PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA INFORMÁTICA

DESARROLLO DE UNA ARQUITECTURA PARA JUEGOS EN UN ENTORNO UBICUO

JORGE ANDRÉS INOSTROZA URRUTIA

FRANCISCO JOSÉ REYES CÁCERES

PARA OPTAR AL TÍTULO PROFESIONAL DE

INGENIERO DE EJECUCIÓN EN INFORMÁTICA

INFORME FINAL DEL PROYECTO

JULIO DE 2009

A Dios que ha iluminado siempre mi camino y a mi familia que siempre me ha dado el apoyo incondicional y son el pilar fundamental en mi vida.

Jorge Inostroza Urrutia.

A mi familia, que siempre me apoyó en todo momento, en especial a Dios y mi tata, los amo.

Francisco Reyes Cáceres.

Resumen

La tecnología móvil se encuentra en un mercado creciente y competitivo, y dispositivos móviles, de variadas gamas y tecnologías, como PDAs, smartphones, teléfonos celulares, entre otros son sus representantes, ofreciendo una gran cantidad de servicios y recursos a utilizar.

A medida que crece el desarrollo de estos dispositivos y sus capacidades de procesamiento y almacenamiento también aumenta la producción de aplicaciones de mayor envergadura que exploten estas capacidades. Es aquí en donde el mercado de videojuegos móviles se hace presente. Sin embargo debido a que estos dispositivos se encuentran en una diversidad de gamas, sistemas operativos e implementaciones, se genera una problemática en el desarrollo de videojuegos, debido a la falta de homogeneidad y compatibilidad en la comunicación entre los dispositivos.

El presente trabajo propone un modelo de arquitectura de videojuegos bajo un entorno ubicuo para dispositivos móviles, enfocado principalmente en la comunicación de estos dispositivos mediante el protocolo Bluetooth con el objetivo de homogeneizar las aplicaciones para dispositivos de distintas plataformas y brindar soporte a los desarrolladores de videojuegos.

Palabras claves: tecnología móvil, PDA, smartphones, celulares, videojuego, comunicación, arquitectura, entorno ubicuo, bluetooth.

Abstract

Mobile technology is in an increasingly competitive market, and mobile devices, of various ranges and technologies such as PDAs, smartphones, cell phones, among others are their representatives, offering a wealth of services and resources to use.
As it grows, the development of these devices and their processing and storage capabilities also increases the production of larger applications that exploit these capabilities. This is where the mobile gaming market appears. However due these devices are in a variety of ranges, operating systems and implementations, it creates a problem in game development area because of the lack of uniformity and consistency in the communication between devices.
This article proposes an architecture model game under a ubiquitous environment for mobile devices, primarily focused on the communication of these devices using the Bluetooth protocol with the aim of standardizing applications for different platforms and devices providing support to developers of video games.

Keywords: mobile technology, PDA, smartphones, cell phones, videogame, communication, architecture, ubiquitous environment, bluetooth.

Índice

Índice de ilustraciones

Indice de tablas

Lista de abreviaturas y siglas

AFH: Adaptive Frequency Hopping.

CDC: Connected Device Configuration.

CLDC: Connected, Limited Device Configuration.

FEC*:* Forward Correction Error.

GHZ: Giga Hertz.

HCI*:* Host Controller Interface.

J2EE: Java 2 Enterprise Edition.

J2ME: Java 2 Micro Edition.

J2SE: Java 2 Service Edition.

KVM: Kilobyte Virtual Machine.

L2CAP: Logical Link Control and Adaptation Protocol.

MIDP: Mobile Information Profile.

MOD: Modification.

OBEX: Object Exchange.

P2P: Peer to Peer.

PAN: Personal Area Network.

QoS: Quality of Service.

RFCOMM: Radio Frequency Communication.

SDP: Service Discovery Protocol.

SIG: Special Interest Group (Bluetooth).

SMS: Short Messaging Service.

1. Introducción

El creciente boom en el mercado de dispositivos móviles se está dando a escala mundial sin dejar estrato social desapercibido. Esta explosión por adquirir estos dispositivos ha pasado a convertirse en una necesidad básica para los usuarios más exigentes y ligados a la tecnología de punta. Hoy en día estos dispositivos cuentan con una gran gama de servicios para ofrecer, los cuales, muchas veces están dirigidos a un dominio de usuarios en particular.

Estos dispositivos además de entregar tecnología de último nivel, entregan funcionalidades importantes a sus usuarios, las cuales les brindan independencia, libertad, facilidad de uso y principalmente comodidad para enfrentar el diario vivir.

