, el descubrimiento y monitoreo de peers en cuanto a su dirección física es responsabilidad de la implementación del servicio Discovery.

La solución comúnmente utilizada para resolver este problema es la utilización de servidores de directorio. Cada aplicación entonces tiene una lista de servidores en el código a los que va cuando desea iniciar los procesos de descubrimiento, otras alternativas son multicast o broadcast. Este acercamiento, sin embargo, no es considerado oportuno dado que los escenarios para los que el framework es creado son aquellos en los que la red es local y no necesariamente con conectividad hacia Internet.

Esta característica de la especificación, genera entonces la preocupación entorno a la interoperabilidad de las diferentes implementaciones del framework sobre la misma tecnología de transporte. Frente a esto se asume que la solución a este problema en las diversas tecnologías será tomada de facto llevando a que se adopte la solución definida por la implementación más fuerte. **RF3, RF4, RF21 y RF22.**

3.2.4. Patrones de Diseño

Los patrones de diseño son soluciones conocidas a problemas conocidos. La utilización de patrones en el diseño del framework tiene dos objetivos. El primero asegura que no tengamos que resolver problemas en donde soluciones conocidas ya han sido previamente implementadas y probadas. El segundo objetivo es hacer el framework entendible por los usuarios de tal forma que sea fácil de modificar aumentando su adaptabilidad y extensibilidad.

A continuación mostraremos los patrones de diseño utilizados en el diseño del framework.

3.2.4.1. Patrón Singleton

Según [25], el patrón singleton es utilizado cuando se desea asegurar que existe una y sólo una instancia de determinada clase. Esto se logra a través de métodos estáticos y haciendo que la clase sea quien cree la instancia. Los casos de aplicación de este patrón son:

- Cuando debe existir exactamente una instancia de la clase. En este caso la instancia debe ser accesible desde un punto bien conocido, por ejemplo `Clase.getInstance()`.
- Cuando los usuarios de acuerdo a la situación externa necesiten utilizar una subclase de la instancia. Por lo que la clase es la encargada de proveer al usuario con la instancia de la subclase correcta dependiendo de la situación.

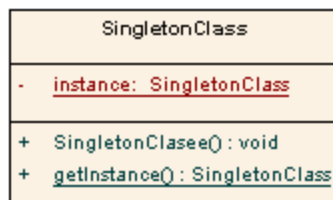


Figura16: Patrón Singleton.

En nuestro framework se usa el patrón singleton en cada una de las clases de los servicios (Connection Service, DiscoveryService, GroupService, P2PNetworkService) y en la clase Session. Los servicios permiten que el framework sólo pueda tener una instancia de cada servicio en un momento dado buscando atomicidad de operaciones y consistencia del estado de cada servicio, además de un reducido consumo de memoria. Según el diseño hecho del framework un usuario sólo puede pertenecer a una sesión de juego, por lo que la clase sesión es un singleton, además de la facilidad para la sincronización de la misma.

3.2.4.2. Patrón Observador

De acuerdo a [23] el patrón observador es utilizado cuando se quiere notificar a uno o más objetos acerca de cambios ocurridos en un objeto central. Estos cambios están relacionados principalmente con el cambio de los valores encapsulados por el objeto.

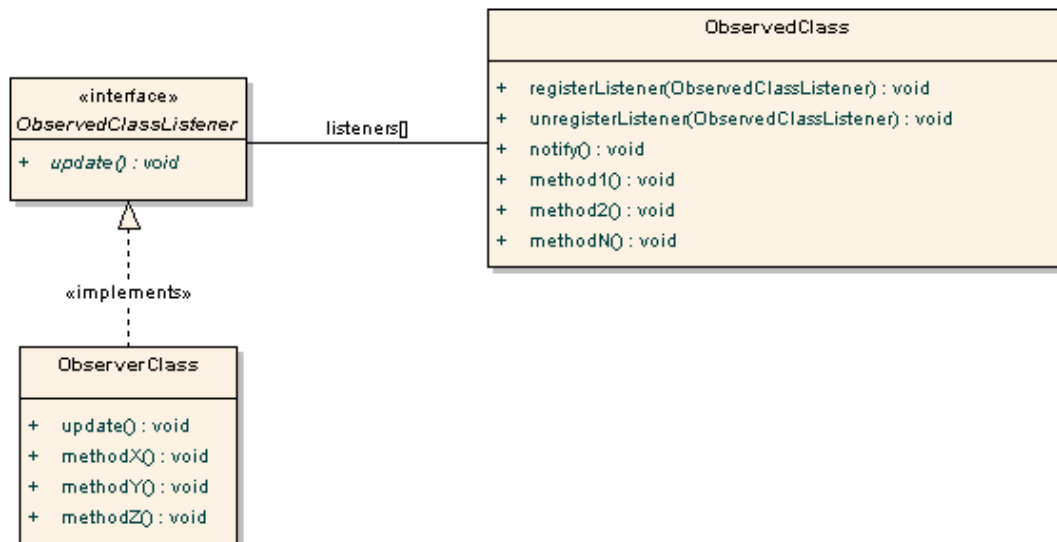


Figura17 : Patrón Observador.

En el framework hemos utilizado el patrón observador en una situación:

- Después de un proceso de sincronización la instancia de la clase Session es actualizada. Esta actualización puede resultar en nuevos peers en la sesión y peers que abandonaron la misma. Estos cambios se notificaban por medio de la interfase P2PNetworkListener y los métodos peerJoined () y peerLeft (). Cabe notar que a pesar de ser dos métodos en vez de un solo método llamado update, estos métodos cumplen con la función de notificación.

3.2.4.3. Patrón Subscriber Publisher

Este patrón es muy similar al patrón observador. La diferencia es más de fondo: en el patrón observador la notificación se hace acerca del cambio en los valores de los atributos de un objeto. En el patrón Publisher suscribir las notificaciones son acerca de la aparición de nuevos atributos/ objetos que no necesariamente existían previamente con otro valor. El ejemplo más claro es la llegada de un mensaje.

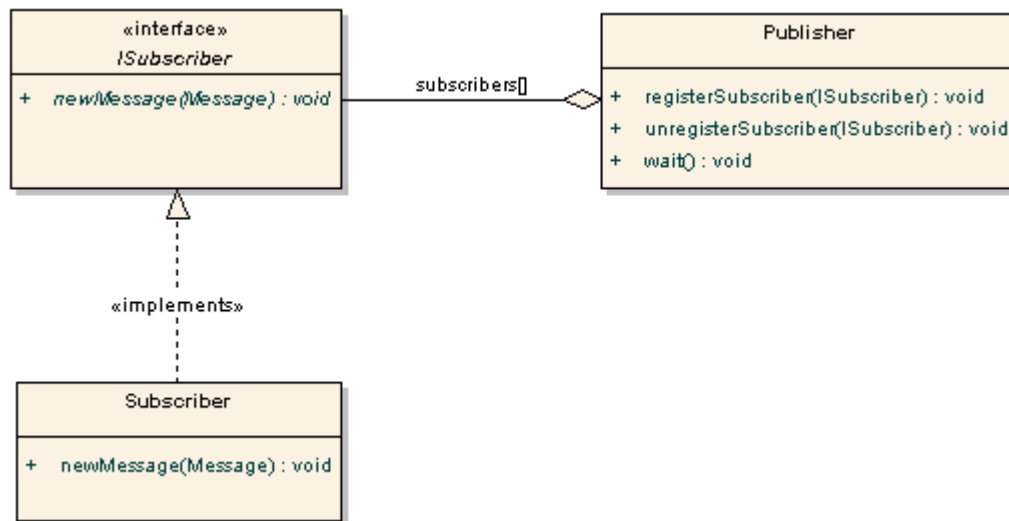


Figura18 : Patrón Publisher - Subscriber.

En el framework hemos utilizado el patrón Publisher subscriber en varias situaciones, algunas de las mismas son:

- Cuando el ConnectionService detecta un cambio en la dirección de alguno de los peers pertenecientes a la sesión notifica de este cambio por medio de la interfase PeerAddressChangeListener y el método processAddressChangeListener().
- Cuando el ConnectionService recibe datos del medio, él los interpreta como messages o responses, luego notificará por medio de la interfase ConnectionListener y los métodos processMessage(ConnectionMessage) y processResponse(ConnectionResponse) según sea el caso.
- Cuando el DiscoveryService está haciendo la búsqueda de sesiones de juego, notifica al P2PNetworkService acerca de las nuevas sesiones encontradas por medio de la interfase ServiceDiscoveryListener y el método sessionFound().

3.2.5. Servicios del Framework

El GA P2P Network Framework está compuesto por cuatro servicios. Estos cuatro servicios proveen las funcionalidades identificadas en los requerimientos para los juegos multiusuario P2P (ver requerimientos detallados en la definición de cada uno de los servicios) de la siguiente forma:

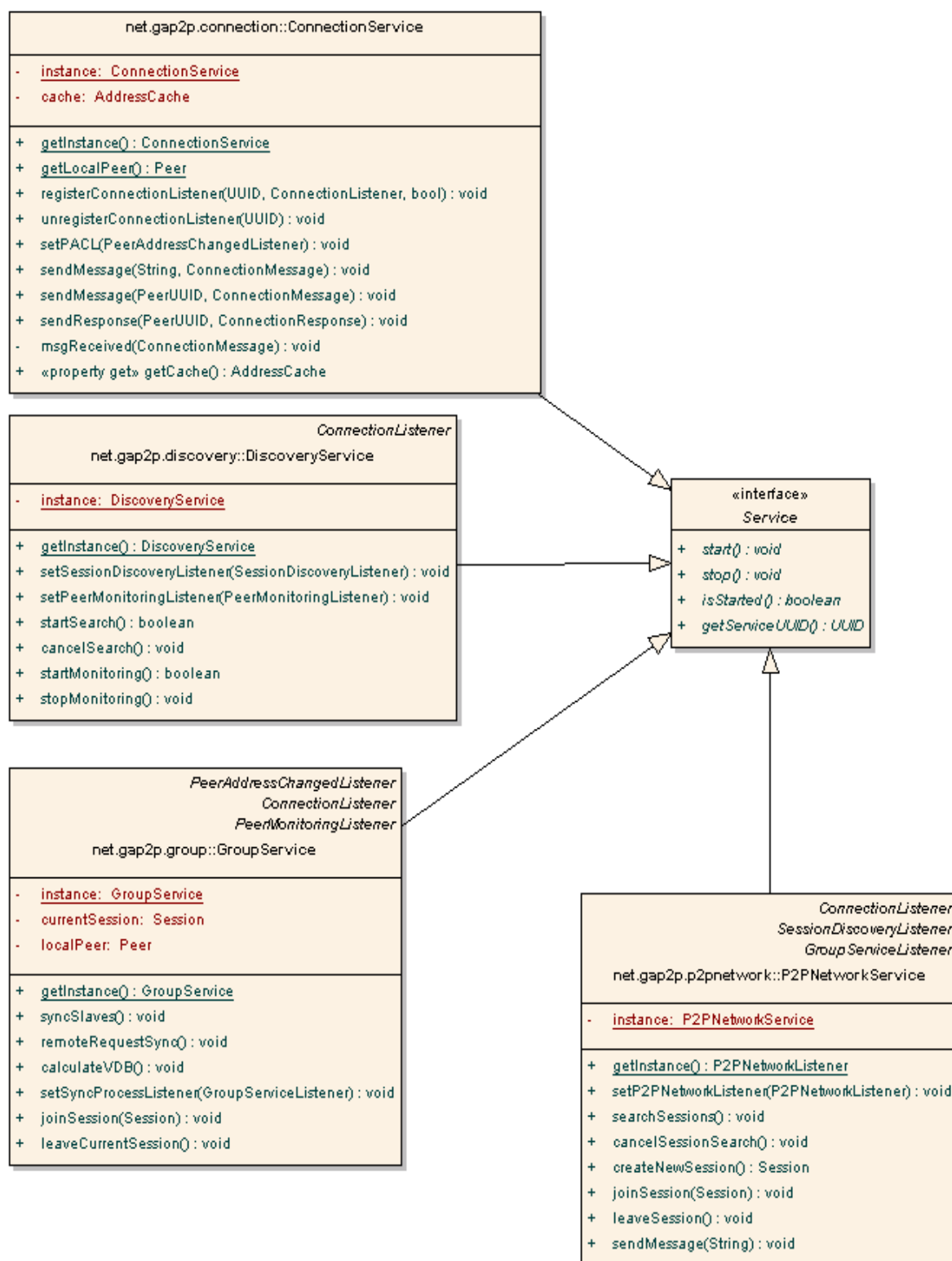


Figura19 : Framework Services.

- **ConnectionService:** Este es el servicio básico del framework. Tiene la responsabilidad de enviar los mensajes desde el peer local hasta un destino. El envío se hace utilizando la tecnología de transmisión de la plataforma. Utiliza un wrapper a ese nivel llamado el Connection Wrapper(CW) para hacer el encapsulamiento, envío y entrega de los mensajes que recibe de servicios en capas superiores. Cuando el peer local tiene el rol de maestro es este servicio quien hace el reenvío de mensajes a los demás peers que participan en la sesión (ver imágenes 3 y 4). También provee un mecanismo para el registro de los demás servicios que hacen uso del ConnectionService para permitir la entrega local de mensajes.
- **DiscoveryService:** Este servicio permite el descubrimiento de sesiones de juego y el monitoreo de peers. Como primera instancia en el descubrimiento de sesiones, efectúa un descubrimiento de peers. Las funcionalidades de descubrimiento y monitoreo de peers no se especifican en los protocolos propuestos dada la alta dependencia con la tecnología de transporte de red para estos procesos. Una vez descubiertos los peers se procede al descubrimiento de sesiones y esto se hace a través del protocolo Discovery Protocol (DP).
- **GroupManagementService:** Tiene como objetivo fundamental la coordinación del grupo de peers que pertenecen a una misma sesión y esto lo hace ejerciendo control de acceso a la sesión y de un mecanismo de sincronización. El control de acceso a la sesión se hace a través de mensajes simples del Group Management Protocol (GMP). La sincronización se da a nivel de objetos del framework y variables de juego. Los objetos del framework tienen una definición de "schema" dentro del conjunto de protocolos, el cual es utilizado para su intercambio. Su serialización, deserialización y actualización es realizada por el framework mismo. La sincronización de variables de juego se hace a través de la serialización de una colección de valores que se pasan desde el juego al framework por medio de los eventos. Todo el proceso de sincronización se hace por medio de mensajes del GMP.
- **P2PNetworkService:** Este es el servicio que sirve como punto de entrada y de salida al software cliente del framework. Este servicio encapsula las funcionalidades de los demás servicios y ofrece puntos de acceso a las mismas. También encapsula el envío y recepción de mensajes desde el juego hacia los demás peers en mensajes del CW.

Cada servicio se define en términos de clases e interfaces, buscando la mayor homogeneidad entre las diversas implementaciones del mismo. Además de los métodos y propiedades aquí propuestas para estas clases e interfaces, la implementación puede tener otros elementos adicionales.

3.2.5.1. Connection Service

Este es el servicio más interno del framework y tiene la responsabilidad de enviar los mensajes desde el peer local hasta un peer de destino. Este envío se hace utilizando la tecnología de transmisión de la plataforma, por lo que este servicio debe encapsular todos los procesos de creación de sockets, administración de memoria, entre otros, propios de la plataforma y la tecnología de transmisión. Este servicio implementa los siguientes requerimientos:

Código:	Requerimiento:
RF1	El framework debe permitir establecer y terminar conexiones, de acuerdo a la tecnología de transporte, con otros peers.
RF2	El framework debe permitir a un peer remoto establecer conexiones con el peer local.
RF3	El framework debe proveer al juego independencia de la tecnología de transporte subyacente.
RF4	El framework debe proveer encapsulamiento de las direcciones físicas de los peers al juego y a los servicios de alto nivel del framework mismo.
RF5	El framework debe detectar y controlar los cambios de direcciones físicas e interfaces que un peer remoto pueda utilizar.
RF6	El framework debe hacer notificación sobre las fallas en la capa de red.
RF7	El framework debe reciclar las conexiones de tal forma que se minimice el número de veces que una conexión tiene que ser abierta.
RF12	El framework permite el registrar servicios a la capa de transporte.
RF13	El framework entrega los mensajes entrantes al servicio que vaya dirigido.
RF14	El framework debe hacer notificación a la aplicación cliente sobre eventos en el mismo.
RF15	El framework debe poder identificar el peer origen de un mensaje.
RF16	El framework debe permitir enviar mensajes a todos los peers.
RF17	El framework debe permitir enviar mensajes a un peer en particular.

Implementa un wrapper llamado el Connection Wrapper (CW) para hacer el encapsulamiento, envío y entrega de los mensajes que recibe de servicios en capas superiores. Cuando el peer local tiene el rol de Maestro es este servicio quien hace el reenvío de mensajes a los demás peers que participan en la sesión (ver imágenes 3 y 4). También provee un mecanismo para el registro de los demás servicios que hacen uso del ConnectionService para permitir la entrega local de mensajes por medio del ServiceUUID.

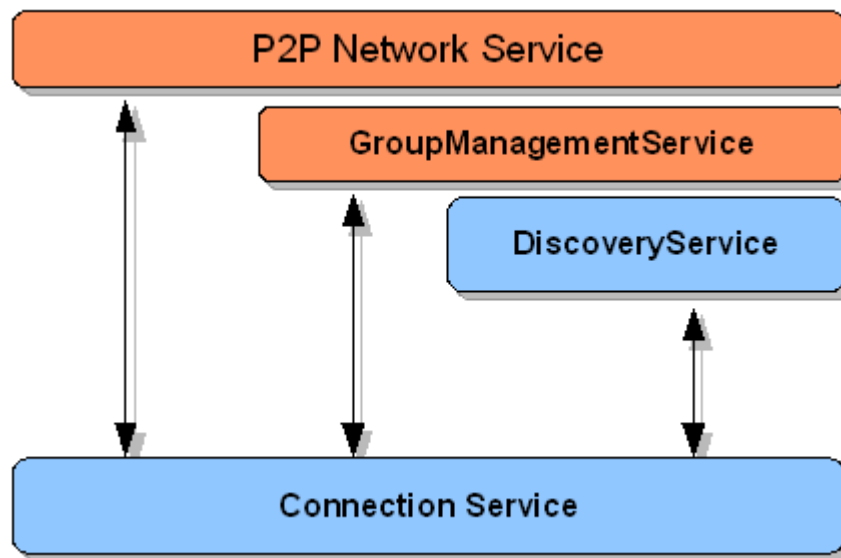


Figura20: Connection Service Building Blocks.

El ConnectionService está compuesto por cuatro clases/interfaces que juntas brindan las funcionalidades requeridas:

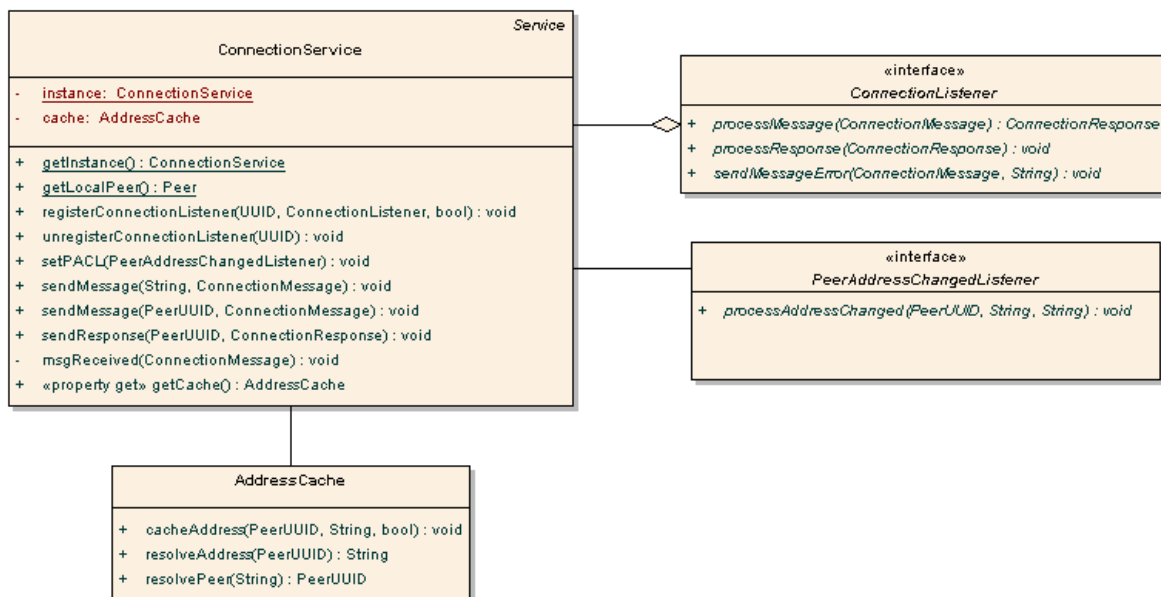


Figura21: Clases e interfaces que componen el ConnectionService.

- **ConnectionService:** Es la clase principal que implementa la clase Service y funciona como un singleton. Cuando se inicializa el servicio esta clase comenzará a escuchar por mensajes dirigidos al peer local. Al recibir algún mensaje del tipo ConnectionMessage, reconoce el identificador ServiceUUID del mensaje y hace la entrega al servicio correspondiente por

medio de un `ConnectionListener`. Lo anterior implica que los servicios deben tener registrado su `ServiceUUID` en el `ConnectionService` para que los mensajes dirigidos al servicio puedan ser entregados. Al registrar el `ConnectionListener` es necesario hacer explícitos dos parámetros: El `UUID` que identifica el servicio y un 'bool', el cual en caso que el peer local sea Maestro, deberá reenviar los mensajes recibidos para ese servicio a los demás miembros de la sesión.

- **ConnectionListener:** Es la interfaz que es llamada para hacer la entrega de los mensajes recibidos. El método `processMessage()` devuelve un `ConnectionResponse`, el cual deberá ser enviado, sin embargo, este valor puede ser nulo, en este caso el `ConnectionResponse` no se envía. Cuando el `ConnectionService` falla al realizar el envío de un mensaje llama el método `sendMessageError`.
- **AddressCache:** Este es un contenedor en donde el peer local almacenará los pares dirección/PeerUUID. De esta forma, en todo el ámbito del framework los peers serán identificados por el PeerUUID independientemente de su dirección física. Es responsabilidad del `ConnectionService` determinar la dirección física de un peer y mantener actualizados los pares dirección/PeerUUID.
- **PeerAddressChangedListener:** Como parte del proceso de mantener actualizados los pares dirección/PeerUUID, al recibir cada mensaje se validará el par en el `AddressCache`. Si la dirección física asociada a un peer ha cambiado entonces el `ConnectionService` llama el método `processAddressChanged()` de esta interfaz. En caso que el peer notificado sea un peer Maestro, este llamado desencadena un proceso de sincronización.

El `ConnectionService` utiliza un wrapper llamado el `Connection Wrapper(CW)`, compuesto por:



Figura22: Mensajes de Connection Wrapper.

- **ConnectionMessage:** Este mensaje es enviado desde un peer a uno o más peers. Se envía a un servicio a quien será transferido el mensaje una vez sea recibido por el ConnectionService en el peer remoto. El servicio de destino es claramente identificado con un UUID de servicio. Este mensaje puede o no generar mensajes de respuesta. Sin embargo, el peer que envía debe asumir que no se generará ninguno.
- **ConnectionResponse:** Es el mensaje de repuesta a un ConnectionMessage y es enviado desde cualquiera de los peers que recibió del ConnectionMessage hacia el peer de origen. Puede ser enviado por uno o más peers de los que recibieron el ConnectionMessage, pero sólo uno por peer. El envío de este mensaje de respuesta es asíncrono.

3.2.5.2. Discovery Service

Este servicio tiene la responsabilidad de implementar y proveer las funcionalidades de descubrimiento y monitoreo de peers. Al igual que el ConnectionService muchos detalles de este servicio dependen ampliamente en la plataforma de implementación del framework. Este servicio implementa los

siguientes requerimientos:

Código:	Requerimiento:
RF3	El framework debe proveer al juego independencia de la tecnología de transporte subyacente.
RF5	El framework debe detectar y controlar los cambios de direcciones físicas e interfaces que un peer remoto pueda utilizar.
RF6	El framework debe hacer notificación sobre las fallas en la capa de red.
RF14	El framework debe hacer notificación a la aplicación cliente sobre eventos en el mismo.
RF18	El framework permite realizar descubrimiento de peers que ejecutan instancias del juego.
RF19	El framework debe permitir descubrir diferentes sesiones de juego ejecutándose en el momento.
RF21	El framework hace monitoreo de los peers pertenecientes a la sesión en términos de la posibilidad de establecer conexión con cada uno de ellos.
RF22	El framework realiza notificaciones en cuanto a los cambios de visibilidad hacia los demás peers pertenecientes a la sesión.

El DiscoveryService hace uso del Discovery Protocol (DP) para hacer el descubrimiento de sesiones de juego. Sin embargo, los procesos de descubrimiento y monitoreo de peers no se detallan en esta especificación debido a la alta dependencia que tienen con la plataforma. Por ejemplo, para el caso de Bluetooth, esta misma tecnología provee los mecanismos de descubrimiento y monitoreo, para el caso de LAN/Ethernet se pueden implementar por medio de grupos UDP y un protocolo simple de "echo". Por lo tanto, se deja abierta la posibilidad para que la implementación sobre cierta plataforma bien acoja alguna otra propuesta anterior o proponga un nuevo mecanismo para implementar estos servicios.

Estos protocolos son el Peer Discovery Protocol (PDP) y el Peer Monitoring Protocol (PMP) y bien pueden ser definidos como el resto de protocolos del framework o pueden mapearse a herramientas de la tecnología.

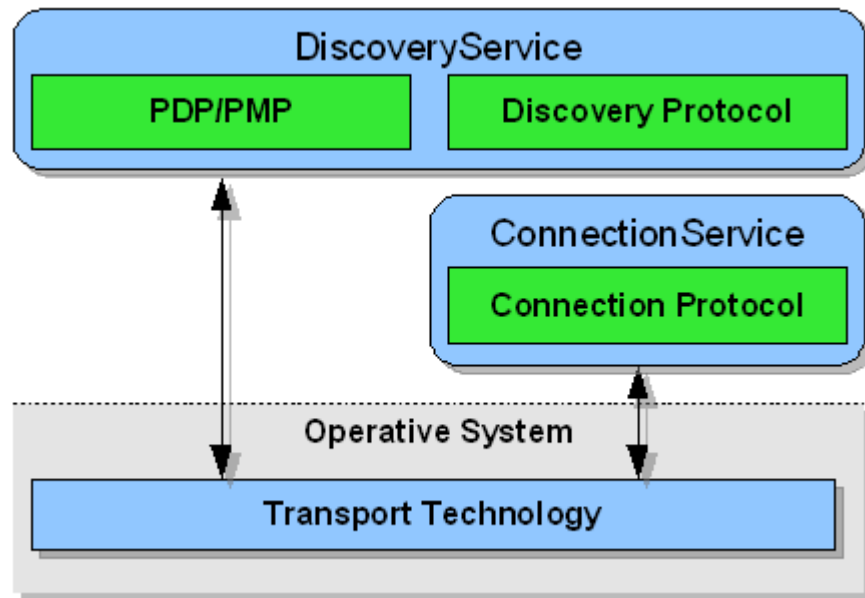


Figura23 : Discovery Service Building Blocks.

Las siguientes interfaces/ clases participan en la construcción lógica del servicio:

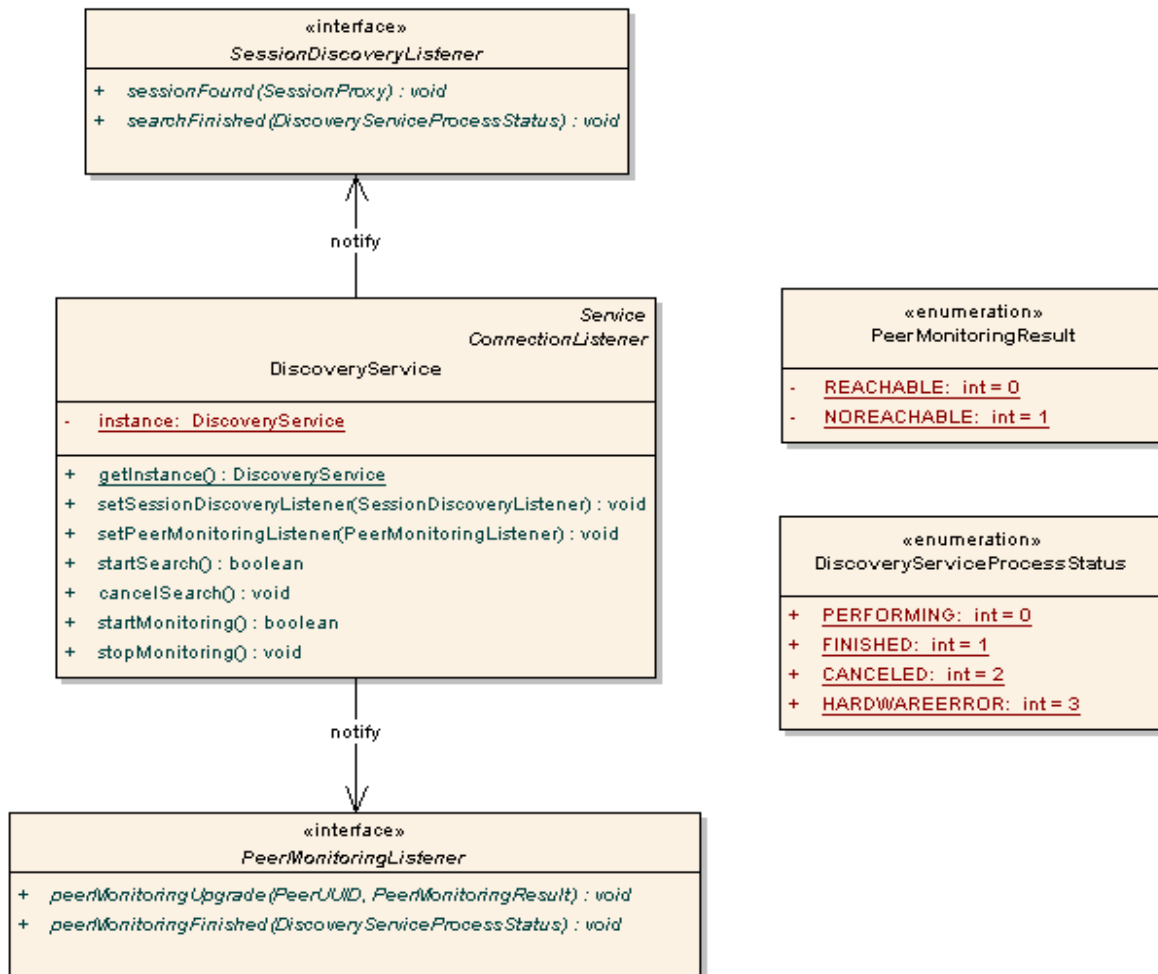


Figura24: Clases e interfaces que componen el DiscoveryService.

- **DiscoveryService:** Esta es la clase principal del servicio que implementa Service y funciona como un singleton. También implementa la interfaz ConnectionListener y por lo tanto este singleton está registrado en la clase ConnectionService con el UUID de servicio para el DiscoveryService. Las dos funcionalidades principales, el descubrimiento de sesiones y el monitoreo de peers, pueden ser accedidas a través de los métodos startSearch() y startMonitoring() respectivamente.
- **SessionDiscoveryListener:** Un objeto cuya clase implementa esta interfaz debe ser entregado como parámetro en el método startSearch() del DiscoveryService, y es este objeto quien será notificado acerca de las nuevas sesiones que se encuentren y de la finalización del proceso de búsqueda de sesiones.
- **PeerMonitoringListener:** Un objeto cuya clase implementa esta interfaz debe ser entregado como parámetro en el método startMonitoring() del DiscoveryService, y es este objeto quien será notificado acerca de las

condiciones de visibilidad y posibilidad de conexión a los diferentes peers miembros de la sesión.

- **PeerMonitoringResult:** Esta clase encapsula la enumeración de posibles resultados a obtener en el proceso de monitoreo de peers. El resultado puede ser REACHABLE significando que es posible establecer una conexión con el peer y en caso contrario NOREACHABLE.
- **DiscoveryServiceProcessStatus:** Esta clase al igual que la anterior realiza el encapsulamiento del estado del proceso de descubrimiento de sesiones. En un momento dado, un proceso de descubrimiento puede estar PERFORMING (ejecutándose), FINISHED (terminado), CANCELED (cancelado) o de presentarse algún problema de hardware HARDWAREERROR (error de hardware).

El Discovery Protocol es el único de los protocolos del DiscoveryService que es especificado, esto debido a que los demás protocolos del DiscoveryService es dependiente de la red en la cual este implementado. Este protocolo está compuesto por:

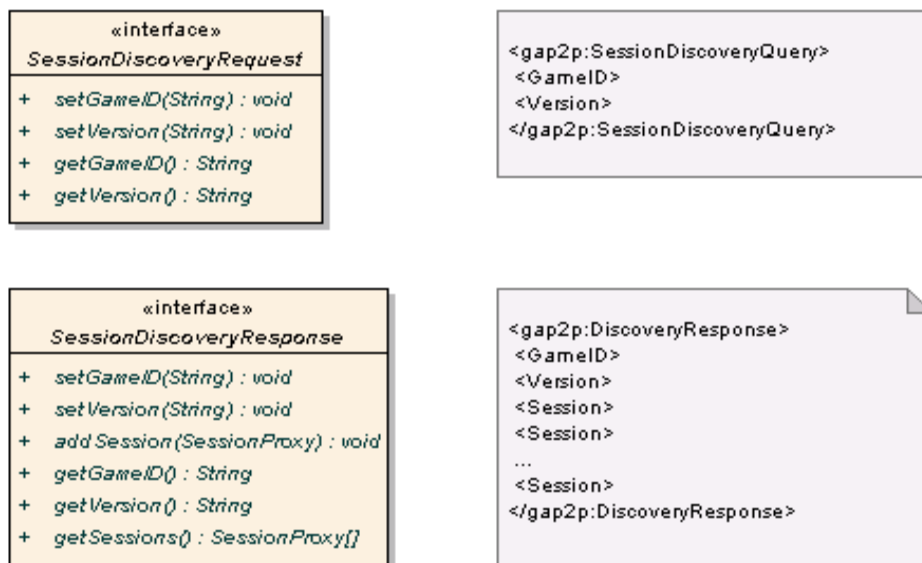


Figura25: Mensajes del Discovery Protocol.

- **SessionDiscoveryRequest:** Para la petición sobre información de la sesión se utiliza el presente mensaje. Cada peer responderá acerca de su propia sesión. Si existe un peer Rendezvous este se encargará de propagar el mensaje de descubrimiento a los demás peers que conozca y enrutará de vuelta las respuestas a la petición en un sólo mensaje. La petición utiliza los valores UUID de juego y versión para asegurar que sean clientes compatibles.

- **SessionDiscoveryResponse:** Este mensaje es generado por cualquier peer como respuesta a un SessionDiscoveryRequest. Este mensaje es encapsulado dentro de un ConnectionResponse. En el caso de peers normales, se devuelve información acerca de ellos mismos solamente. En el caso de un peer Rendezvous, este generará un response con información acerca de todos los peers que conoce.
- **SessionProxy**²⁰: Esta clase es un descriptor de la sesión, conteniendo información acerca del identificador de la sesión, el peer Maestro y todo el conjunto de peers que hacen parte de la misma. Los peers y el Maestro se definen bajo la misma estructura y el orden de los peers es el mismo orden de tiene el vector del dueño del balón en ese momento.
- **PeerProxy**²¹: Esta clase es un descriptor de un peer, conteniendo tres valores fundamentales: el nick del peer, su identificador de peer y su dirección física.

3.2.5.3. Group Management Service

Este servicio se concentra en dos conjuntos de funcionalidades: las relativas al ingreso / salida de un peer a la sesión, y las relativas a la sincronización. Este servicio hace plena utilización del ConnectionService para el paso de mensajes. Este servicio implementa los siguientes requerimientos:

Código:	Requerimiento:
RF5	El framework debe detectar y controlar los cambios de direcciones físicas e interfaces que un peer remoto pueda utilizar.
RF8	El framework debe permitir la creación de sesiones de juego.
RF9	El framework debe permitir entrar a una sesión de juego existente.
RF10	El framework debe permitir a los jugadores de una sesión aceptar nuevos jugadores que desean entrar a hacer parte de la misma.
RF11	El framework debe permitir a los jugadores de una sesión rechazar nuevos jugadores que desean entrar a hacer parte de la misma.
RF14	El framework debe hacer notificación a la aplicación cliente sobre eventos en el mismo.
RF22	El framework realiza notificaciones en cuanto a los cambios de visibilidad hacia los demás peers pertenecientes a la sesión.
RF23	El framework debe proveer un mecanismo de sincronización de objetos de control y variables de juego.
RF24	El framework administra y sincroniza la información necesaria para garantizar la continuidad de la sesión de juego.

²⁰No aparece en la figura.

²¹No aparece en la figura.

RF25	El framework notifica y garantiza la continuidad de la sesión cuando un peer sale de la misma.
------	--

El GroupManagementService hace uso del Group Management Protocol (GMP). Este protocolo utiliza toda la infraestructura del ConnectionService para el envío y recepción de peticiones, mensajes y respuestas. Cabe notar también que el GroupManagementService es quien implementa las interfaces PeerAddressChangeListener y PeerMonitoringListener, dado que es quien puede encontrar estos mensajes útiles para reorganizar la sesión e iniciar procesos de sincronización.

Este servicio establece la membresía de los peers a la sesión y quien coordina el proceso de continuidad de la sesión ante la ausencia del Maestro, tal como se explica anteriormente en este documento.

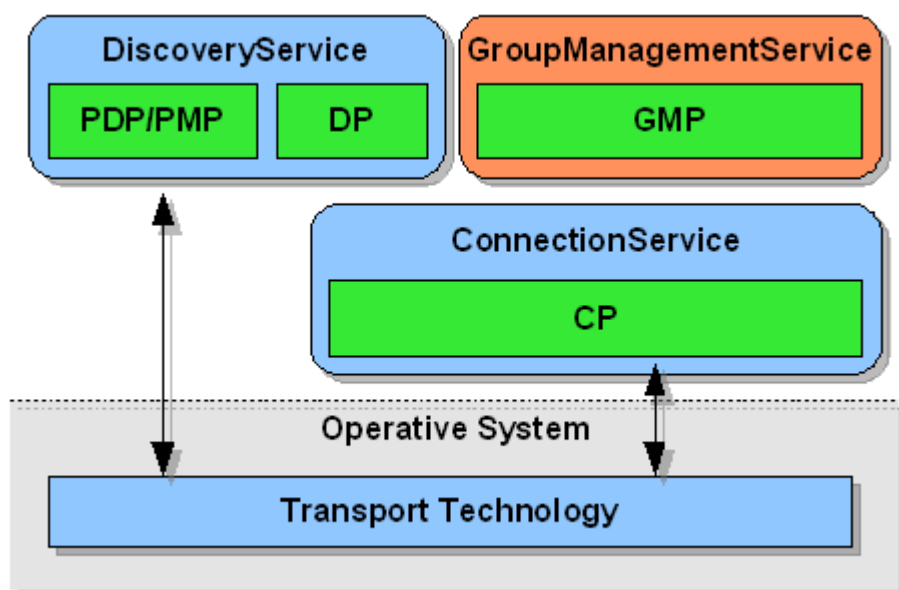


Figura 26 (26): Group Management Service Building Blocks.