El mundo tecnológico está evolucionando a un tipo de hardware mucho más portátil, es decir más pequeño, liviano, potente y además poseedor de una gran gamma de mecanismos de comunicación integrados en los dispositivos, dejando las puertas abiertas a desarrolladores de middleware y software para que entren en este mercado ofreciendo soluciones acorde a las necesidades cotidianas de los usuarios finales.

Sin embargo este reto no es nada sencillo, debido a que la plataforma sobre la que se trabaja ya no son computadores de escritorio o portátiles, las condiciones cambian debido a muchas limitantes que poseen los dispositivos móviles en relación a computadores, los cuales cuenta con mayores capacidades de procesamiento y almacenamiento.

Además hay muchas variables en juego, partiendo por la arquitectura con la que cuentan los dispositivos móviles, variados sistemas operativos y componentes de hardware hacen de esta labor algo muy complejo, es decir la heterogeneidad entre dispositivos es mucho más alta, lo que hace mucho más difícil el diseño de una solución estándar.

Ciertamente los usuarios se encuentran rodeados de entornos computarizados embebidos, debido a esto los computadores ya no juegan roles de actores principales, esta situación muchas veces pasa desapercibida, debido al uso cotidiano que los usuarios le dan a estos dispositivos (por ej.: sistemas de navegación GPS, dispositivos móviles: smartphones, PDA's, sistemas inteligentes de seguridad como cámaras IP, entre otros). Esta área de investigación de la informática relativamente nueva recibe el nombre de computación ubicua (actualmente llamada pervasiva) [11].

Las tecnologías de comunicación, son actualmente una parte fundamental de muchos de los dispositivos móviles de hoy en día, estas tecnologías, en gran medida, inalámbricas han ido en un aumento y mejora constantemente, proporcionando características de comunicación con distintos tipos de dispositivos y con el tiempo han ido evolucionando en mejoras en tasas de transferencias de datos, mayores rangos de cobertura, mejoramiento en la seguridad de la transmisión de los datos, entre otras.

En base a lo ultimo mencionado es posible decir que el escenario de trabajo de la computación ubicua es realmente complejo sí se desea entregar una solución integral, por lo que muchas veces se requiere de agentes para lidiar con estos entornos llenos de restricciones.

La industria de videojuegos ha crecido mucho desde los primeros juegos para dispositivos celulares llegando a impulsar los nuevos avances tecnológicos a nivel de hardware.

Los primeros juegos fueron construidos para una gama de jugadores, denominados los casual gamer[1], dejando de lado a los hardcore gamers[2], ya que los dispositivos no eran tan potentes como para poder soportar juegos de alto rendimiento, pero con el avance de los años se ha ido demostrando lo contrario, los dispositivos ya cuentan con grandes prestaciones que han hecho cautivar a muchos hardcore gamers, ya que estos son los que explotan en gran mayoría todos los recursos del dispositivo, uno de estos recursos más explotados es la comunicación entre varios dispositivos en partidas de juego, más conocidas como partidas multiplayer[3]. La construcción de juegos de tipo multiplayer contempla muchos retos (además de los que la computación pervasiva trae por si sola) como lo son coordinación y la prestación servicios QoS[4], entre otros.

La comunicación hoy en día es un factor importante a considerar para obtener éxito con las aplicaciones desarrolladas en cualquier tipo de plataforma, desde una aplicación que corre en un entorno web en una terminal hasta un videojuego que corre en un smartphone, es posible encontrar funcionalidades para poder entablar comunicación con otras entidades. Por esta razón se ha dado especial énfasis y prioridad en el desarrollo del módulo de comunicación.

Tomando en cuenta el escenario en que se encuentran inmersos los videojuegos sobre los dispositivos móviles es importante la construcción o planteamiento de una arquitectura que ayude a estos nuevos escenarios dando soporte en ramas como lo son la comunicación, la coordinación o sincronismo entre los dispositivos, la calidad y el contexto.

Este trabajo de investigación realiza el estudio de herramientas de software para el planteamiento de una arquitectura de videojuegos en un entorno ubicuo y posterior desarrollo de prototipos de videojuegos bajo el protocolo bluetooth como especificación de comunicación dentro de los dispositivos móviles. En el capítulo del estado del arte se estudiarán los inicios de las herramientas para desarrollo de videojuegos, bluetooth y proyectos que se han enfocado en entregar herramientas y soluciones a las comunicaciones y videojuegos sobre los dispositivos**.**

Para plantear esta arquitectura se elaborarán, estudios a frameworks[5] de comunicación y de de videojuegos multiplayer, en caso en que el estudio de compatibilidad arrojara que los frameworks no fuesen compatibles o no sirviesen como solución planteada a esta problemática, se construirían componentes que suplirían este propósito.

[1] Casual Gamer denota a aquellos jugadores de video juegos que no dedican gran parte de su tiempo a esta actividad.
[2] HardcoreGamer 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.
[3] Que participen más de un jugador en la misma sesión del juego.
[4] Calidad por el servicio prestado.
[5] Es un conjunto de clases que cooperan entre si y que juntas hacen un diseño reusable para un tipo especifico de software.

1. Introducción.

A lo largo de este trabajo se mostrarán los modelos estáticos y dinámicos que irán componiendo la realización de esta arquitectura y detallando cada uno de sus módulos y funcionalidades.

2. Objetivos

2.1. Objetivo general.

Diseñar una arquitectura ubicua para videojuegos sobre dispositivos móviles, la cual se compone de un conjunto de librerías que aporten a la programación de videojuegos móviles, enfocada principalmente en la comunicación de los dispositivos a través de la pila de protocolos que ofrece bluetooth.

2.2. Objetivos específicos.

A continuación se listan los principales objetivos que se espera contemple este modelo de arquitectura.

- Familiarizarse con la plataforma y el entorno de de desarrollo J2ME en primera instancia, en pos del desarrollo de los módulos de la arquitectura y prototipos de videojuegos.

- Estudio de la norma y especificación bluetooth sobre los dispositivos móviles y java, esto involucra un estudio a nivel técnico de las capacidades, ventajas y desventajas que esta tecnología brinda, como también el aprendizaje de la pila de protocolos con los que cuenta bluetooth.

- Estudio y concepción de trabajos relacionados con tecnologías móviles, evaluar trabajos ya desarrollados como los frameworks JXME y GA P2P[6] Network Framework.

- Realizar un análisis en base a las necesidades básicas del desarrollo de videojuegos en dispositivos móviles, para así obtener un diseño modular de cada uno de los componentes con los que contará esta arquitectura.

- Implementar y desarrollar el módulo de comunicación entre dispositivos bajo el estándar bluetooth, a través de los protocolos L2CAP y RFCOMM.

- Desarrollar una serie de prototipos desechables que implementen la arquitectura, con el objetivo de testear los módulos desarrollados.

- Realizar pruebas a nivel de test unitarios sobre la comunicación entre dispositivos.

[6] P2P: Peer to Peer, punto a punto, modelo de sistemas distribuidos donde cada individuo de la red se comporta como cliente y servidor a la vez.

3. Estado del Arte

3.1. Computación ubicua.

La Computación ubicua es un área de la informática donde la informática o minicomputadores están en nuestro ambiente cotidiano sin que nosotros lo tengamos en cuenta, ósea, que nuestro entorno este provisto de las tecnologías de la información (muy integrada) en los objetos, cosas, tareas y el entorno, dejando a la computación en si a un segundo plano, ojalá lo más imperceptible posible, con el fin de potenciar cada vez más las tareas cotidianas en todo ámbito y sentido.

Mark Weiser fue el primero en pronunciar esta nueva tendencia de la informática (en 1988), la cual fue reconocida a nivel mundial en 1991 por su trabajo [11]. Su visión era crear entornos de computación y comunicación integrados de tal forma que fuese inapreciable para cualquier persona. Esta idea fue muy temprana para su tiempo, debido que tanto el software como hardware no estaban como en la posición adecuada para poder acompañarlo en la nueva revolución. Paradójicamente sus ideas criticadas en su tiempo son ahora productos comerciales masificados y en constante investigación y revisión, **ver figura 1** (Ej.: dispositivos móviles, sensores avanzados, entre otros).

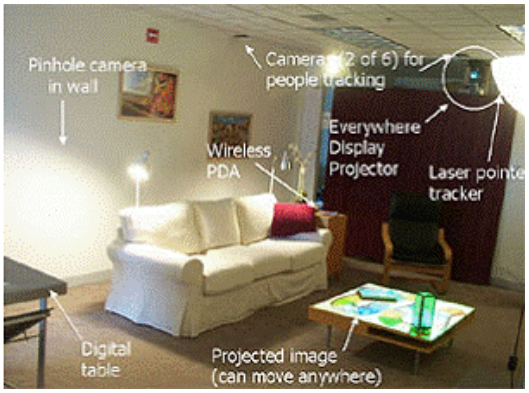

Figura 1 - Habitación ubicua

En [11] Mark Weiser propuso que los computadores fuesen sustituidos por computadores invisibles e integrados en objetos de uso diario, ver cuadro. Para que esta afirmación de la computación ubicua (ideal) sea realidad se deben cumplir ciertas propiedades en estos objetos:

- **Comunicación**: Todos los objetos deben tener capacidad de comunicación tanto con el usuario cómo con los objetos del mismo del entorno.