Las siguientes interfaces/ clases participan en la construcción lógica del servicio:

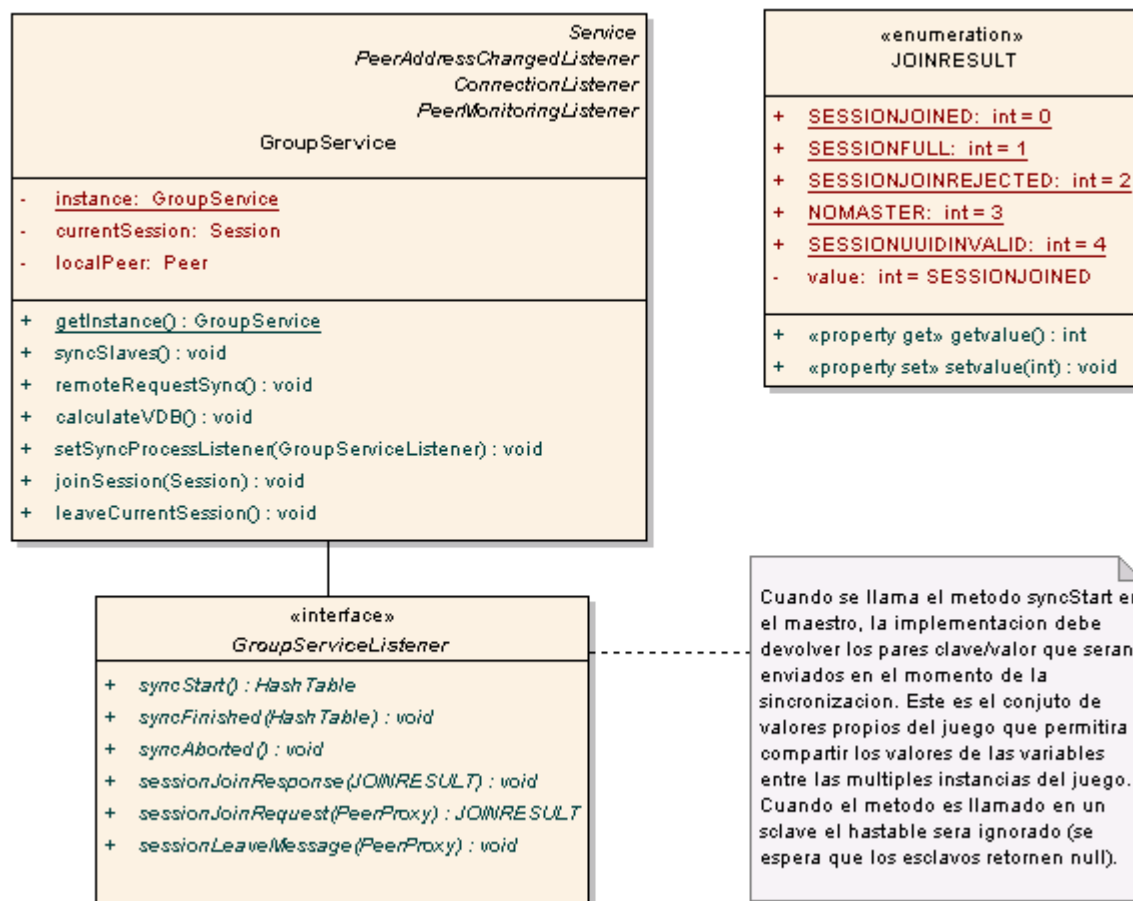


Figura 27 (27): Clases e interfaces que componen el GroupManagementService.

- **GroupService:** Esta es la clase principal del servicio que implementa *Service* y es un singleton. Esta clase a su vez implementa las interfaces *PeerMonitoringListener* y *PeerAddressChangedListeners*, las cuales le notifican acerca de los cambios de visibilidad hacia los miembros del grupo y acerca del cambio en la dirección física de los miembros del grupo. En ambos casos estas notificaciones podrían provocar el inicio de un proceso de sincronización. También ofrece las funcionalidades para reorganizar el vector del dueño del balón, ingresar y salir de la sesión.
- **GroupServiceListener:** Esta es la interfaz utilizada por el *GroupService*, para hacer las notificaciones pertinentes en cuanto a los procesos de sincronización, y la entrada y salida de nuevos usuarios a la sesión.
- **JOINRESULT:** Esta es una enumeración que contiene los posibles valores con los que se responde a una petición de ingresar a la sesión.

Este servicio hace uso del Group Management Protocol (GMP), el cual está compuesto por:

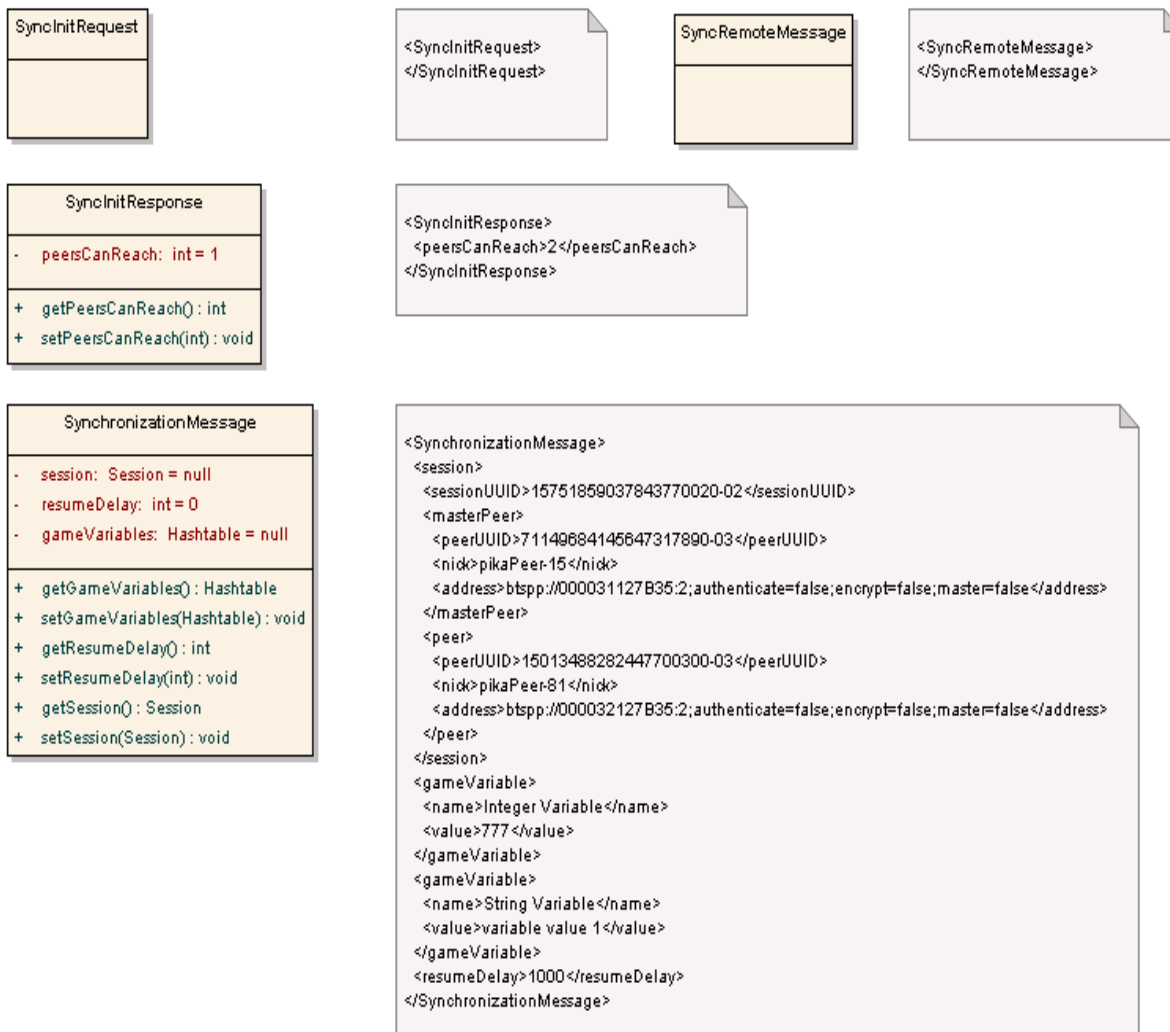


Figura28: Mensajes del Group Management Protocol – Part 1.

- **SynchronizationInitRequest:** Este mensaje es una petición realizada por el Maestro a cada uno de los peers de la sesión. El objetivo es comenzar el proceso de sincronización en el que se intercambia información acerca de los peers, del vector del dueño del balón y ciertos valores del dominio del juego que permiten asegurar que todos los peers tienen el mismo estado desde el momento de la sincronización. Al momento de enviar este mensaje se inicia un contador que permite calcular, al momento de recibir una respuesta, de manera aproximada el tiempo de transmisión. Este tiempo de transmisión permitirá a los peers volver al estado de ejecución aproximadamente en el mismo momento.
- **SynchronizationInitResponse:** Este mensaje es enviado por los peers al Maestro. Tiene dos propósitos: 1) Servir para terminar de calcular el RTT para esta pareja Maestro / esclavo, y 2) Proporcionar parámetros para

calcular el ordenamiento del vector del dueño del balón. En este momento sólo se considera para dicho cálculo el número de peers que un peer puede alcanzar.

- **SynchronizationMessage:** Este es el mensaje que contiene toda la información a sincronizar entre los peers de la sesión: variables del framework y variables del juego. En la Especificación de Protocolos se verá una descripción más detallada.
- **SynchronizationRemoteMessage:** Este es un mensaje enviado desde cualquiera de los Esclavos hacia el Maestro. El propósito de este mensaje es pedir al Maestro que comience un proceso de sincronización dado que el Esclavo ha tenido cambios que podrían afectar el ordenamiento en el vector del dueño del balón. También puede hacerse la petición desde la implementación del juego en el Esclavo para obtener los valores de las variables compartidas del juego.



Figura29: Mensajes del Group Management Protocol - Part 2.

- **JoinSessionRequest:** Este mensaje es enviado por un Esclavo al Maestro de la sesión. Busca gestionar el ingreso del nuevo Esclavo a la sesión y al ser una petición genera una respuesta. La decisión de dejar ingresar al nuevo jugador a la sesión de juego es tomada por la aplicación, y en la mayoría de las veces por el usuario del peer Maestro.
- **JoinSessionResponse:** Este es el mensaje de respuesta que se genera ante una petición de ingreso a sesión. Este mensaje es generado por el peer que ha recibido la petición encapsulado en un mensaje response del CW. Este mensaje debe especificar el resultado de la petición de ingreso a

la sesión y puede tomar varios valores de los de la enumeración JOINRESULT.

- **LeaveSessionMessage:** Este es un mensaje de notificación enviado por un peer al Maestro cuando el peer se retira de la sesión. Este mensaje no genera un response.

3.2.5.4. P2P Network Service

Este servicio tiene como función principal servir de punto de contacto único del framework con la implementación del juego. A través de este servicio el juego inicializa los servicios de red P2P, descubre sesiones, ingresa a salas de juego o crea sesiones nuevas y principalmente envía y recibe mensajes hacia y desde los demás peers de la sesión.

Por su naturaleza este servicio implementará o se apoyará en otros servicios para cubrir todos los requerimientos mostrados anteriormente. Este servicio entonces no define ni implementa un protocolo mas se apoya en los demás servicios. Para el caso del envío y recepción de mensajes del juego entre peers los encapsula directamente en mensajes del CW el cual esta implementado en el ConnectionService.

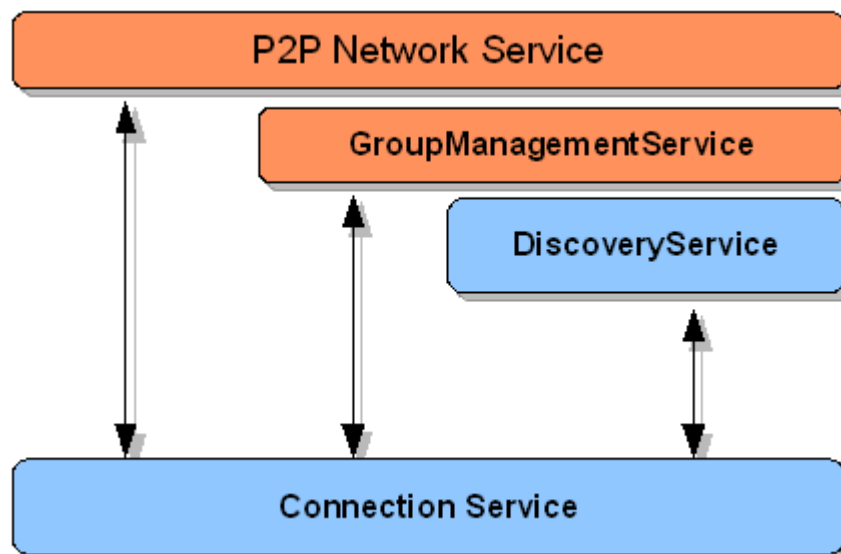


Figura30: P2P Network Service Building Blocks.

Las clases que participan en la construcción de este servicio son:

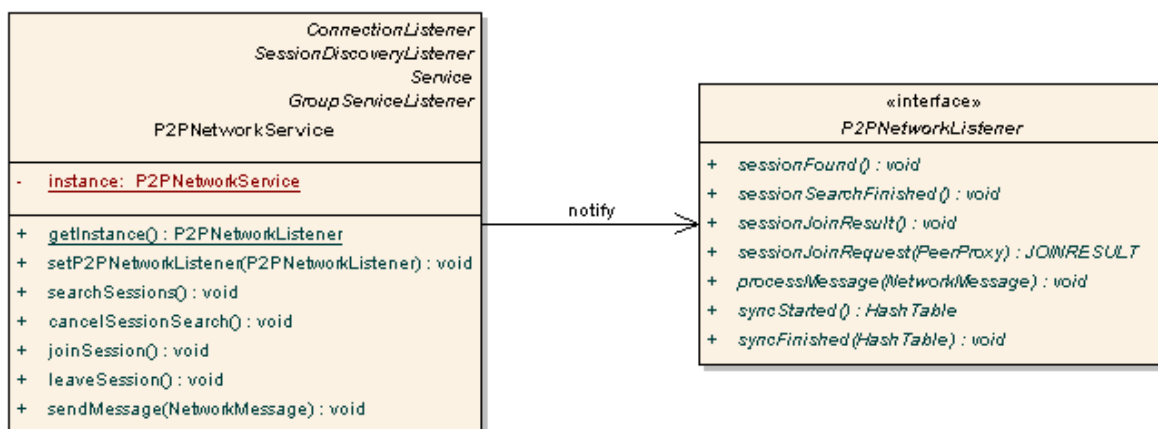


Figura31: Clases e interfaces que componen el P2PNetworkService.

- **P2PNetworkService:** Esta es la clase principal del servicio y funciona como un singleton. El juego ejecutara los llamados a los servicios por medio de esta como punto central.
- **P2PNetworkListener:** A través de esta interfaz el framework se comunica con el juego, para la entrega de mensajes, las notificaciones sobre el paso a modo de sincronización y el paso a modo normal y las peticiones y respuesta de ingreso a una sesión. También sobre los resultados del proceso de descubrimiento de sesiones.

Para este servicio no se define un protocolo. Sin embargo se hace uso de la siguiente clase para facilitar el paso de mensajes:

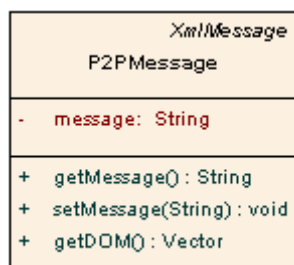


Figura32: Clase utilizada para intercambiar mensajes a nivel del P2P Network Service.

- **P2PMessage:** Este mensaje encapsula una cadena de texto que es intercambiada entre instancias del juego en diferentes peers. Los mensajes a enviar son entregados por el juego al framework encapsulados por un objeto de esta clase e igualmente el framework entrega los mensajes recibidos al juego como objetos de esta clase.

3.2.6. Componentes Estructurales del Framework

A continuación se presenta un conjunto de clases que determinan la estructura básica del framework:

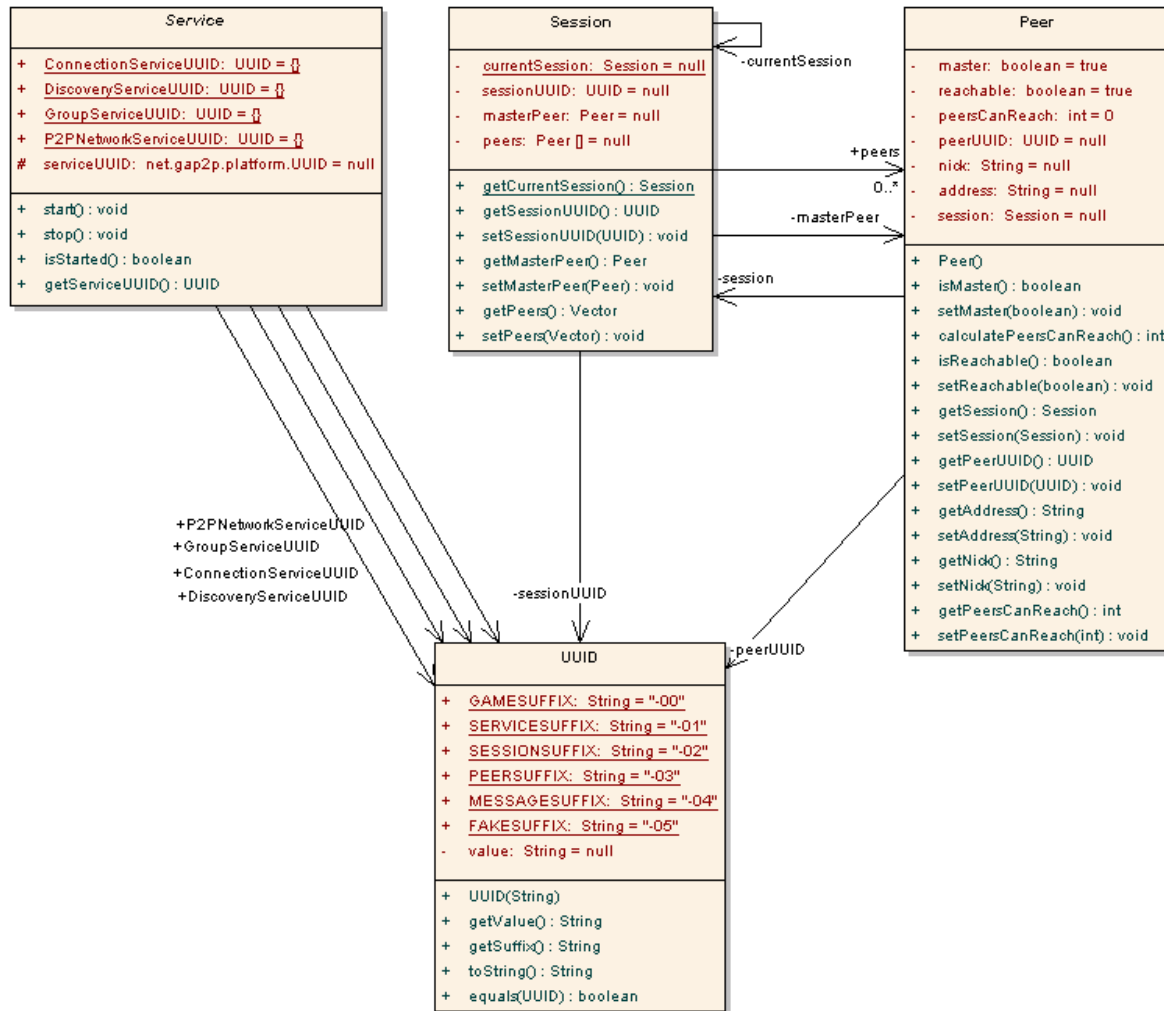


Figura33: Clases básicas estructurales del framework.