- **Memoria**: Los objetos deben de contar con una memoria que les sirva para ofrecer un mejor servicio e interacción entre ellos mismos como también para los usuarios.

- **Sensibles al contexto**: deben tener capacidad de adaptarse a las situaciones, lugares, preferencias del usuario o quizás a nuevos objetos en el entorno.

- **Reactivos**: deben de ser capaces de reaccionar al ocurrir determinadas situaciones.

Se observa claramente que la computación ubicua necesita de muchas ramas de la informática para poder brindar o soportar estas características, las ramas más importantes son:

- **Computación sensible**: para poder reaccionar e interactuar con el entorno.

- **Computación móvil**: permite trabajar sin tener que acceder a un computador, en cualquier espacio físico e incluso cuando se está movimiento.

- **Redes de comunicación**: para establecer canales y protocolos de comunicación.

- **Inteligencia Artificial**: para el aprendizaje, adecuación y resoluciones de problemas complejos.

- **Interfaz Hombre Maquina**: con el fin de que la interacción sea siempre una experiencia provechosa, en el contexto de uso dado.

Ya hoy en día la computación ubicua es una área bastante grande de investigación donde los sistemas distribuidos y la computación móvil son los conejillos de india para explotar los entornos ubicuos, pero debido a las exigencias de la computación ubicua, la tecnología de agentes está siendo fuertemente adoptadas con grandes resultados y con grandes productos comerciales ; es más, en Europa el concepto de Agentes se está usando como producto de software en entornos ubicuos y pervasivos (mayormente agentes móviles) [1].

3.2 Bluetooth.

Bluetooth es una tecnología de conectividad inalámbrica de radio de corto alcance que permite la comunicación entre dispositivos remotos. Fue desarrollada para la creación de **PAN** (Personal Área Network), específicamente para redes inalámbricas. Bluetooth utiliza un radio estándar de corto alcance diseñado para implementar la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia seguro y libre (2.4 GHz).

Se diseñó pensando básicamente en tres objetivos: pequeño tamaño, mínimo consumo y bajo precio, y dentro de sus funcionalidades básicas encontramos que:

- Facilita las comunicaciones entre dispositivos móviles y fijos.
- Elimina los cables (tecnología inalámbrica).
- Se pueden crear pequeñas redes inalámbricas.

Este protocolo está enfocado directamente a los dispositivos relacionados con el sector de las telecomunicaciones y la informática personal, donde su uso puede encontrarse en PDAs, teléfonos móviles, Smartphone, pagers, computadoras portátiles, ordenadores personales, impresoras, cámaras digitales, sistemas de navegación, entre otros.

Figura 2 - Bluetooth

3.2.1 Bluetooth SIG

Las especificaciones del protocolo Bluetooth son realizadas por el Bluetooth Special Interest Group (SIG), o Grupo de Interés Especial en Bluetooth, el cual fue fundado en 1998 y que se inició con sólo cinco miembros.

Ahora con más de 9000 compañías integradas se producen a diario nuevos productos que hacen uso de esta tecnología. Algunas de las compañías que se han integrado son Ericsson, Intel, Lenovo, Microsoft, Motorola, Nokia, Toshiba, entre otras. Todas estas empresas se encuentran dedicadas al desarrollo de nuevos dispositivos que implementen esta tecnología.

Las principales tareas del Bluetooth SIG son de publicar las especificaciones de Bluetooth, administrar el programa de calificación, proteger la marca registrada de Bluetooth, y difundir la tecnología inalámbrica de este protocolo.

3.2.2 Versiones de bluetooth

En el año 1994, Ericsson inició un estudio para investigar la viabilidad de una nueva interfaz de bajo costo y consumo para la interconexión vía radio (eliminando así cables) entre dispositivos como teléfonos móviles y otros accesorios. El estudio partía de un largo proyecto que investigaba unos multi-comunicadores conectados a una red celular, hasta que se llegó a un enlace de radio de corto alcance, llamado ***MC link***.