- **Service:** Interfaz que define dos importantes métodos para un servicio: el método `start()` permitirá al servicio enviar y recibir mensajes y ejecutar lógica del negocio. El método `stop()` causará que el servicio deje de ejecutar las funciones iniciadas con el método `start()`. El llamado a estos métodos es sincrónico.
- **UUID:** Interfaz para una clase que encapsule un UUID (ver diseño del framework, diagrama conceptual, UUID).
- **Session:** Esta clase implementa la interfaz `SessionProxy` del `DiscoveryService` proveyéndole la información que esta requiere y

encapsulando lo demás. También permite acceder a la sesión actual.

- **Peer:** Esta clase implementa la interfaz PeerSession del DiscoveryService proveyéndole la información que esta requiere y encapsulando lo demás. También permite acceder a la instancia del peer local.

3.2.6. Relaciones Entre Servicios

Los servicios del framework ofrecen sus funcionalidades a los otros. La forma como estos servicios se relacionan con los demás se mostrara a continuación por medio de una tabla de relaciones y diversos diagramas de secuencia a nivel de servicios.

3.2.6.1. Tabla de Relaciones

La siguiente tabla muestra de forma estática las relaciones entre los servicios del framework. Para cada servicio, de muestran dos tipos de relaciones con los demás. En la primera columna (Nombre) se presenta el servicio cliente del servicio en análisis. En la segunda columna (Funcionalidad Ofrecida) aparecen las funcionalidades del servicio en análisis utilizadas por el servicio cliente. En la tercera columna (Mensaje Enviado) aparecen las notificaciones que recibe el servicio cliente. Estas notificaciones son por medio de interfaces “*Listener*” implementadas por el servicio cliente y matriculadas en el servicio en análisis.

Nombre	Funcionalidad Ofrecida	Mensaje Enviado
Connection Service		
Discovery Service	- sendMessage(dirfisica)	- processMessage() - processResponse() - messageWasSent() - sendMessageError()
Group Management Service	- sendMessage(uuidpeer)	- processMessage() - processResponse() - messageWasSent() - sendMessageError() - processAddressChange()
P2P Network Service	- start() - stop() - sendMessage(uuidpeer)	- processMessage() - processResponse() - messageWasSent() - sendMessageError()
Discovery Service		
Connection Service		
Group Management Service	- startMonitoring()	- peerMonitoringUpgrade()

	- stopMonitoring()	- peerMonitoringFinished()
P2P Network Service	- start() - stop() - startSearch() - cancelSearch()	- sessionFound() - searchFinished()
Group Management Service		
Connection Service		
Discovery Service		
P2P Network Service	- start() - stop() - createNewSession() - joinSession() - leaveCurrentSession()	- sessionJoinResponse() - sessionJoinRequest() - sessionLeaveMessage() - syncStarted() - syncFinished() - syncAborted() - recoveryStarted() - recoveryFinished()
P2P Network Service		
Connection Service		
Discovery Service		
Group Management Service		

3.2.6.2. Relaciones Dinámicas

A continuación se muestran la manera como se relacionan los servicios del framework en algunos escenarios. El objetivo de estos diagramas de secuencia es dar al lector una idea de la dependencia y causalidad entre las funcionalidades ofrecidas por los servicios al juego y entre ellos.

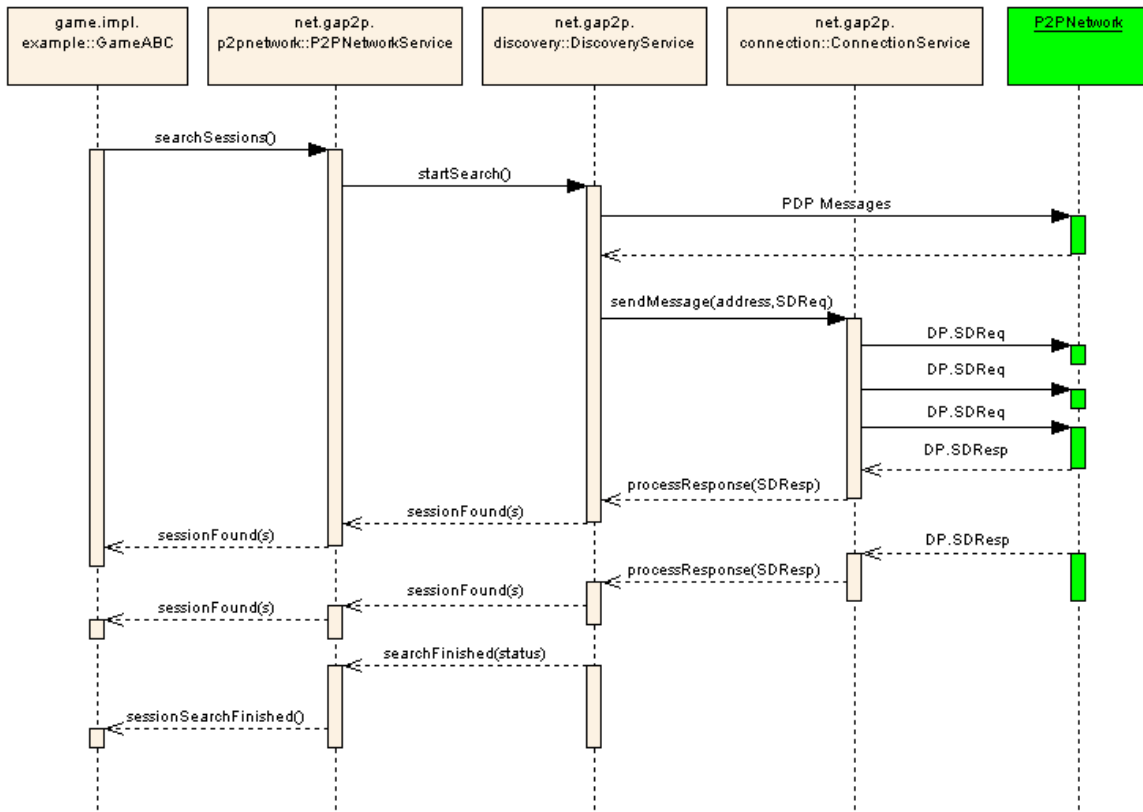


Figura34: Session Search - Nuevo Dispositivo.

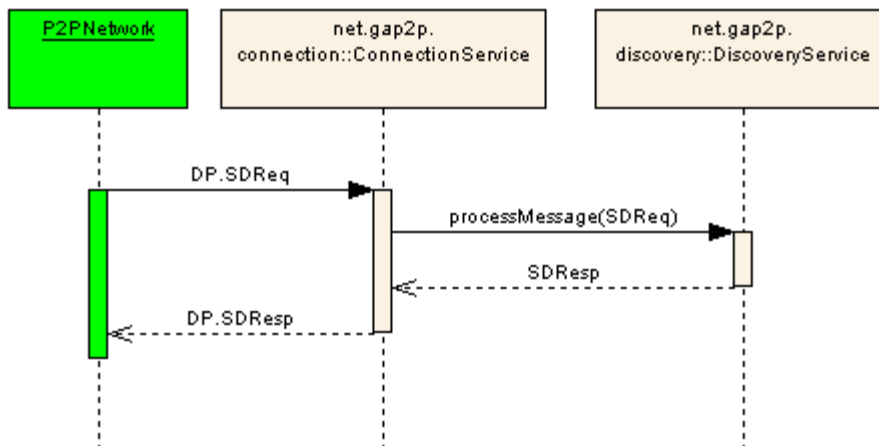


Figura35: Session Search - Dispositivo que hace parte de la red P2P.

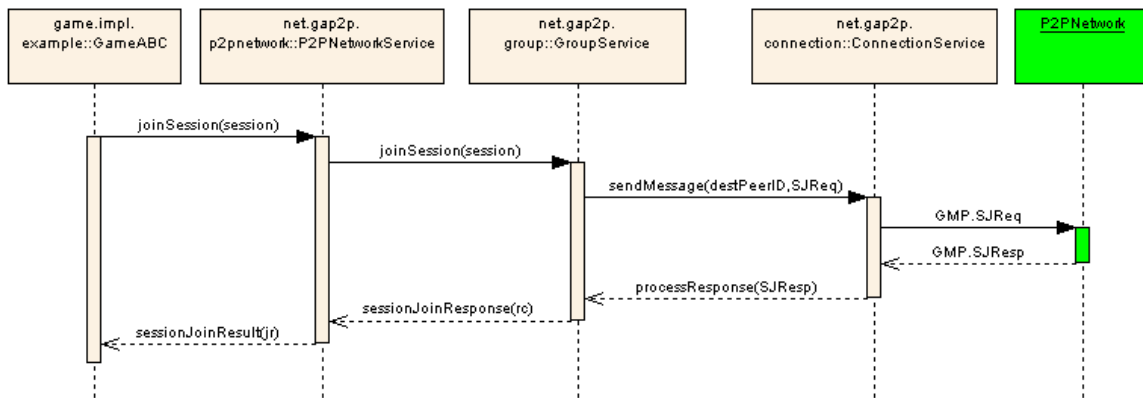


Figura36: Join Session Request - Nuevo Dispositivo.

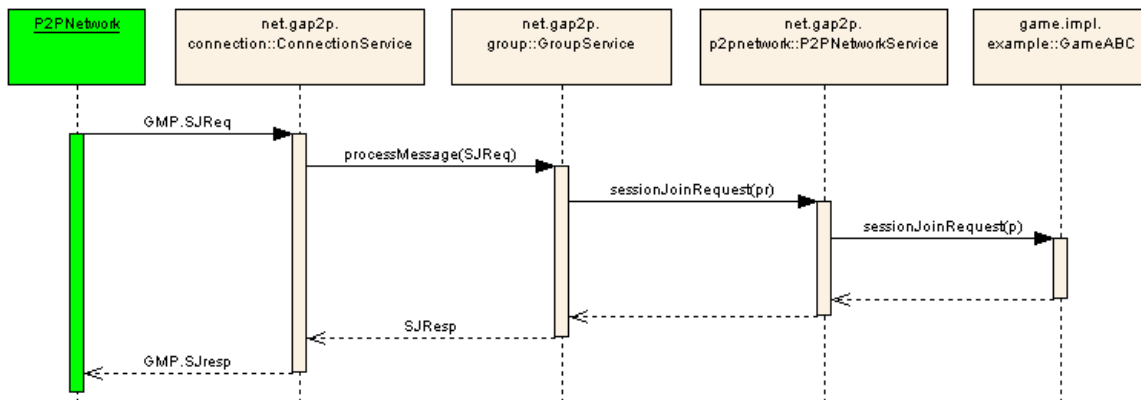


Figura37: Join Session Request - Maestro de la sesión.

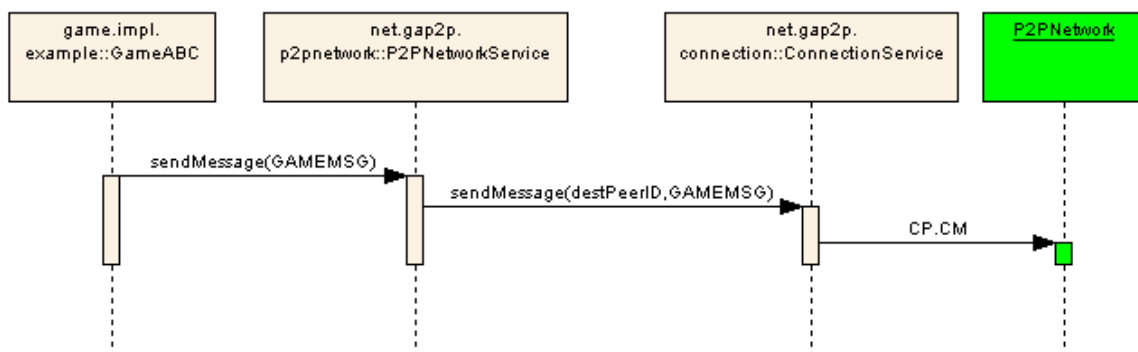


Figura38: Mensaje de juego - Dispositivo Esclavo.

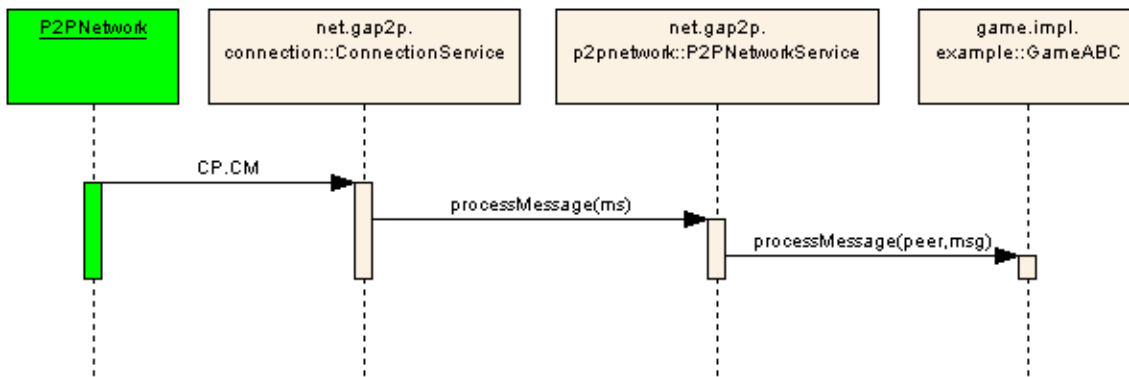


Figura39: Recepción Mensaje de Juego - Dispositivo Esclavo.

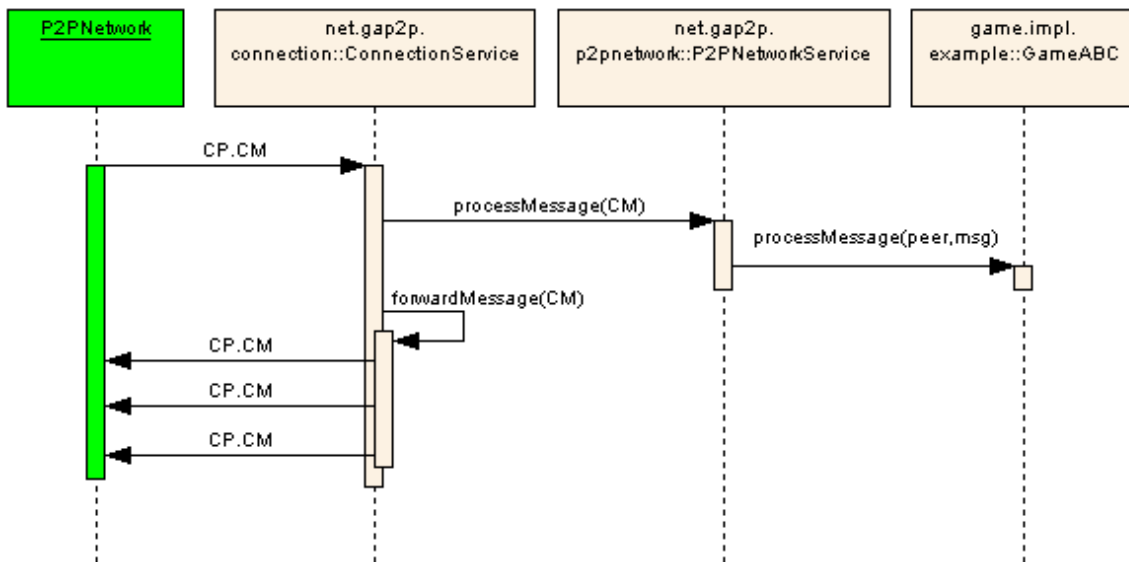


Figura40: Recepción Mensaje de Juego - Dispositivo Maestro.

3.2.7. Protocolos del Framework

En esta sección se presenta la definición de los protocolos utilizados como base para el framework **GAP2P Network Framework**. El diseño lógico de la solución señala cuatro servicios a ser prestados por el framework y cada uno de estos servicios se basa en un protocolo subyacente. El protocolo Discovery Protocol (DP) asiste en las funciones de descubrimiento de peers y sus propiedades y monitoreo del contexto. Por último el protocolo Group Management Protocol

(GMP) está encargado del control de acceso y salida de los peers a las sesiones, igual que del proceso de sincronización de información.

Los dos protocolos del GA P2P Network Framework han sido diseñados para comunicaciones ad-hoc, pervasivas, las cuales son comunicaciones que se dan en este tipo de redes, buscando la conformación de grupos autoorganizados que puedan garantizar la continuidad de las sesiones de juego multiusuario sin importar las salidas o pérdidas de los diferentes participantes en la sesión. Sin la necesidad de tener una infraestructura centralizada para el control de las sesiones y los grupos (como en la mayoría de las soluciones P2P de hoy en día).

Estos protocolos no hacen ninguna suposición sobre la tecnología de transporte que esta siendo utilizada. Sin embargo, propiedades de las mismas junto con las características de la implementación del framework que se utilice pueden tener implicaciones serias en el desempeño del juego. Estas implicaciones pueden incluso crear una experiencia de usuario muy negativa o hacer el juego inviable.

Todos estos protocolos hacen uso de mensajes XML para su implementación buscando usar un estándar del mercado que nos permita hacer una implementación transparente en otras tecnologías, y se encuentra la mayor cantidad de veces los mensajes de CW encapsulando los de todos los protocolos. Sin embargo, cada protocolo es independiente de los otros, hay posibilidades de agregar nuevos protocolos si se ve necesario solo a que especificar este, como los aquí definidos.

La especificación aquí presentada permite que los protocolos sean implementados sobre enlaces de bajo costo y desempeño (no se necesita mucha capacidad de red para que estos sean ejecutados), en algunos casos unidireccionales. Todo esto con la idea de poder cubrir la mayor cantidad de dispositivos que quieran hacer parte de las sesiones de juego. Sin embargo no sobra aclarar:

- Para que dos peers puedan hacer parte de la misma sesión de juego es necesario que los dos peers tengan implementaciones compatibles del framework sobre la misma tecnología de transporte y que estén al alcance el uno del otro. Si estas condiciones no se cumplen, entonces es necesario que existan peers intermedios dedicados explícitamente a la traducción desde una tecnología de transporte a otra. Estos son los peers denominados **Rendezvous**.
- Para una eficiente comunicación en el marco de los juegos multiusuario, es recomendable que la implementación provea canales bi-direccionales o en su defecto tiempos de respuesta óptimos para la elevada tasa de mensajes que sea necesario enviar y recibir desde un peer.