A medida que este proyecto avanzaba se fue haciendo claro que éste tipo de enlace podía ser utilizado ampliamente en un gran número de aplicaciones, ya que tenía como principal virtud que se basaba en un chip de radio.

A partir de esto es como surgió Bluetooth, y sus primeras especificaciones que pronto se convertirían en las primeras versiones del protocolo.

3.2.2.1 Bluetooth 1.0

La versión 1.0 de la especificación de Bluetooth fue liberada en el año 1999, un año después de la formación del Bluetooth SIG.

Ya con un año Bluetooth estaba siendo incorporado en una amplia variedad de dispositivos, luego de tres años más de 500 productos contaban con esta tecnología. La versión 1.2, a diferencia de la 1.1, provee una solución inalámbrica complementaria para coexistir Bluetooth y Wi-Fi en el espectro de los 2.4 GHz, sin interferencia entre ellos.

La versión 1.2 usa la técnica "Adaptive Frequency Hopping (AFH)", que ejecuta una transmisión más eficiente y un cifrado más seguro. Para mejorar las experiencias de los usuarios, la V1.2 ofrece una calidad de voz (Voice Quality – Enhanced Voice Processing) con menor ruido ambiental, y provee una más rápida configuración de la comunicación con los otros dispositivos bluetooth dentro del rango del alcance, como pueden ser PDAs, HIDs (Human Interface Devices), computadoras portátiles, computadoras de escritorio, Headsets, impresoras y celulares.

3.2.2.2 Bluetooth 2.0

En el año 2004 Bluetooth SIG lanzo la especificación de la versión 2.0, donde destaca como característica principal mejoras en las tasas de transferencias de datos.

La versión 2.0, creada para ser una especificación separada, principalmente incorpora la técnica "Enhanced Data Rate" (EDR) que le permite mejorar las velocidades de transmisión en hasta 3Mbps a la vez que intenta solucionar algunos errores de la especificación 1.2.

La versión 2.1, simplifica los pasos para crear la conexión entre dispositivos, además el consumo de potencia es 5 veces menor.

La versión 2.2 aumenta considerablemente la velocidad de transferencia. La idea es que el nuevo Bluetooth trabaje con Wi-Fi, de tal manera que sea posible lograr mayor velocidad en los Smartphones**.**

3.2.2.3 Bluetooth 3.0

El 21 de Abril del 2009 Bluetooth SIG anuncia la llegada de esta nueva especificación, que promete ser mucho más rápida que su antecesora la versión 2.0/2.1, y donde se habla de transferencias de datos nunca antes vistas, como el traspaso de un DVD completo en sólo segundos.

Con una cantidad imponente de compañías respaldando a esta tecnología en crecimiento, Bluetooth se asegura dominar el protocolo de comunicaciones móviles para el pronto futuro.

3.2.3 Especificaciones técnicas

El protocolo de conexión Bluetooth cuenta con una serie de características y especificaciones técnicas que serán listadas a continuación:

- Esta tecnología opera en la banda libre de radio **ISM1** a 2.4 Ghz.
- Su máxima velocidad de transmisión de datos es de 1 hasta 3 Mbps.
- Bluetooth utiliza un esquema de reconocimiento rápido y saltos de frecuencia para garantizar la robustez del enlace.
- El rango de alcance Bluetooth depende de la potencia empleada en la transmisión.
- La mayor parte de los dispositivos que usan Bluetooth transmiten con una potencia nominal de salida de 0 dBm, lo que permite un alcance de unos 10 metros en un ambiente libre de obstáculos.
- Emplea una corrección de error hacia delante (*FEC, Forward Correction Error*) que reduce el efecto del ruido aleatorio en enlaces de larga distancia.

3.2.3.1 Salto de frecuencia

Debido a que la banda ISM está abierta a cualquier dispositivo, el sistema de radio Bluetooth deberá estar preparado para evitar las múltiples interferencias que se pudieran producir. Éstas pueden ser evitadas utilizando un sistema que busque una parte no utilizada del espectro o un sistema de salto de frecuencia.

En este caso la técnica de salto de frecuencia es aplicada a una alta velocidad y una corta longitud de los paquetes (1600 saltos/segundo). Con este sistema se divide la banda de frecuencia en varios canales de salto, donde, los transceptores, durante la conexión van cambiando de uno a otro canal de salto de manera pseudo-aleatoria.

Los paquetes de datos están protegidos por un esquema ARQ (repetición automática de consulta), en el cual los paquetes perdidos son automáticamente retransmitidos.

3.2.3.2 El canal

Bluetooth utiliza un sistema FH/TDD (salto de frecuencia/división de tiempo dúplex), en el que el canal queda dividido en intervalos de 625 ìs, llamados ***slots***, donde cada salto de frecuencia es ocupado por un *slot*.

Dos o más unidades Bluetooth pueden compartir el mismo canal dentro de una *piconet*, donde una unidad actúa como maestra, controlando el tráfico de datos en la *piconet* que se genera entre las demás unidades, donde éstas actúan como esclavas, enviando y recibiendo señales hacia el maestro.

El salto de frecuencia del canal está determinado por la secuencia de la señal, es decir, el orden en que llegan los saltos y por la fase de esta secuencia. En Bluetooth, la secuencia queda fijada por la identidad de la unidad maestra de la *piconet* (un código único para cada equipo), y por su frecuencia de reloj.

3.2.3.3 Datagrama bluetooth

La información que se intercambia entre dos unidades Bluetooth se realiza mediante un conjunto de *slots* que forman un paquete de datos. Cada paquete comienza con un código de acceso de 72 bits, que se deriva de la identidad maestra, seguido de un paquete de datos de cabecera de 54 bits. Éste contiene importante información de control, como tres bits de acceso de dirección, tipo de paquete, bits de control de flujo, bits para la retransmisión automática de la pregunta, y chequeo de errores de campos de cabecera.

La dirección del dispositivo es en forma hexadecimal. Finalmente, el paquete que contiene la información, que puede seguir al de la cabecera, tiene una longitud de 0 a 2745 bits.

En cualquier caso, cada paquete que se intercambia en el canal está precedido por el código de acceso. Los receptores de la *piconet* comparan las señales que reciben con el código de acceso, si éstas no coinciden, el paquete recibido no es considerado como válido en el canal y el resto de su contenido es ignorado.

Figura 3 - Datagrama bluetooth

3.2.3.4 Establecimiento de la conexión

La conexión con un dispositivo, se hace mediante un message *page*. Si la dirección es desconocida, antes del mensaje *page* se necesitara un message *inquiry*.

Antes de que se produzca ninguna conexión, se dice que todos los dispositivos están en modo *standby*. Un dispositivo en modo *standby* se despierta cada 1.28 segundos para escuchar posibles mensajes *page/inquiry*. Cada vez que un dispositivo se despierta, escucha una de las 32 frecuencias de salto definidas. Un mensaje de tipo *page*, será enviado en 32 frecuencias diferentes. Primero el mensaje es enviado en las primeras 16 frecuencias (128 veces), y si no se recibe respuesta, el maestro mandará el mensaje *page* en las 16 frecuencias restantes (128 veces). El tiempo máximo de intento de conexión es de 2.56 segundos.

En el estado conectado, el dispositivo Bluetooth puede encontrarse en varios modos de operación:

- ***Active mode*:** En este modo, el dispositivo Bluetooth participa activamente en el canal.

- ***Sniff mode*:** En este modo, el tiempo de actividad durante el cual el dispositivo esclavo escucha se reduce. Esto significa que el maestro sólo puede iniciar una transmisión en unos *slots* de tiempo determinados.
- ***Hold mode*:** En el estado conectado, el enlace con el esclavo puede ponerse en espera. Durante este modo, el esclavo puede hacer otras cosas, como escanear en busca de otros dispositivos, atender otra *piconet*, etc.
- ***Park mode*:** En este estado, el esclavo no necesita participar en la *piconet*, pero aún quiere seguir sincronizado con el canal. Deja de ser miembro de la *piconet*. Esto es útil por si hay más de siete dispositivos que necesitan participar ocasionalmente en la *piconet*.

3.2.3.5 Creación de redes

Los dispositivos bluetooth son capaces de crear redes ad-hoc entre ellos. Esto vale decir que si un equipo se encuentra dentro del radio de cobertura de otro, éstos pueden establecer conexión entre ellos.

3.2.3.6 Piconet

Cada dispositivo tiene una dirección única de 48 bits, basada en el estándar IEEE 802.11 para WLAN. En principio sólo son necesarias un par de unidades con las mismas características de hardware para establecer un enlace. Dos o más unidades Bluetooth que comparten un mismo canal forman una ***piconet***. Dos o más unidades Bluetooth que comparten un mismo canal forman una *piconet*.