UUID – Universal Unique Identifier

Los identificadores UUID, son cadenas de texto que identifican una entidad de manera única dentro de todo el alcance de la solución. Los UUID deben ser generados de forma tal que la posibilidad de generar dos UUID iguales sea extremadamente baja. Se define como una cadena de 20 caracteres de notación hexadecimal más el identificador de tipo. El identificador de tipo hace explícita la utilización que se le está dada a un UUID específico. En este momento hay definidos cuatro tipos de identificadores. El identificador de tipo se separa de los 20 caracteres principales por medio de un guión '-'.

Los tipos son:

Juego: 00

Servicio: 01

Sesión: 02

Peer: 03

Mensaje: 04

Por ejemplo un identificador de juego puede ser: 159B108E38159B108E38-00, mientras que un identificador de mensaje podría ser: 40ACD8847140ACD88471-04.

<XSD para el UUID>

Los mensajes en el protocolo son enviados por servicios en capas lógicas superiores a la del servicio Connection. Cada una de estas capas (llamadas Servicios), que hace uso de los mensajes Connection, debe estar registrada dentro del servicio de Connection y tener un identificador UUID de servicio. Así pues, cuando un mensaje llega al peer local, este indicará a que servicio va dirigido por medio del UUID de servicio. Todas las implementaciones de un servicio deben de compartir el mismo UUID de servicio el cual está definido en esta especificación:

- UUID Discovery Service: 00000000000000000000D-01
- UUID Group Management Service: 00000000000000000000G-01
- UUID GA P2P Network Service: 00000000000000000000GA-01

3.2.7.1. Discovery Protocol (DP)

Este es un protocolo para el descubrimiento de nuevas sesiones de juego y el monitoreo de peers. El descubrimiento y monitoreo de peers hacen uso extensivo

de la capa de transporte y sus mecanismos para suministrar dicha funcionalidad. Por lo que la definición de como deben funcionar estos mecanismos no se incluye dentro del protocolo.

Este protocolo está encargado del descubrimiento de sesiones de juego. Las demás funcionalidades prestadas por el servicio de DiscoveryService, son propias de la implementación, tal como se describió en Descubrimiento y Monitoreo de Peers, por lo que no se define protocolo para ellas en esta especificación

Este protocolo maneja un modelo de petición / respuesta, es no orientado a la conexión, half-duplex, de grupo, unicast, no bloqueante y asíncrono. Por lo tanto toda petición tiene una respuesta asociada. Y se encapsulan utilizando la funcionalidad de response de CW.

ESPECIFICACIÓN DEL SERVICIO

El protocolo implementa un servicio que incluye detección cambios en las direcciones físicas, notificaciones de cambios de visibilidad de un peer perteneciente a una sesión, descubrimientos de peers y sesiones de juego.

Sucesos entrantes

<i>Nombre</i>	<i>Interfaz</i>	<i>Significado</i>
—		
SERSESON	DPE	Se recibió una primitiva de servicio búsqueda de sesiones
TEXP	TIME	Expira el timer esperando SessionDiscoveryRequest
SDRESP	CWE	Se recibe un mensaje SessionDiscoveryRequest
—		

Estados

<i>Nombre</i>	<i>Significado</i>
—	
INAC	Inactivo
BUSSS	En espera de llegada de sesiones de juego.
—	

Sucesos salientes

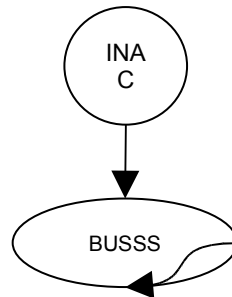
<i>Nombre</i>	<i>Interfaz</i>	<i>Significado</i>
—		
SENSMSG	DPS	Dar formato y mensaje de petición de sesión
—		

Acciones específicas

[1] = *Start-timer* (Iniciar el cronómetro) empleando la cola TIME

[2] = Cambiar Estado actual de la entidad de protocolo

[3] = *Stop-timer* (Parar el cronómetro) empleando la cola TIME



FORMATO DE LOS MENSAJES

Este protocolo está construido sobre el CW y lo utiliza para encapsular, enviar y entregar sus mensajes. Por lo tanto tiene un identificador de servicio registrado en la implementación del Connection Service. Dicho UUID de servicio es 00000000000000000000D-01. Todos los mensajes a nivel de CW que transportan mensajes de DP tienen en su identificador de servicio el anterior UUID.

Session Discovery Request

Al iniciar la aplicación y antes de ingresar a alguna sesión de juego, el usuario deseará conocer acerca de las sesiones de juego que están disponibles y posteriormente intentará tomar parte de ellas o iniciar una sesión por sí mismo. El proceso de descubrimiento de la sesión se divide en dos pasos: primero está el descubrimiento “físico” de los peers y luego se hará una petición a cada uno de los peers encontrados acerca del estado de su sesión. El descubrimiento “físico” es realizado directamente por la implementación y los detalles de su funcionamiento no se definen en esta especificación.

Para la petición sobre información de la sesión se utiliza el presente mensaje. Cada peer responderá acerca de su propia sesión. Si existe un peer Rendezvous este se encargará de propagar el mensaje de descubrimiento a los demás peers que conozca y enrutará de vuelta las respuestas a la petición en un sólo mensaje. Si el peer no pertenece a ninguna sesión entonces no responderá al mensaje.

```
<xs:element name="SessionDiscoveryRequest" type="gap2p:SessionDiscoveryRequest"/>
```

```
<xs:complexType name="SessionDiscoveryRequest">
```

```
<xs:sequence>
```

```
<xs:element name="GameUUID" type="gap2p:UUID" />
```

```
<xs:element name="GameName" type="xs:String" />
```

```
<xs:element name="Version" type="xs:String" />
```

```
</xs:sequence>
```

```
</xs:complexType>
```

<GameUUID>	Este es el identificador del juego. Generalmente está quemado en el código y es único entre todos los juegos que se construye sobre GA P2P Network. Se recomienda utilizar una función que genere el UUID a partir de un nombre mas representativo (159B108E38159B108E38-00 = transforma("PingPong")).
<GameName>	Este es un texto fácil de leer con el nombre del juego. Este texto es opcional y no necesariamente único.
<Version>	Identificador de la versión del juego que se está buscando. Puede ser nulo

Session Discovery Response

Este mensaje es generado por cualquier peer como respuesta a un SessionDiscoveryRequest. Este mensaje es encapsulado dentro de un ConnectionResponse. En el caso de peers normales, se devuelve información acerca de ellos mismos solamente. En el caso de un peer Rendezvous, este generara un response con información acerca de todos los peers que conoce.

```
<xs:element name="SessionDiscoveryResponse" type="gap2p:SessionDiscoveryResponse"/>
```

```
<xs:complexType name="SessionDiscoveryResponse">
```

```
<xs:sequence>
```

```
<xs:element name="GameUUID" type="gap2p:UUID" />
```

```
<xs:element name="Version" type="gap2p:String" />
```

```
<xs:element name="Session" type="gap2p:Session" min="1" max="unbound" />
```

```
</xs:sequence>
```

```
</xs:complexType>
```

<GameUUID>	Este es el identificador del juego. Generalmente está quemado en el código y es único entre todos los juegos que se construyan sobre GA P2P Network. Se recomienda utilizar una función que genere el UUID a partir de un nombre mas representativo (159B108E38159B108E38-00 = transforma("PingPong")).
<Version>	Identificador de la versión del juego que implementa el peer que responde. Puede ser nulo
<Session>	Este es el descriptor de la sesión del peer que genera el response. Los peers en este mensaje deben estar organizados de acuerdo al vector del dueño del balón.

```
<xs:complexType name="Session">
```

```
<xs:sequence>
```

```
<xs:element name="SessionUUID" type="gap2p:UUID" />
```

```
<xs:element name="MasterPeer" type="gap2p:Peer" />
```

```
<xs:element name="Peer" type="gap2p:Peer" min="0" max="unbound" />
```

```
</xs:sequence>
```

```
</xs:complexType>
```

<SessionUUID>	Este es el identificador de la sesión.
<MasterPeer>	Esta es la descripción del peer Maestro para dicha sesión.
<Peer>	Este es cada uno de los peers que hacen parte de la sesión. El orden en que aparezcan estos peers en el mensaje se asume como el orden del vector del dueño del balón

```
<xs:complexType name="Peer">
```

```

<xs:sequence>
  <xs:element name="PeerUUID" type="gap2p:UUID" />
  <xs:element name="Nick" type="xs:string" />
  <xs:element name="Address" type="xs:string" />
</xs:sequence>
</xs:complexType>

```

<PeerUUID>	Identificador del peer. Es independiente del cualquier dirección física
<Nick>	Apodo del peer. Puede ser escogido por el usuario o generado automáticamente por la implementación. La idea es que este será el nombre a mostrar en el juego al usuario durante el proceso de descubrimiento.
<Address>	Esta es la dirección física del peer. Esta dirección tiene sentido en el contexto de la implementación del servicio Connection.

3.2.7.2. Group Management Protocol (GMP)

Este protocolo permite, en primera instancia, controlar el acceso de nuevos peers a la sesión y su salida de la misma. También está encargado de garantizar la continuidad de las sesiones de juego a través de la sincronización de objetos, la administración del llamado “vector del dueño del balón” y la recuperación de la sesión ante la ausencia del Maestro, es no orientado a la conexión, half-duplex, de grupo, unicast, no bloqueante y asíncrono.

Este protocolo está construido completamente sobre el CW y tiene dos objetivos específicos: gestionar las operaciones de ingreso y salida de un peer a una sesión y hacer la sincronización de objetos entre los elementos de la sesión. Este segundo objetivo apunta a su vez a dos cosas: garantizar al máximo las condiciones de continuidad de la sesión al presentarse algún inconveniente con el Maestro a través del vector del dueño del balón y sirve como punto de sincronizado de variables de juego en orden de asegurar que todos los participantes “ven” lo mismo.

ESPECIFICACIÓN DEL SERVICIO

El protocolo implementa un servicio que incluye la creación de sesiones de juego, unirse a una sesión de juego existente, rechazo de petición de unión de sesiones de juego, sincronización de objetos de control y variables del juego y garantizar la continuidad de una sesión si un peer sale del juego.

Sucesos entrantes

<i>Nombre</i>	<i>Interfaz</i>	<i>Significado</i>
–		
JOINSESON	GME	Se recibe una primitiva de servicio de unión a una sesión.
LEAVESESON	GME	Se recibe una primitiva de salida de la sesión.
TEXP	TIME	Expira el timer esperando SessionJoinRequest
SJRESP	PNE	Se recibe un mensaje SessionJoinResponse
SYINIRESP	GME	Se recibe respuesta de inicio de info a sincronizar
SYNMSG	GME	Se recibe mensaje con las variables compartidas del juego
–		

Estados

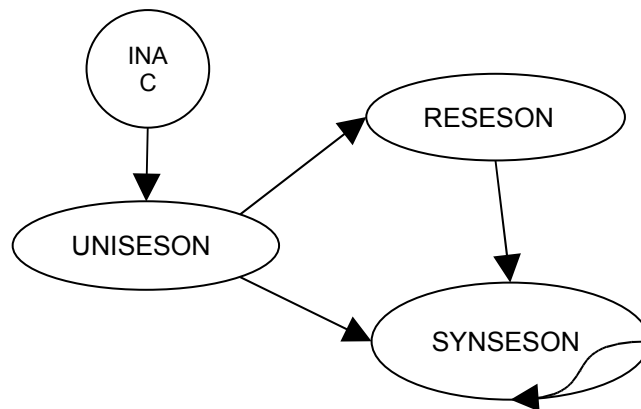
<i>Nombre</i>	<i>Significado</i>
–	
INAC	Inactivo
UNISESON	En sesión de juego
RESESON	Recuperando sesión de juego
SYNSESON	Sincronizando datos de la sesión de juego
–	

Sucesos salientes

<i>Nombre</i>	<i>Interfaz</i>	<i>Significado</i>
–		
SENSMSG	GMS	Dar formato y mensaje de petición de unión a la sesión
SYINIREQ	GMS	Se inicia envío de info a sincronizar
SYNRETMSG	GMS	Se envía mensaje para actualizar info a sincronizar
–		

Acciones específicas

- [1] = *Start-timer* (Iniciar el cronómetro) empleando la cola TIME
- [2] = Cambiar Estado actual de la entidad de protocolo
- [3] = *Stop-timer* (Parar el cronómetro) empleando la cola TIME



FORMATO DE LOS MENSAJES

Al ser un protocolo enteramente construido sobre el CW, posee un UUID de servicio que es 00000000000000000000G-01. Lo que significa que todos los mensajes de CW tendrán dicho identificador en el campo servicio y que ambas implementaciones de CW tendrán registrado un servicio que atiende los mensajes recibidos con dicho identificador.

Join Session Request

Este mensaje es enviado por un cliente al Maestro de la sesión. Busca gestionar el ingreso del nuevo Esclavo a la sesión y al ser una petición genera un response. La decisión de dejar ingresar al nuevo jugador a la sesión de juego es tomada por la aplicación, y en la mayoría de las veces por el usuario del peer Maestro.

```
<xs:element name="SessionJoinRequest" type="gap2p:SessionJoinRequest"/>
```

```
<xs:complexType name="SessionJoinRequest">
```

```
<xs:sequence>
```

```
<xs:element name="SessionUUID" type="gap2p:UUID" />
```

```
<xs:element name="Peer" type="gap2p:Peer" />
```

```
</xs:sequence>
```

```
</xs:complexType>
```

<SessionUUID>	Identificador de la sesión a la que se quiere ingresar. Si la sesión no es la del peer Maestro que está recibiendo la petición esta es
---------------	--

	rechazada.
<Peer>	Este campo es utilizado para mostrar información acerca de quien hace la petición de ingreso al usuario del peer Maestro.

Join Session Response

Este es el mensaje de respuesta que se genera ante una petición de ingreso a sesión. Este mensaje es generado por el peer que ha recibido la petición encapsulado en un mensaje response del CW. Este mensaje debe especificar el resultado de la petición de ingreso a la sesión y puede tomar varios valores:

- 0: Session Joined.
- 1: Session Full.
- 2: Session Join Rejected. Este es el valor utilizado cuando el usuario del peer Maestro ha rechazado explícitamente la petición de ingreso.
- 3: No Master. Este valor es utilizado cuando el peer que recibió la petición de ingreso no es el peer Maestro de la sesión.
- 4: Session UUID Invalid.

```
<xs:element name="SessionJoinResponse" type="gap2p:SessionJoinResponse"/>
```

```
<xs:complexType name="SessionJoinResponse">
```

```
<xs:sequence>
```

```
<xs:element name="JoinRequestResult" type="xs:integer" />
```

```
</xs:sequence>
```

```
</xs:complexType>
```

<JoinRequestResult>	Este es el valor explicado anteriormente.
---------------------	---

Leave Session Message

Este es un mensaje de notificación enviado por un peer al Maestro cuando el peer se retira de la sesión. Este mensaje no genera un response.

```

<xs:element name="SessionLeaveMessage"
type="gap2p:SessionLeaveMessage"/>

<xs:complexType name="SessionLeaveMessage">

<xs:sequence>

</xs:sequence>

</xs:complexType>

```

Synchronization Init Request

Este mensaje es una petición realizada por el Maestro a cada uno de los peers de la sesión. El objetivo es comenzar el proceso de sincronización en el que se intercambia información acerca de los peers, del vector del dueño del balón y ciertos valores del dominio del juego que permiten asegurar que todos los peers tienen el mismo estado desde el momento de la sincronización.

Esta petición sólo puede ser generada por el Maestro de la sesión, en cualquier otro caso el mensaje será desechado. Este mensaje tiene dos propósitos: 1) entrar en modo de sincronización en el que no se deben de enviar ningún otro tipo de mensajes, 2) junto con el mensaje de respuesta, determinar el RTT para esta pareja Maestro/Esclavo y así determinar el valor la variable "resumeDelay" del mensaje de sincronización.

Este mensaje de petición es vacío.

```

<xs:element name="SynclnitRequest" type="gap2p:SynclnitRequest"/>

<xs:complexType name="SynclnitRequest">

<xs:sequence>

</xs:sequence>

</xs:complexType>

```

Synchronization Init Response

Este mensaje es enviado por los peers al Maestro. Tiene dos propósitos: 1) servir para terminar de calcular el RTT para esta pareja Maestro/Esclavo, y 2) proporcionar parámetros para calcular el ordenamiento del vector del dueño del

balón En este momento sólo se considera para dicho cálculo el número de peers que un peer puede alcanzar.

Al ser recibido este response, el Maestro para el contador que había iniciado para este peer. Y determina el RTT para la comunicación con el Esclavo De allí deduce el tiempo que se demorará aproximadamente el mensaje de sincronización en ser enviado y entregado y este tiempo se le resta al tiempo de resumeDelay.

```
<xs:element name="SyncInitResponse" type="gap2p:SyncInitResponse"/>
```

```
<xs:complexType name="SyncInitResponse">
```

```
<xs:sequence>
```

```
<xs:element name="PeersCanReach" type="xs:integer" />
```

```
</xs:sequence>
```

```
</xs:complexType>
```

<PeersCanReach>	Este es el número de peers a los que el peer puede conectarse por línea directa. Esto tiene mucho sentido en redes inalámbricas de corto rango como Bluetooth en donde no necesariamente puede alcanzar a ver a todos los peers de que están dentro de la sesión. De allí que se utilice este parámetro para calcular el orden del vector del dueño del balón y así asegurar que se tenga la máxima continuidad de la sesión de juego.
-----------------	--

Synchronization Message

Este es el mensaje que contiene toda la información a sincronizar entre los peers de la sesión:

- Variables del framework: esta es la información de la sesión y de los peers que participan en ella. Es especialmente importante sincronizar esta información dado que muchos de los valores tienden a cambiar y necesitan estar actualizados en todos los peers de la sesión. El valor del identificador de la sesión puede cambiar cada vez que se hace un cambio de Maestro. Las direcciones físicas de los peers miembros de la sesión pueden cambiar además de los nuevos peers que pueden haber entrado a la sesión. Y por

último el vector del dueño del balón pudo haber sido modificado en su orden por el Maestro.

- Variables del juego: Estas son las variables que el juego desea sincronizar. Esta es información que solo concierne a la implementación del juego y es la implementación del juego la encargada de su interpretación.

El mensaje además contiene el valor en milisegundos para que cada uno de los peers salga del estado de sincronización y vuelva a modo de juego. Este valor es el que se calcula en base al valor por defecto menos el RTT/2 para el peer específico.

```
<xs:element name="SynchronizationMessage" type="gap2p:SynchronizationMessage"/>
```

```
<xs:complexType name="SynchronizationMessage">
```

```
<xs:sequence>
```

```
<xs:element name="Session" type="gap2p:Session" />
```

```
<xs:element name="GameVariable" type="gap2p:GameVariable" min="0" max="unbound" />
```

```
<xs:element name="ResumeDelay" type="xs:integer" />
```

```
</xs:sequence>
```

```
</xs:complexType>
```

<Session>	Este es el descriptor de la sesión del peer que genera el response. Los peers en este mensaje deben de estar organizados de acuerdo al vector del dueño del balón
<GameVariable>	Este es el conjunto de variables que el juego desea sincronizar en todos los peers. Este elemento es completamente opcional pero se recomienda que los juegos implementen ciertos controles en la sincronización de sus objetos haciendo uso de este mecanismo.
<ResumeDelay>	Este es el tiempo en milisegundos que el peer que recibe el mensaje debe esperar antes de salir del estado de sincronización y volver al estado de juego.

```
<xs:complexType name="GameVariable">
```

```

<xs:sequence>
  <xs:element name="Name" type="xs:string" />
  <xs:element name="Value" type="xs:string" />
</xs:sequence>
</xs:complexType>

```

<Name>	Este es el nombre que identifica el valor que se está sincronizando. Este nombre es definido por el juego.
<Value>	Este es el valor en su representación como cadena de texto.