Para regular el tráfico en el canal, uno de los dispositivos participantes se convertirá en **maestro**, pero por definición, la unidad que establece la *piconet* asume éste papel y todos los demás serán **esclavos**. Los participantes podrían intercambiar los papeles si una unidad esclava quisiera asumir el papel de maestra. Sin embargo sólo puede haber un maestro en la *piconet* al mismo tiempo. Hasta ocho usuarios o dispositivos pueden formar una *piconet* y hasta diez *piconets* pueden coexistir en una misma área de cobertura, en otras palabras Bluetooth permite hasta 255 dispositivos en **HoldMode** dentro de la *piconet*.

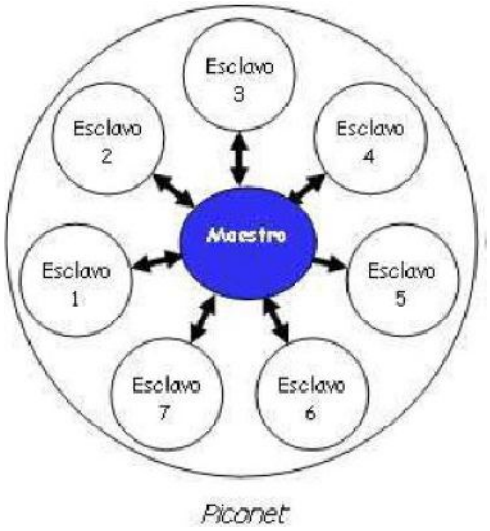

Figura 4 – Piconet.

3.2.3.7 Medios y velocidades

Además de los canales de datos, están habilitados tres canales de voz de 64 kbit/s por *piconet*. Las conexiones son uno a uno con un rango máximo de diez metros, aunque utilizando amplificadores se puede llegar hasta los 100 metros, pero en este caso se introduce alguna distorsión. Los datos se pueden intercambiar a velocidades de hasta 1 mbit/s. El protocolo banda base que utiliza Bluetooth combina las técnicas de circuitos y paquetes para asegurar que los paquetes llegan en orden.

3.2.3.8 Scatternet

Los equipos que comparten un mismo canal sólo pueden utilizar una parte de su capacidad. Aunque los canales tienen un ancho de banda de un 1Mbit, cuantos más usuarios se incorporan a la *piconet*, disminuye la capacidad hasta unos 10 kbit/s más o menos. Para poder solucionar este problema se adoptó una solución de la que nace el concepto de scatternet.

El rendimiento, en conjunto e individualmente de los usuarios de una *scatternet*es mayor que el que tiene cada usuario cuando participa en un mismo canal de 1 Mbit hasta 3 Mbit. Además, estadísticamente se obtienen ganancias por multiplexación y rechazo de canales salto. Debido a que individualmente cada *piconet* tiene un salto de frecuencia diferente, diferentes *piconets* pueden usar simultáneamente diferentes canales de salto.

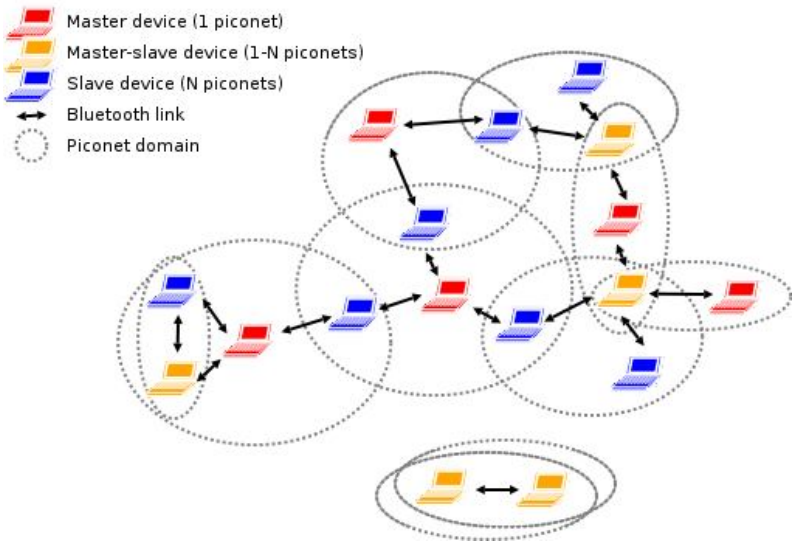

Figura 5 – Scatternet

3.2.3.9 Descubrimiento de servicios

Uno de los principales objetivos que tendrá este trabajo investigativo será la búsqueda y descubrimiento de servicios entre dispositivos móviles situados en una red, las tecnologías a elegir dependerán de una serie de factores que deben ser estudiados con el fin de tener un punto comparativo.