Synchronization Remote Message

Este es un mensaje enviado desde cualquiera de los Esclavos hacia el Maestro. El propósito de este mensaje pedir al Maestro que comience un proceso de sincronización dado que el Esclavo que está pidiendo el inicio del mismo ha tenido cambios que podrían afectar el ordenamiento en el vector del dueño del balón. También puede hacerse la petición desde la implementación del juego para obtener los valores de las variables compartidas del juego.

```

<xs:element name="SyncRemoteMessage" type="gap2p:SyncRemoteMessage"/>
<xs:complexType name="SessionLeaveMessage">
  <xs:sequence>
  </xs:sequence>
</xs:complexType>

```

3.3.Implementación en J2ME con Bluetooth

A continuación se presentara algunos de los detalles de la implementación del framework sobre la plataforma J2ME utilizando Bluetooth como tecnología de comunicaciones. Primero se analizan los requerimientos y la condición de su implementación. Luego se muestran estadísticas del código fuente y por último algunas de las partes más importantes del mismo.

3.3.1. Requerimientos Funcionales Cubiertos

Código:	Impl:	Detalle:
RF1	Si	Lo hace a través de sockets bluetooth.
RF2	Si	Cada peer tiene una instancia de la clase BTServer que a través de un ConnectionNotifier espera por conexiones entrantes.
RF3	Si	todo lo relacionado con bluetooth es tratado exclusivamente en el paquete net.gap2p.connection.btimpl.
RF4	Si	todo lo relacionado con bluetooth es tratado exclusivamente en el paquete net.gap2p.connection.btimpl.
RF5	Si	A traves de la clase ConnectionService
RF6	Si	Se hace a través de la clase BTConnectionHandlerListener propia de la implementación del framework.
RF7	Si	Se hace por medio de la clase ConnectionsHashtable del paquete net.gap2p.util.
RF8	Si	Se hace por medio de la clase GroupService del paquete net.gap2p.group.
RF9	Si	Se hace por medio de la clase GroupService del paquete net.gap2p.group.
RF10	Parcial	Únicamente el peer con el rol de maestro puede aceptar nuevos miembros en la sesión. Se dice parcial por que este requerimiento solo se implementa para el peer maestro. Los peers esclavos ignoran peticiones de este tipo.
RF11	Parcial	Únicamente el peer con el rol de maestro puede rechazar otros peers de entrar a la sesión. Se dice parcial por que este requerimiento solo se implementa para el peer maestro. Los peers esclavos ignoran peticiones de este tipo.
RF12	Si	Para registrar un servicio implementado por el usuario es necesario modificar la clase MessageHelper para incluir la rutina de deserializacion de los mensajes propios del

		nuevo servicio.
RF13	Si	Si es un servicio implementado por el usuario el framework primero intentará deserializar el mensaje utilizando la clase MessageHelper.
RF14	Si	Se implementa por medio de la interface P2PNetworkListener del paquete net.gap2p.p2pnetwork, que debe ser implementada por alguna clase de la aplicacion.
RF15	Si	Esto se hace por medio del campo srcPeerUUID del mensaje ConnectionMessage.
RF16	Si	El envío a todos los peers se hace por medio del maestro. Cuando un servicio se registra con el ConnectionService se debe explicitar si los mensajes de este servicio serán repartidos a los demás peers de la sesión o no.
RF17	Si	El mensaje llega a todos los peers, es responsabilidad de servicio descartar un mensaje si no va dirigido hacia el peer local.
RF18	Si	Esto se realiza por medio del DiscoveryService.
RF19	Si	Sin embargo solo permite iniciar conexión con aquellas sesiones cuyo maestro esta al alcance del peer local.
RF20	Si	Esto se realiza por medio del DiscoveryService.
RF21	Si	Esto se realiza por medio del DiscoveryService.
RF22	Si	Esto se realiza por medio de la interface PeerMonitoringListener de la clase DiscoveryService.
RF23	Si	Esto se implementa como un HashTable que va encapsulado dentro del mensaje SynchronizationMessage. El maestro entrega dicho HashTable al framework como respuesta a la notificacion syncStarted() del P2PNetworkListener.
RF24	Si	Esto se implementa por medio del mensaje SynchronizationMessage y el servicio GroupService.
RF25	Si	Esto se implementa por medio del servicio GroupService.

3.3.2. Estadísticas del Código

Las siguientes son estadísticas del tipo NCSS. Esta estadística determina la complejidad de los métodos, las clases/interfaces y los paquetes contando del número de líneas de código de comando no comentadas en cada archivo (NCSS: Non Commenting Source Statements²²).

Esta estadística busca informar al desarrollador sobre aquellas clases y paquetes con una métrica alta de NCSS, lo que significa que aquel componente tiene

²² En este trabajo utilizamos el NCSS tal y como se define en <http://kcllee.com/clemens/java/javancss/>.

demasiadas responsabilidades y/o funcionalidades por lo que es necesario considerarlo para reingeniería. El tope superior de esta métrica es definido dentro de los estándares de codificación de cada grupo de desarrollo y no existe un estándar para ello.

Cual es el objetivo dentro del proyecto de esto?

Para nuestro framework **podemos** utilizar esta estadística para obtener información en cuanto a lo siguiente:

- **Distribución de Cargas:** Los 4 servicios tienen métricas relativamente cercanas con media en 292 NCSS. Esto muestra que cada servicio tiene una complejidad cercana a la de los demás, lo que significa una repartición equitativa de la implementación de los requisitos. Esto se puede interpretar como consecuencia de un buen diseño del framework y brinda buenas condiciones de mantenibilidad del mismo.
- **Funcionalidad Común -vs- Funcionalidad Propia de la implementación:** Una implementación del framework desde **las fuentes producidas** en este trabajo requeriría la reimplementación de los paquetes `net.gap2p.connection.btimpl`, `net.gap2p.protocol.btimpl` y `net.gap2p.util`. Estos tres paquetes tienen una métrica NCSS de 588. El resto del framework (sin contar la librería para la serialización y deserialización de mensajes XML KXML2) suma una métrica NCSS de 1605. Interpretar estos datos revela que el 75% del framework puede ser reutilizado para construir una implementación sobre otra tecnología (100% para construir una aplicación que utilice la actual implementación del framework). Lo que se traduce en que tomaría un cuarto o menos del tiempo utilizado para realizar esta primera implementación.
- **Framework -vs- Aplicación:** La aplicación construida sobre el framework y utilizada para esta prueba estadística es bastante sencilla. Sin embargo de no existir el framework hubiese requerido todo el trabajo de diseño y construcción del mismo. Con la utilización del framework la aplicación como tal ocupa un 7% del código requerido para ofrecer el servicio de Chat. (El Chat si es representativo de los típicos requerimientos de un desarrollador de juegos sobre este framework?, creo que el 7% está sesgado, ya que este framework no se utiliza para desarrollar un Chat? O si?

Nr.	Classes	Functions	NCSS	Javadocs	Package
1	4	36	275	3	<code>net.gap2p.connection</code>
2	4	22	212	0	<code>net.gap2p.connection.btimpl</code>
3	5	35	292	1	<code>net.gap2p.discovery</code>
4	3	38	376	2	<code>net.gap2p.group</code>
5	2	46	227	2	<code>net.gap2p.p2pnetwork</code>
6	4	32	119	9	<code>net.gap2p.platform</code>
7	12	64	316	4	<code>net.gap2p.protocol</code>

8	2	6	19	0	net.gap2p.protocol.btimpl
9	4	19	357	1	net.gap2p.util
10	1	59	834	8	org.kxml2.io
11	3	29	226	0	test

	44	386	3253	30	Total

Packages	Classes	Functions	NCSS	Javadocs	per
11.00	44.00	386.00	3253.00	30.00	Project
	4.00	35.09	295.73	2.73	Package
		8.77	73.93	0.68	Class
			8.43	0.08	Function

Nr.	NCSS	Functions	Classes	Javadocs	Class
1	27	5	0	1	net.gap2p.connection.AddressCache
2	15	2	0	0	net.gap2p.connection.btimpl.BTClient
3	118	10	1	0	net.gap2p.connection.btimpl.BTConnectionHandler
4	7	6	0	0	net.gap2p.connection.btimpl.BTConnectionHandlerListener
5	52	4	0	0	net.gap2p.connection.btimpl.BTServer
6	5	4	0	0	net.gap2p.connection.ConnectionListener
7	226	26	0	2	net.gap2p.connection.ConnectionService
8	2	1	0	0	net.gap2p.connection.PeerAddressChangedListener
9	250	27	1	1	net.gap2p.discovery.DiscoveryService
10	17	4	0	0	net.gap2p.discovery.DiscoveryServiceProcessStatus
11	3	2	0	0	net.gap2p.discovery.PeerMonitoringListener
12	3	0	0	0	net.gap2p.discovery.PeerMonitoringResult
13	3	2	0	0	net.gap2p.discovery.SessionDiscoveryListener
14	348	30	5	2	net.gap2p.group.GroupService
15	9	8	0	0	net.gap2p.group.GroupServiceListener
16	8	0	0	0	net.gap2p.group.JoinResult
17	12	11	0	0	net.gap2p.p2pnetwork.P2PNetworkListener
18	203	35	1	2	net.gap2p.p2pnetwork.P2PNetworkService
19	49	16	0	2	net.gap2p.platform.Peer
20	17	4	0	5	net.gap2p.platform.Service
21	19	7	0	2	net.gap2p.platform.Session
22	28	5	0	0	net.gap2p.platform.UUID
23	14	5	0	0	net.gap2p.protocol.btimpl.XmlElement
24	2	1	0	0	net.gap2p.protocol.btimpl.XmlMessage
25	45	13	0	0	net.gap2p.protocol.ConnectionMessage
26	45	13	0	0	net.gap2p.protocol.ConnectionResponse
27	11	3	0	1	net.gap2p.protocol.P2PMessage
28	23	7	0	0	net.gap2p.protocol.SessionDiscoveryRequest
29	40	7	0	0	net.gap2p.protocol.SessionDiscoveryResponse
30	21	5	0	0	net.gap2p.protocol.SessionJoinRequest
31	11	3	0	0	net.gap2p.protocol.SessionJoinResponse
32	5	1	0	1	net.gap2p.protocol.SessionLeaveMessage
33	48	7	0	0	net.gap2p.protocol.SynchronizationMessage
34	5	1	0	1	net.gap2p.protocol.SyncInitRequest
35	11	3	0	0	net.gap2p.protocol.SyncInitResponse
36	5	1	0	1	net.gap2p.protocol.SyncRemoteMessage
37	21	2	0	0	net.gap2p.util.ArrayHelper
38	17	5	0	1	net.gap2p.util.ConnectionsHashtable
39	272	8	0	0	net.gap2p.util.MessageHelper
40	28	4	0	0	net.gap2p.util.UUIDGenerator
41	830	59	0	8	org.kxml2.io.KXmlParser
42	33	3	0	0	test.ChatRoomMessageList
43	108	9	0	0	test.ChatRoomsDirectoryList
44	66	17	0	0	test.TestMIDlet
Average Object NCSS:					70.05
Average Object Functions:					8.77
Average Object Inner Classes:					0.18
Average Object Javadoc Comments:					0.68
Program NCSS:					3,253.00

3.3.3. Ejemplos de Código

Esta es la guía del programador del framework?

A continuación **mostraremos** algunas piezas de código de forma ilustrativa de lugares que son importantes **dentro del framework** y la aplicación que lo utiliza.

- **ConnectionService.sendMessage():** Este método es utilizado por los demás servicios (cuales servicios?) para enviar mensajes a un peer por su dirección específica en vez de su identificador. La resolución de la dirección con base en el identificador único UUID se realiza en un paso previo o no se realiza como es el caso de los mensajes de descubrimiento. (como es esto último?)

```

95     private synchronized UUID sendMessage(String address,
ConnectionMessage cm, boolean openConnectionIfNecessary)
96     {
97         BTConnectionHandler handler =
            getConnection(address, openConnectionIfNecessary);
98         if(handler == null)
99             return null;
100
101         //3- Prepares the message to be sent
102         cm.setSrcPeerUUID(getLocalPeer().getPeerUUID());
103         cm.setMessageUUID(UUIDGenerator.generateRandomUUID(
            UUID.MESSAGESUFFIX));
104         if(cm.getDestPeerUUID() == null)
105             cm.setDestPeerUUID(UUIDGenerator.Empty);
106         cm.setRendezvous(ConnectionService.RENDEZVOUS);
107
108         //2- Sends the message using the just grabbed handler
109         String xmlMsg = MessageHelper.getMessageAsXml(cm);
110         System.out.println("Message To Send: " + xmlMsg); esto no creo
que deba aparecer, supongo que es debuggin de uds.
111         handler.queueMessageForSending(
            cm.getMessageUUID(), xmlMsg.getBytes());
112
113         //3- Keeps a pointer to the message until it's sent!
114         messagesBeingSent.addElement(cm);
115
116         return cm.getMessageUUID();
117     }

```

- **ConnectionService.handleReceivedMessage():** Este método es el que recibe y procesa los mensajes recibidos en el (o por el) ConnectionService desde otros peers en la red. Este método es invocado por la implementación y no esta disponible a los demás servicios del framework. Aquí los mensajes son deserializados y entregados al servicio indicado. Si es el caso se hace reenvió del mensaje a los demás peers de la sesión.

```

319 //Here the ConnectionMessages as well as the
      ConnectionResponses are received.
320 public byte[] handleReceivedMessage (BTConnectionHandler btch,
byte[] msgBytes)
321 {
322     System.out.println("Message Received: " + new String(msgBytes));
323     ConnectionMessage cm =
      MessageHelper.getConnectionMessage(msgBytes);
324     if(cm != null)
325     {
326         UUID serviceUUID = cm.getServiceUUID();
327         ConnectionListener cl =
      getRegisteredConnectionListener(serviceUUID);
328         if(cl == null)
329             return null;
330
331         //Handes the message for retransmission
332         forwardMessage(cm);
333         //Checks for a change of the physical address
334         phisicalAddressCheck(btch, cm.getSrcPeerUUID());
335
336         ConnectionResponse cr = null;
337         try
338         {
339             cr = cl.processMessage(cm);
340         }
341         catch(Exception e)
342         {
343             System.out.println("ERROR: The processMessage
      method threw an exception.");
cuando hay excepciones reales en el celular, en donde muestra estos
mensajes?
344             e.printStackTrace();
345             btch.stop();
346         }
347
348         if(cr == null)
349             return null;
350
351         cr.setMessageUUID(cm.getMessageUUID());
352         cr.setOrgSrcPeerUUID(cm.getSrcPeerUUID());
353         cr.setServiceUUID(serviceUUID);
354         cr.setSrcPeerUUID(getLocalPeer().getPeerUUID());
355         cr.setRendezvous(ConnectionService.RENDEZVOUS);
356         String xmlMsg = MessageHelper.getMessageAsXml(cr);
357         System.out.println("Response To Send: " + xmlMsg);
358         return xmlMsg.getBytes();
359     }
360
361     ConnectionResponse cr =
      MessageHelper.getConnectionResponse(msgBytes);
362     if(cr != null)
363     {
364         UUID serviceUUID = cr.getServiceUUID();
365         ConnectionListener cl =
      getRegisteredConnectionListener(serviceUUID);
366

```

```

367     //Checks for a change of the phisical address
368     phisicalAddressCheck(btch, cr.getSrcPeerUUID());
369
370     if(cl != null)
371         cl.processResponse(cr);
372     return null;
373 }
374 return null;
375 }

```

- **DiscoveryService.processMessage():** Este método es invocado por el ConnectionService al hacer la entrega de un mensaje que tiene como identificador de servicio el DiscoveryService. Este servicio solo recibe mensajes de tipo SessionDiscoveryRequest a los cuales responde automáticamente enviando la información relacionada a la sesión a la que pertenece el peer local.

```

195 public ConnectionResponse processMessage(ConnectionMessage cm)
196 {
197     // It only receives ServiceDiscoveryRequest
198     Object content = cm.getContent();
199     if (content instanceof SessionDiscoveryRequest)
200     {
201         ConnectionResponse cr = new ConnectionResponse();
202         // The peer's session has not been created
203         // neither the peer has joined
204         // another session.
205         if(group.getCurrentSession() == null)
206             return null;
207
208         SessionDiscoveryRequest sdr =
209             (SessionDiscoveryRequest) content;
210         if(sdr.getGameUUID().equals(P2PNetworkService.getGameUUID()) &&
211            sdr.getVersion().equals(P2PNetworkService.getVersion()) &&
212            sdr.getGameName().equals(P2PNetworkService.getGameName()))
213         {
214             SessionDiscoveryResponse sdresp =
215                 new SessionDiscoveryResponse();
216             sdresp.setGameUUID(P2PNetworkService.getGameUUID());
217             sdresp.setVersion(P2PNetworkService.getVersion());
218             sdresp.setSession(group.getCurrentSession());
219
220             cr.setContent(sdresp);
221             return cr;
222         }
223     }
224     return null;
225 }

```

- **GroupService.syncSlaves():** Este método es invocado por el peer maestro cuando desea iniciar el proceso de sincronización con los demás peers

esclavos que pertenecen a la sesión.

```

49 public void syncSlaves()
50 {
51     // If the local peer is recovering, at the end of the recovery
process
52     // will make a call for synchronizing slaves again.
53     if(currentSession == null ||
54         currentSession.getPeers() == null ||
55         currentSession.getPeers().size() == 0 ||
56         !localPeer.isMaster() ||
57         mode == MODERECOVERY)
58     {
59         return;
60     }
61
62     varsToSynchronize = listener.syncStarted();
63
64     removeNonRechablePeersFromSession();
65
66     SyncInitRequest sireq = new SyncInitRequest();
67     ConnectionMessage cm = new ConnectionMessage();
68     cm.setContent(sireq);
69     cm.setServiceUUID(getServiceUUID());
70
71     Enumeration peers = currentSession.getPeers().elements();
72     while(peers.hasMoreElements())
73     {
74         Peer tmpPeer = (Peer)peers.nextElement();
75         connection.sendMessage(tmpPeer, cm);
76     }
77
78     timerSyncRequest = new Timer();
79     timerSyncRequest.schedule(new TimerTask() {
80         public void run() {continueSyncSlaves();}
81     }, Service.WAITINGTIMEOUTSHORT);
82 }

```

- **GroupService.continueSynSlaves():** Este método es la continuación del proceso de sincronización comenzado por el maestro y se ejecuta una vez se reciban todas las respuestas desde los esclavos a la petición de inicio del proceso de sincronización.

```

84 /**
85  * This method is called when all the SyncInitResponses
      are received or when
86  * it times out for the responses to come.
87  */
88 private void continueSyncSlaves()
89 {
90     //TODO: What happens when the syncinitresponses are
      not completed?
91     //1- recalculates the order of the vector
92     calculateVDB();
93 }

```

```

94 //2- sends the synchronization message to all the peers
95 SynchronizationMessage sm = new SynchronizationMessage();
96 sm.setSession(currentSession);
97 sm.setResumeDelay(1000);
98 sm.setGameVariables(varsToSynchronize);
99
100 ConnectionMessage cm = new ConnectionMessage();
101 cm.setContent(sm);
102 cm.setServiceUUID(getServiceUUID());
103
104 Enumeration peers = currentSession.getPeers().elements();
105 while(peers.hasMoreElements())
106 {
107     Peer tmpPeer = (Peer)peers.nextElement();
108     connection.sendMessage(tmpPeer, cm);
109 }
110
111 try {
112     Thread.sleep(sm.getResumeDelay());
113 } catch(Exception e) {}
114
115 listener.syncFinished(null);
116 }

```

- **P2PNetworkService.start():** Este es la rutina interna de inicialización del framework. Aquí el P2PNetworkService obtiene las referencias únicas a los demás servicios y establece las relaciones entre los mismos por medio de los Listeners. Por último inicializa los demás servicios y retorna.

```

169 /**
170  * This method is the starting point for the entire framework.
171     When this
172     * method is called all the services get initialized, the relations
173     * between them are stablished (in terms of listeners) and
174     * finally they are
175     * started.
176  */
177 public void start()
178 {
179     if(isStarted())
180         return;
181
182     aborting = false;
183     Writer writer = new Writer();
184     Thread writeThread = new Thread(writer);
185     writeThread.start();
186
187     isRecovering = false;
188     isSynchronizing = false;
189     messagesToSend.removeAllElements();
190
191     //1-Creates the services
192     connection = ConnectionService.getInstance();
193     connection.start();

```

```

192
193     group = GroupService.getInstance();
194     discovery = DiscoveryService.getInstance();
195
196     //2-Stablishes relations between the services
197     connection.setPeerAddressChangedListener(group);
198     discovery.setSessionDiscoveryListener(this);
199     discovery.setPeerMonitoringListener(group);
200     group.setListener(this);
201
202     //3-Registers the services with the ConnectionService
203     connection.registerConnectionListener(
204         discovery.getServiceUUID(), discovery, false);
205     connection.registerConnectionListener(
206         group.getServiceUUID(), group, false);
207     connection.registerConnectionListener(
208         getServiceUUID(), this, true);
209
210     //4-Starts all the services
211     group.start();
212     discovery.start();
213
214     //5-Sets the service as started
215     super.start();
216 }

```

Esto es de la guía del programador?

- **TestMIDlet:** En estas líneas de código muestran la forma en que una aplicación cliente, en este caso un MIDlet, declara e inicializa el framework para ser utilizado.

Que hace? O debe hacer?

```

28     public TestMIDlet()
29     {
30         pingString = String.valueOf(System.currentTimeMillis());
31         pingString = pingString.substring(
32             pingString.length() - 2, pingString.length());
33         System.out.println("Ping String: " + pingString);
34
35         p2pservice = P2PNetworkService.getInstance(
36             "testgame", "1.0", "pikaPeer-" + pingString);
37         p2pservice.setP2PNetworkListener(this);
38     }
39
40     protected void startApp() throws MIDletStateChangeException
41     {
42         p2pservice.start();
43     }

```

Veo la guía de implementación del framework o guía del programador muy pobre.

3.3.4. Entorno de Desarrollo

Que requiere o necesita el entorno de desarrollo en la Guía del Programador? El mismo?

Para el desarrollo del framework se utilizó el entorno de desarrollo **Eclipse** en su versión 3.1[1], este es un entorno integrado de desarrollo (IDE) multiplataforma libre para crear aplicaciones clientes de cualquier tipo. La primera y más importante aplicación que ha sido realizada con este entorno es el **afamado** IDE Java llamado *Java Development Toolkit* (JDT) y el compilador incluido en Eclipse, que usaron para desarrollar el propio Eclipse.

También se utilizó la plataforma Carbide.j 1.5 de Nokia la cual permite utilizar eclipse como entorno de desarrollo para aplicaciones J2ME. También permite simular escenarios de uno o más dispositivos en una red bluetooth por medio de la herramienta llamada Nokia Connectivity Framework.

Las herramientas **de** utilizaron durante las diferentes etapas del desarrollo de la siguiente forma:

- **Desarrollo:** eclipse-SDK-3.1-win32 & carbide.j-1.5, con la librería `xmllpull_1_1_3_1.jar` como una referencia externa al proyecto y con el archivo fuente de `KXmlParser` dentro de la estructura de archivos de clase.
- **Empaquetamiento:** WTK2.5 Beta configurado el proyecto con acceso a bluetooth (JSR 82), MIDP 2 y CLDC 1.1, con la librería `kxml2-min-2.3.0.jar` en la carpeta `lib/` del proyecto y sin el archivo fuente del `KXmlParser` dentro de la estructura de archivos de clase.
- **Ejecución en entorno de pruebas:** NCF (Nokia Connectivity Framework), con el `.jar` empaquetado con el WTK2.5 y con el simulador Prototype 4.0 S60 MIDP Emulator.

5. GUIA PARA EL DESARROLLADOR

Le falta como guía del programador. Como ejemplo observa el de la guía de programación con JMF.

Se debe mostrar diagramas de secuencia y/o estados para utilizar el framework.

Para un programador, que hacen los objetos del framework?

Hasta el momento **sólo ha mencionado de la especificación formal** del framework, en el siguiente apartado se explican los pasos a seguir para implementar juegos que hagan uso del mismo. Es **claro** que el único punto de interacción del programador con el framework es el servicio (o objeto) P2PNetworkService el cual tiene como función principal servir de punto de contacto único del framework con la implementación del juego.

Lo primero que se debe hacer para usar los servicios del framework es implementar la interfase P2PNetworkListener, al implementar esta interfase es necesario implementar los siguientes métodos:

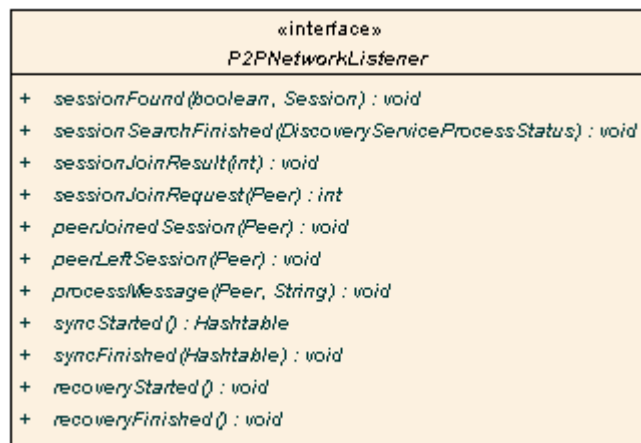


Imagen 41: P2PNetworkListener, interfase que debe ser implementada para usar el framework.

Método	Detalle
sessionFound(boolean canConnect, Session session);	Método invocado cuando se encuentra una sesión de juego.
SessionSearchFinished(DiscoveryServiceProcessStatus st);	Método invocado cuando termina la búsqueda de las sesiones del juego cercanas.

<code>sessionJoinRequest(Peer peer);</code>	Método invocado a la hora que se recibe una petición de ingreso a mi sesión.
<code>sessionJoinResult(int joinResult);</code>	Método invocado por el resultado que se genera al unirse a una sesión.
<code>processMessage(Peer peer, String msg);</code>	Método invocado cuando se reciben mensajes de un peer en específico.
<code>peerJoinedSession(Peer peer);</code>	Método invocado en el momento que un peer se une a una sesión.
<code>peerLeftSession(Peer peer);</code>	Método invocado en el momento que un peer deja la sesión.
<code>Hashtable syncStarted();</code>	Método que envía los valores del a sincronizar del juego o la aplicación.
<code>syncFinished(Hashtable ht);</code>	Método que recibe los valores a sincronizar.
<code>recoveryStarted();</code>	Método que se implementa cuando empieza a recuperar una sesión.
<code>recoveryFinished();</code>	Método que se implementa cuando termina de recuperar una sesión.

Luego de la implementar la interfaz lo que sigue es obtener una instancia del P2PNetworkService y **asociarla al servicio el listener de la siguiente forma**. Esta instancia funciona como un Singleton:

```
34     p2pservice = P2PNetworkService.getInstance(
        GAME_NAME, GAME_VERSION, LOCAL_PEER_NAME);
35     p2pservice.setP2PNetworkListener(this);
```

Por último para dejar el servicio activo es necesario iniciarlo mediante siguiente método:

```
40     p2pservice.start();
```

Mientras que el servicio está activo, éste adopta diferentes estados. Este puede actuar de dos formas, como dueño del juego o la aplicación (maestro) o como invitado (esclavo), cada peer invitado puede dejar la sesión y **pasar ser dueño** de esta, en el momento que este no encuentre un peer cercano con mayor prioridad²³.

Un peer puede también crear una nueva sesión de juego. En esta nueva sesión el

²³Ver el algoritmo en el numeral 3.2.3.3 acerca de continuidad de la sesión.

peer que la crea es el primer dueño o maestro. Para hacer la creación es necesario llamar el método `p2pservice.createNewSession()` el cual comenzará a responder peticiones de descubrimiento con información de la nueva sesión de juego.

Por otra parte un peer también puede entrar a una sesión existente, para esto primero debe buscar las sesiones que estén a su alcance y luego escoger a la que se quiere conectar para así establecer el enlace con el peer dueño de la sesión. El peer solo se podrá conectar con aquellas sesiones a las que puede contactar directamente al peer maestro. Para este se utilizan los siguientes métodos:

Método	Detalle
<code>p2pservice.searchSessions()</code>	Para la búsqueda de sesiones de juego que estén el rango del dispositivo. El rango es definido por la tecnología de transporte utilizada en la implementación. En el caso de la implementación J2ME/BT el rango es de 10 metros. Esto hay que aclararlo, en el sentido que BT si es de alcance físico, pero cuando se implemente otros transporte que? Por ejemplo Wifi o http?
<code>p2pservice.joinSession(session)</code>	Para unirse a una sesión de juego cuyo peer maestro esta dentro del rango.

Después que dos peers estén conectados entre sí éstos pueden intercambiar mensajes con el método `p2pservice.sendMessage(msg)`. Estos mensajes van al peer maestro y éste los reenvía a los demás peers de la sesión.

Por último es importante ver otras funcionalidades del servicio las cuales son: `p2pservice.cancelSessionSearch()` la cual **nos** permite cancelar la búsqueda de sesiones de juego y `p2pservice.leaveSession()` que **nos** permite dejar una sesión ya sea voluntariamente o abruptamente.

5.1. Validación del Framework Mediante Un Ejemplo

A continuación se presenta un ejemplo de una aplicación desarrollada por el grupo de trabajo para la validación del framework (cual grupo?). Presentar esta aplicación busca que la misma sea utilizada como punto de referencia para otros desarrollos sobre el framework.

5.1.1. Mi primer MIDlet con el GA P2P Network Framework

Para mostrar el funcionamiento del framework se implementará un Chat (porque un Chat?), desde éste **podremos** ver como se crean sesiones, descubren sesiones, los peers ingresan a sesiones y se envían mensajes, entre diferentes peers en un misma sesión.

Nuestro primer paso para la implementación es crear el midlet y las demás clases donde **vamos** a implementar la aplicación, la primera clase que **tendremos** es TestMIDlet, el cual es nuestro midlet, la clase ChatRoomsDirectoryList, que será la encargada de crear sesiones, descubrir sesiones y unirse a ellas, entre otras funcionalidades y la clase ChatRoomMessageList que básicamente es desde la cual **enviamos** los mensajes a los diferentes peers. Al crear el midlet, este debe implementar la interfaz P2PNetworkListener que funciona como un listener??? (no sería mejor decir que como un manejador de eventos?) de la siguiente forma:

```
15 public class TestMIDlet
    extends MIDlet implements P2PNetworkListener
```

Luego de esto en el constructor se obtiene una instancia del servicio y asociamos el listener al servicio.

```
28 public TestMIDlet()
29 {
30     pingString = String.valueOf(System.currentTimeMillis());
31     pingString = pingString.substring(
        pingString.length() - 2, pingString.length());
32     System.out.println("Ping String: " + pingString);
33
34     p2pservice = P2PNetworkService.getInstance(
        GAMENAME, GAMEVERSION, LOCALPEERNAME);
35     p2pservice.setP2PNetworkListener(this);
36 }
```

Antes de iniciar el servicio **vamos** a implementar todos los métodos de la interfaz.

```
108 public void processMessage(Peer peer, String msg)
109 {
110     messageList.newMessage(peer.getNick(), msg);
111 }
112
113 public void sessionFound(boolean canConnect, Session session)
114 {
115     directoryList.sessionFound(canConnect, session);
116 }
117
118 public void sessionSearchFinished(
    DiscoveryServiceProcessStatus status)
119 {
120     directoryList.searchSessionsFinished(status);
```

```

121 }
122
123 public int sessionJoinRequest(Peer peer)
124 {
125     return JoinResult.SESSIONJOINED;
126 }
127
128 public void sessionJoinResult(int joinResult)
129 {
130     if(joinResult != JoinResult.SESSIONJOINED)
131     {
132         changeTo(ROOMSDIRECTORY);
133         directoryList.showErrorSessionJoin(joinResult);
134     }
135 }
136
137 public void syncFinished(Hashtable ht)
138 {
139 }
140
141 public Hashtable syncStarted()
142 {
143     Hashtable tableGameVariables = new Hashtable();
144     tableGameVariables.put("String Variable", "variable value 1");
145     tableGameVariables.put("Integer Variable", String.valueOf(777));
146     return tableGameVariables;
147 }

```

Luego de implementar todos los métodos de la interfaz **vamos** a iniciar el servicio desde el constructor o desde donde lo considere necesario.

```

38 protected void startApp() throws MIDletStateChangeException
39 {
40     p2pservice.start();
41     directoryList = new ChatRoomsDirectoryList(
42         "Chat Rooms", List.IMPLICIT, this);
43     messageList = new ChatRoomMessageList(
44         "Room Messages", List.IMPLICIT, this, pingString);
45     changeTo(ROOMSDIRECTORY);
46 }

```

Por último **nuestro** midlet debe parar el servicio al terminar su ciclo de vida.

```

102 protected void destroyApp(boolean p1)
103         throws MIDletStateChangeException
104 {
105     p2pservice.cancelSessionSearch();
106     p2pservice.stop();
107 }

```

Porque invocar de manera determinística el método `cancelSessionSearch`? Esto no debería ser condicionante:

```
If (searchingsessions == true)
    p2pservice.cancelSessionSearch();
```

5.1.2. Utilización de los Servicios

Ahora se implementarán los servicios de creación, descubrimiento y **aclope** de sesiones desde la clase ChatRoomsDirectoryList.

Un peer puede crear la sesión o ir en busca de sesiones. Primero **vamos** a ver que debe implementar un peer cuando este sea dueño de la sesión. Aquí **podemos** ver que mediante el comando create sesión el peer está creando la sesión.

```
52     public void commandAction(Command command, Displayable source) {
...
68         else if(command == createCommand)
69             {
70                 deleteAll();
71                 removeAllCommands();
72
73                 midlet.changeTo(TestMIDlet.ROOMMESSAGES);
74
75                 p2pservice.createNewSession();
76             }
...
95     }
```

Por otra parte, si éste se va a unir a una sesión ya creada debe llamar el método p2pservice.searchSessions() e implementar de manera que **vea** conveniente sessionFound() y searchSessionsFinished(DiscoveryServiceProcessStatus status). Y por último para unirse a la sesión de la aplicación hacer el llamado de p2pservice.joinSession(session).

```
52     public void commandAction(Command command, Displayable source) {
...
57         else if(command == joinCommand || command == List.SELECT_COMMAND)
58             {
59                 Session session =
                    (Session) sessionsFound.elementAt(getSelectedIndex());
60
61                 deleteAll();
62                 removeAllCommands();
63
64                 midlet.changeTo(TestMIDlet.ROOMMESSAGES);
65
66                 p2pservice.joinSession(session);
67             }
...
95     }
```

5.1.3. Envió de Mensajes

El envío de mensajes de un peer a otro se hace mediante la clase `ChatRoomMessageList` la cual implementa (si lo implementa? O ya esta implementado en el framework?)el método `p2pService.sendMessage(msg)` para el envío de mensajes a todos los peers que se encuentra en esa sesión.

```

41     public void commandAction(Command command, Displayable source)
42     {
43         if(command == sendMessageCommand)
44         {
45             String msg = "Some Message Text";
46             p2pService.sendMessage(msg);
47             newMessage("local", msg);
48         }
49         else if(command == clearScreenCommand)
50         {
51             deleteAll();
52         }
53         else if(command == leaveRoomCommand)
54         {
55             p2pService.leaveSession();
56             midlet.changeTo(TestMIDlet.ROOMSDIRECTORY);
57         }
58     }

```

5.1.4. Ejecución

Para la ejecución del chat se utilizan las herramientas comunes para compilar y ejecutar midlets lo único necesario para que este compile es la inclusión de la librería de framework.

(no se si vienen más adelante, la ejecución o no real en dispositivos bluetooth)

5.2 Comparativo entre el desarrollo de juegos con este Framework y sin el Framework

Después de implementar un ejemplo simple, ahora **vamos** a ver un comparativo entre el desarrollo utilizando el framework y un desarrollo hecho sin el framework con el API JSR 82 (Bluetooth API) y **mostraremos** las ventajas y desventajas del uso del framework.

El juego que **vamos** a desarrollar para hacer este comparativo es piedra, papel y tijera, es un juego multijugador donde el jugador puede tener un rival remoto o

puede jugar contra la computadora.

En que consiste el juego: piedra, papel y tijera (muchos no sabemos como es y como se juega

Un jugador al iniciar el juego tiene dos opciones:

- Crear un juego: Cuando un jugador crea un juego empieza a jugar contra la computadora, hasta que otro jugador que tenga el mismo juego lo encuentre y lo escoja para jugar con el.
- Unirse a Juego: permite buscar jugadores que hayan iniciado una sesión de juego. Luego de encontrar uno o varios jugadores que tengan esa sesión el jugador escoge a cual juego se va a unir y empieza a jugar.

Cuando dos jugadores ya están unidos en la misma sesión de juego, éstos empiezan a competir escogiendo una de las tres opciones (piedra papel o tijera), después que los dos jugadores han escogido una de las tres opciones el juego les avisa quien a sido el ganador, si lo hay, y aumenta el puntaje de éste en la pantalla.

Luego de saber en que consiste el juego vamos a empezar a desarrollar éste. La implementación en los dos casos (cuales) juego? contiene tres clases que implementan las funcionalidades antes descritas, estas clases son:

- GameSPS: Esta es la clase principal, la que inicia la aplicación. También es la clase que recibe todos los eventos de entrada, salida, búsqueda, inicio de sesión, manejo del estado del juego y además implementa la inteligencia de la computadora. (que es inteligencia de la computadora?)
- GameDirectoryList: Esta clase es la encargada de la creación de sesiones de juego, y la búsqueda de estas mismas.
- GameMessageList: Esta clase es la encargada de generar las acciones inherentes al juego, hacer jugadas, y enviarlas.

Aunque en las dos implementaciones se utilizan los mismo nombres para cada uno de los servicios que debe ofrecer el juego existen diferencias en la implementación.

Como en el ejemplo del numeral ... se puede ver que lo primero que hace el programador es crear las sesiones de juego en el caso del maestro y unirse a ellas en el caso de esclavo.

A continuación se muestra como se hacen estas dos acciones, las cuales están

implementadas en la clase GameDirectoryList:

Si se trabaja con el framework, la creación de sesiones se hace por medio del método `createNewSession()` y la búsqueda de sesiones de juegos por medio del método `searchSessions()`. En este caso el programador no tiene que preocuparse por desarrollar o implementar estas funciones ya que son provistas por el framework:

```

else if(command == createCommand)
{
    deleteAll();
    removeAllCommands();
    midlet.changeTo(GameSPS.GAMEMESSAGES);
    p2pservice.createNewSession();
}
else if(command == scanCommand)
{
    deleteAll();
    removeAllCommands();
    setTicker(scaningTicker);
    addCommand(cancelScanCommand);
    p2pservice.searchSessions();
}

```

Sin Framework, aparte de implementar las anteriores líneas de código hay que implementar, la creación de sesiones y la búsqueda de éstas. En este caso el programador crea la clase BluetoothDiscovery la cual implementa los servicios de búsqueda y creación de sesiones, además asume que el programador tiene conocimiento en programación de Bluetooth sobre Java:

Creación de sesiones

```

public void createNewSession() throws BluetoothStateException,
IOException, InterruptedException
{
    acceptAndOpenThread t;
    // Save Discoverability Mode
    saveDiscoverability();
    // Go in Limited Inquiry scan mode
    localDevice.setDiscoverable(DiscoveryAgent.GIAC);
    // Call connector.open to create Notifier object
    notifier = (StreamConnectionNotifier) Connector.open(
    "btspp://localhost:" + serviceUUID + ";name=" + localName +
    ";authorize=false;authenticate=false;encrypt=false" );

    // Spawn new thread which does acceptandopen
    t = new acceptAndOpenThread();
    // wait on thread (until someone connects)
    synchronized( block_s )
    {
        // Start acceptAndOpen

```

```

    t.start();
    // wait
    block_s.wait();
}
// Clear Notifier (is already closed)
notifier = null;
// restore discoverability mode
    restoreDiscoverability();
// return the connection
}

```

Búsqueda de Dispositivos

```

public void searchSessions()
    throws BluetoothStateException, InterruptedException, IOException
{
    // obtain discovery object which will be used for inquiry
    discoveryAgent = localDevice.getDiscoveryAgent();
    // Create Discovery Listener (Inquiry Listener) Object
    listener = new Listener();
    // Hace el llamado de los metodos deviceDiscovery cuando encuentra
    // un dispositivo y inquiryCompleted cuando termina la busqueda
    discoveryAgent.startInquiry(DiscoveryAgent.GIAC, listener);
}

public void deviceDiscovered( RemoteDevice btDevice, DeviceClass cod )
{
    if( cod.getMajorDeviceClass() != MAJOR_DEVICE_CLASS_PHONE )
    { // return in case it's not a phone
        return;
    }
    // It's another phone, so store it in the list
    if( ! cached_devices.contains( btDevice ) )
    {
        cached_devices.addElement( btDevice );
        midlet.sessionFound(true, btDevice);
    }
}

public void inquiryCompleted( int discType )
{
    listener = null;
    midlet.sessionSearchFinished();
}

```

Después de buscar los dispositivos y encontrar los lo siguiente para un peer esclavo es unirse a una sesión de juego, usando el framework las líneas de código a implementar por el programador son:

```

else if(command == joinCommand || command == List.SELECT_COMMAND)
{
    Session session = (Session)sessionsFound.elementAt(getSelectedIndex());
    deleteAll();
    removeAllCommands();
}

```

```
midlet.changeTo(GameSPS.GAMEMESSAGES);
p2pservice.joinSession(session);
}
```

Sin framework aparte de las líneas anteriores debe implementar, las siguientes líneas de código, en la clase BluetoothDiscovery:

```
public void joinSession(RemoteDevice dev)
{
    listener = new Listener();
    UUID [] u = new UUID[1];
    u[0] = new UUID ( serviceUUID, false);
    try {
        discoveryAgent.searchServices(null, u, dev, listener);
    }catch (BluetoothStateException eb){
        eb.printStackTrace();
    }
}

public void servicesDiscovered( int transID, ServiceRecord[] servRecord )
{
    // A Service was found on the device.
    //currServRec = servRecord[0];
    records = new Vector();
    for (int i = 0; i < servRecord.length; i++) {
        records.addElement(servRecord[i]);
    }
}

public void serviceSearchCompleted( int transID, int respCode )
{
    foundServiceRecords = new Vector();
    urlStrings = new Vector();
    for (int i = 0; i < records.size(); i++) {
        // Reset trans action id
        currServRec = (ServiceRecord)records.elementAt(i);
        serviceSearchTransId = -1;
        if( currServRec != null )
        {
            foundServiceRecords.addElement( currServRec );
            urlStrings.addElement(currServRec.getConnectionURL(ServiceRecord.NO
AUTHENTICATE_NOENCRYPT, false ) );
            try {
                String remoteName =
                    ((RemoteDevice)currServRec.getHostDevice()).getFriendlyName(t
rue);
                btConnection = new BluetoothConnection( (String)
urlStrings.elementAt(i), localName, remoteName);
                btConnection.writeString( localName );
                btConnection.setRemoteName( remoteName );
                midlet.sessionJoinResult(midlet.sessionJoinRequest
(remoteName));
                midlet.peerJoinedSession(remoteName);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    i = records.size();
}
```

```

}
}
}

```

A continuación se muestra como se hacen estas dos acciones, las cuales están implementadas en la clase GameMessageList:

Si se trabaja con el framework, el envío de mensaje hace por medio del método `sendMessage()` del `p2pservice`, en este caso la diferencia no se nota en demasía ya que para el envío del mensaje en el juego sin el framework solo se implementan las siguientes líneas en la clase `BluetoothConnection`.

```

public void sendMessage(String s){
    try {
        writeString(s);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

Para finalizar se mostrara cuales son las ventajas y desventajas cuando usamos o no el framework.

Yo creo que es obvio, que hay ventajas en utilizar el framework, respecto a no utilizarlo, pero aca la pregunta es: ¿Qué desventajas hay entonces en utilizar el framework? Como penaliza o no el rendimiento?

Ventajas de usarlo???

- El tiempo de desarrollo se disminuye ya que hay que implementar menos líneas de código.
- El desarrollador de juegos puede dedicarse a la lógica del negocio sin tener por aspectos de comunicación.
- Agrega funcionalidades al juego las cuales son transparentes para el desarrollador del juego, como sincronización de objetos o variables comunes en el juego, recuperación de la sesión del juego, las cuales abría que implementar para poder tenerlas.

Desventajas de Usarlo

- Aumenta el over-head en la red ya que se transmiten mensajes xml (entonces porque utilizar xml? Que ventajas trae?).
- Aumenta el tamaño del jar y la utilización de la memoria del equipo.

6. DISCUSSION (o conclusiones?)

Esta sección presenta diferentes puntos de discusión en torno al proyecto. Entre ellos está la presentación de los problemas que **encontramos** durante las diferentes fases del desarrollo del proyecto, las conclusiones en diferentes aspectos y por último lo que **proponemos** como trabajo futuro del framework.

6.1. Problemas Encontrados

A continuación **hacemos** una lista de los problemas encontrados en el proceso de diseño, implementación y pruebas del framework:

- La implementación de la maquina virtual en los teléfonos utilizados para pruebas (nokia 6620, sony ericsson P900) presentan **bugs** que causan fallos esporádicos o comportamientos inesperados en la aplicación. También se presentan diferencias considerables en la forma como cada implementación J2ME presenta la interfase de usuario, no obteniéndose muchas veces la experiencia de usuario buscada y resultando en pequeños ajustes de la aplicación para cada teléfono.
- La implementación del API bluetooth tiene comportamientos diferentes en cada fabricante. La diferencia de estos comportamientos va desde los tiempos de descubrimiento y establecimiento de la conexión hasta las diferentes configuraciones de seguridad en cuanto al acceso a la interfase Bluetooth.
- La especificación del API bluetooth deja muchas cosas abiertas, lo que se ve reflejado en las implementaciones del API en los diferentes teléfonos. De esta forma **vemos** que algunas implementaciones no soportan la configuración de "scatternet", el número máximo de conexiones simultaneas varia desde 1 para algunos modelos hasta 7. Siendo 7 el máximo supuesto definido en la especificación. Esto se traduce en que algunos de los dispositivos podrán máximo establecer una conexión.
- Los tiempos de descubrimiento entre dispositivos a través de bluetooth es demasiado alto. Incluyendo el intercambio de mensajes SessionDiscoveryRequest y SessionDiscoveryResponse (pregunto: se hace uso de la opción de cache de BT en el framework para reducir estos tiempos? O es un trabajo futuro?) cada petición de búsqueda de sesiones toma en promedio 25 segundos. Esto hace que los juegos del tipo I3 Automatic Triggered y I4 Automatic no sean posibles de implementar (25 segundos después de la detección de la interacción entre dos peers es

probable que la notificación o tomar acción no sea relevante para los jugadores que están en constante desplazamiento). Que tanto es constante desplazamiento y que limitaciones tienen los 10 metros apenas del BT

- Las herramientas de desarrollo utilizadas para la implementación sobre la plataforma J2ME/Bluetooth (3.3.4 Entorno de Desarrollo) presentan una complejidad agregada dado que es necesario cambiar entre herramientas a medida que se esta desarrollando, probando o empaquetando para instalar en un dispositivo. Esto hace que el proceso de desarrollo sea más lento y más propenso a fallos. Seria ideal tener una herramienta única a cargo de los procesos de desarrollo, pruebas y empaquetamiento. Aquí es relevante aclarar un punto anterior: ese es el entorno de dlo para el desarrollador del framework o para el desarrollador de aplicaciones.

Ahora, la pregunta real aca sería cual es la situación real de BT y que tanto realmente se está empleando hoy en día para juegos multiples? Pienso que esto hay que aclararlo.

6.2. Conclusiones

El proyecto GA P2P Network Framework establece la estructura básica para la construcción de frameworks de comunicaciones y contexto utilizados en la construcción de juegos multiusuario y otras aplicaciones de comunicaciones.

El framework toma un acercamiento del tipo peer-to-peer hibrido, con sistema de recuperación (el vector del dueño del balón) para proveer una infraestructura más robusta en donde no existe punto único de fallo, el despliegue es más económico (dado que no existe la necesidad de instalar y mantener servidores intermedios de descubrimiento, entrega y propagación de mensajes) y hay una continuidad en la sesión de juego a pesar de las salidas de peers debido a retiro voluntario y fallas.

La implementación actual del framework sobre J2ME y Bluetooth presenta algunas limitaciones las cuales se pueden vincular directamente con limitaciones en las implementaciones de la maquina virtual de Java y del API bluetooth. **Esperamos** que las nuevas implementaciones de estas tecnologías sean más estables y más ajustadas a las especificaciones.

Los desarrolladores de aplicaciones pueden utilizar el framework en su implementación de J2ME/Bluetooth para construir aplicaciones. En el caso de la aplicación de Chat desarrollada en este trabajo, fue necesario solo el 7% del código, el 93% restante es provisto por el framework. Y en el caso de piedra, papel o tijera porque no se le hizo este análisis? Esto evidentemente se ve reflejado en disminución de los costos y tiempos de desarrollo de los proyectos.

El desarrollo de aplicaciones con el framework disminuye la complejidad del desarrollo siendo necesario interactuar simplemente con la clase

P2PNetworkService e implementar la interfase P2PNetworkListener. Esto sin tener en cuenta la configuración bluetooth, las conexiones, etc.

El framework tiene un diseño que balancea la responsabilidad de la implementación de los requisitos entre 4 servicios. Estos cuatro servicios son diseñados con diferentes patrones de diseño para asegurar la mantenibilidad y extensibilidad del framework. Diferentes protocolos de comunicaciones fueron definidos e implementados para proveer las funcionalidades de más bajo nivel.

Por último queda la estructura base dispuesta para hacer implementaciones del framework en diferentes plataformas de tal forma que exista interoperabilidad entre las mismas y un juego o aplicación pueda desarrollarse en varias plataformas sin perder su modelo de programación ni su funcionalidad.

6.3. Trabajo Futuro

Encontramos una oportunidad en continuar el desarrollo de este proyecto en los siguientes aspectos:

- **Monitoreo al nivel de la aplicación:** A pesar de que el monitoreo de peers fue definido como una funcionalidad dependiente completamente de la implementación, es necesario incluir paso de mensajes en cierto nivel. Esto es para garantizar que el monitoreo no se reduzca a la búsqueda de la dirección física del host o peer, dado que si por ejemplo el proceso de la aplicación falla y deja de correr los peers de la sesión no se enterarían de este evento. Sin embargo el envío de estos mensajes sería un requerimiento opcional aunque la respuesta a peticiones de este tipo si es mandatorio. Este parrafo esta como enrredado.
- **Organización en Red (Wireless Mesh Networking):** Como es posible organizar los peers de tal forma que no existe siempre un Maestro como el conector central pero que aún así sea posible enviar mensajes a toda la red de manera confiable y tener los mecanismos de sincronización.
- **Estadísticas de Red y Logs del Sistema:** El framework debe tener la capacidad de recoger información que ayude a hacer diagnostico del estado de la red, información como el jitter, la latencia, el alcance de la red con determinados parámetros de desempeño, etcétera. Así como el registro de información sobre la ejecución de la aplicación en términos de logs para obtener la trazabilidad requerida en el proceso de control de fallos.
- **Múltiples Interfaces de Red:** El framework debe tener la capacidad de utilizar diferentes interfaces de comunicaciones al mismo tiempo. Es decir hacer una implementación del framework que soporte varias tecnologías de

transporte simultáneamente (Bluetooth y WLAN) y permita establecer conexiones con los diferentes peer a través de la interfase más conveniente en determinado momento. Por ejemplo si dos peers están conectados por medio de bluetooth, al salir del rango de cobertura de bluetooth, el framework debe ser capaz de determinar otra interfase para continuar la comunicación, por ejemplo WLAN.

- **Mejoramiento de la implementación J2ME/Bluetooth:** La implementación J2ME/Bluetooth es una implementación de referencia la cual puede ser mejorada notablemente en los siguientes aspectos:
 - **Intervalo de sincronización:** Optimizar la frecuencia con la que se hacen las sincronizaciones y vincularlas directamente a eventos que la justifiquen.
 - **Conexiones simultaneas:** Debe tomarse el número de conexiones máximo como un recurso a optimizar (variable: `Bluetooth.connected.devices.max`), así pues, los peers pertenecientes a una sesión pueden ser más que el número máximo de conexiones. Esto sin embargo buscando tener bajo sobre-costos por al establecer/terminar las conexiones.
 - **Tamaño de la librería:** Se debe controlar el tamaño de la librería tanto físicamente como en tiempo de ejecución haciendo un correcto uso de los mecanismos de "Garbage Collection".
 - **Manejo de errores:** Se debe mejorar el manejo de errores incluyéndolo en el diseño general del framework.

Y que hay de otras implementaciones como PDA (Windows Mobile) o PALM, esto también debería ser trabajo futuro.

7. REFERENCIAS

- [1] <http://www.eclipse.org>
- [2] Nokia Corporation, Multi-Player MIDP Game Programming, Versión 1.0; Octubre 29, 2003
- [3] Abdulmotaleb El Saddik, Andre Dufour. "Peer-to-Peer Suitability for Collaborative Multiplayer Games," Multimedia Communications Research Lab (MCRLab) University of Ottawa, Ottawa, Canada, K1N 6N5
- [4] FreeMMG: a hybrid peer-to-peer and client-server model for massively multiplayer games. <http://sourceforge.net/projects/freemmg>
- [5] <http://www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html?page=2>, "What is P2P... and What Isn't [Nov. 24, 2000]"
- [6] David Linner, Fabian Kirsch, Ilya Radusch, Stephan Steglich. "Context-aware Multimedia Provisioning for Pervasive Games," *ism*, pp. 60-68, Seventh IEEE International Symposium on Multimedia (ISM'05), 2005.
- [7] <http://www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html?page=2>, "What is P2P... and What Isn't [Nov. 24, 2000]"
- [8] Fábio Reis Cecin, Rodrigo Real, Rafael de Oliveira Jannone, Cláudio Fernando Resin Geyer. "FreeMMG: A Scalable and Cheat-Resistant Distribution Model for Internet Games". Universidade Federal do Rio Grande do Sul.
- [9] http://www.jxta.org/white_papers.html, Project JXTA White Papers.
- [10] <http://eqplayers.station.sony.com/index.vm>, EverQuest.
- [11] Tsun-Yu Hsiao, Shyan-Ming Yuan. "Practical Middleware for Massively Multiplayer Online Games", Septiembre-Octubre 2005, IEEE Computer Society, National Chiao Tung University
- [12] Organization for Economic Co-operation and Development. Mobile subscribers in total/per 100 inhabitants for OECD, 2004. <http://www.oecd.org/dataoecd/19/40/34082594.xls>.
- [13] Ibrahim, J. 4G Features, Bechtel Telecommunications Technical Journal.
- [14] Organization for Economic Co-operation and Development. Digital Broadband

Content: Mobile Content, New Content For New Platforms, 2005.
<http://www.oecd.org/dataoecd/19/7/34884388.pdf>

- [15] Alf Inge Wang, Michael Sars Norum, Carl-Henrik Wolf Lund, "Issues related to Development of Wireless Peer-to-Peer Games in J2ME," aict-iciw, p. 115, Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT-ICIW'06), 2006.
- [16] Tomar P Moran, Paul Dorish. Introduction to this special issue on Context-Aware computing. Special Issue of Human Computer Interaction, Volume 16, 2001.
- [17] Schmidt, A., Beigl, M., Gellersen, H. There is more to Context than Location. Telecooperation Office (TecO), University of Karlsruhe.
- [18] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. Technical report, Hewlett-Packard Company, HP Laboratories Palo Alto, 2002.
- [19] B. J. Wilson. JXTA, second edition. 2002.
- [20] S. S. Bygdås, Øystein Myhre, S. Nyhus, T. Urnes, and Åsmund Weltzien. Bubbles: Navigating content in mobile ad-hoc networks. Technical report, Telenor FOU, 2003.
- [21] Wolf Lund, C.H., Sars Norum, M. The Peer2Me Framework. Department of Computer and Information Science (IDI). Norwegian University of Science and Technology (NTNU).
- [22] M. Conti. Peer-to-peer research at stanford. Technical report, Computer Science Department, 2003.
- [23] D. Schoder and K. Fischbach. Peer-to-peer, anwendungsbereiche und herausforderungen, in: Schoder detlef; fischbach, kai; teichmann, rene: Peer-to-peer (p2p), okonomische, technologische und juristische perspektiven. Technical report, 2002.
- [24] G. Kortuem. A methodology and software platform for building wearable communities. Technical report, University of Oregon, 2002.
- [25] Gamma, E. Helm, R. Johnson, R. Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional Computing Series.
- [26] Holzmann, G. Design and Validation of Computer Protocols. Prentice Hall

Software Series.

- [27] Forum Nokia. Series 80 Platform: Ad Hoc Communications Over WLAN. Version 1.1, July 6, 2005.
- [28] ZigBee Alliance, ZigBee Specification. December 1, 2006.
- [29] Jeffrey, J. M. (1991). Using petri nets to introduce operating system concepts. In *Papers of the twenty-second SIGCSE technical symposium on Computer science education*, pages 324-329.
- [30] <http://www.media.mit.edu/wearables>
- [31] Weiser, M. (1991) The computer for the 21st century. [*Scientific American*](#), 265, 3, 94-104.
- [32] [Norman, D. A. \(1998\). The invisible computer: Why Good Products Can Fail, the Personal Computer Is So Complex, and Information Appliances Are the Solution. Cambridge, MA: MIT Press.](#)