El SIG Bluetooth además del protocolo de comunicación propiamente dicho ha definido una serie de protocolos a nivel de aplicación que facilitan el desarrollo de aplicaciones Bluetooth. Entre ellos se encuentra el **ServiceDiscoveryProtocol** [12]. Este protocolo le permite a un dispositivo Bluetooth conocer cuáles son los servicios que proporcionan los dispositivos con lo que tiene conectividad.

Este protocolo sigue un esquema clásico cliente /servidor en el que la descripción de los servicios se almacena como un **service record** en el dispositivo que actúa como servidor SDP, que, normalmente se corresponde con el que ejerce de maestro en la *piconet* Bluetooth. SDP soporta búsquedas por clases de servicios, y por atributos de servicios, la distinción entre clases y atributos no está bien definida por lo que es necesario que los fabricantes se pongan de acuerdo en estas definiciones para que este protocolo sea interoperable a nivel práctico.

SDP solo permite el descubrimiento de servicios y no entra en cómo se accede a ellos, se considera que se debe definir en los protocolos de más alto nivel, que hacen uso de SDP. Sin embargo, dentro de SDP se define un atributo común a todos los servicios. ProtocolDescriptionList, en el que se define la lista de protocolos con los que se puede acceder al servicio.

3.3 J2ME

Java 2 Platform Micro Edition, el lenguaje de programación que surge como necesidad de permitir que aplicaciones creadas en Java sean soportadas en dispositivos con procesamiento limitado; celulares, palms, pagers son claros ejemplos de éstos. Cabe mencionar que J2ME conserva la compatibilidad con **J2SE**, aunque de cierta más limitada debido a las características que poseen estos dispositivos. Actualmente J2ME abarca dos categorías de dispositivos. Primeramente están aquellos denominados como fijos, que se conectan a la red fija y sus capacidades de almacenamiento varían entre los 2 y los 16 megabytes (set top boxes, Internet TV, sistemas de navegación de automóviles). Por otro lado están los dispositivos móviles, que acompañan al usuario en el quehacer diario, donde sus microprocesadores **RISC/CISC** no supera los 32 bits, y su principal método de comunicación es vía inalámbrica (celulares, palms, pagers). J2ME se caracteriza por poseer una arquitectura modular, adaptada a las limitaciones de los diferentes dispositivos en los q se integra.

3.3.1 La arquitectura J2ME

La arquitectura de J2ME se divide en tres partes, Máquina Virtual, Configuraciones y Perfiles, las cuales se describen a continuación.

3.3.1.1 Máquina virtual

En la actualidad J2ME soporta dos máquinas virtuales: la Java Virtual Machine que se emplea en ediciones J2SE y en J2EE para los dispositivos de procesadores de 32 bit, y la [9] para arquitecturas de 16/32 bits pero con capacidades de almacenamiento limitado.

3.3.1.2 Configuraciones

Definen una serie de bibliotecas Java que están disponibles para un conjunto de dispositivos, con similares capacidades de procesamiento y memoria, J2ME soporta varias configuraciones, en la actualidad existen dos estandarizadas:

- **Connected, Limited Device Configuration (CLDC)** [9], que engloba en general a dispositivos personales móviles.

- **Connected Device Configuration (CDC)** [9], que engloba en general a dispositivos fijos. Por motivos de compatibilidad es un súper conjunto de CLDC.

Ambas configuraciones tiene clases comunes con la J2SE, que permite la compatibilidad, pero poseen además clases específicas para los tipos de dispositivos para los que se definieron.

3.3.1.3 Perfiles

Definen un conjunto de API's que pueden emplearse para desarrollar aplicaciones para una familia particular de dispositivos. El principal objetivo en la definición de un perfil es garantizar la comunicación de las aplicaciones entre un conjunto de dispositivos que soportan el mismo perfil. Un mismo dispositivo puede soportar diferentes perfiles y los perfiles se desarrollan para una determinada configuración.

Sobre CLDC y CDC se han estandarizado: **Mobile Information Device Profile** (MIDP) [9] para teléfonos móviles, pagers y PDAs de bajo costo. **Personal Digital Assistant Profile** (PDAP) para asistentes personales. La diferencia añadida respecto al anterior es la parte grafica que es menos limitada.

3.3.2 MIDP

MIDP (Mobile Information Profile) es uno de los perfiles que pueden ser desarrollados bajo la configuración de CLDC y CDC. Los perfiles representan un conjunto de API que son implementadas sobre un conjunto de dispositivos junto a una configuración en particular. Los perfiles pueden extender de paquetes opcionales, y una colección de API especifica una tecnología en particular. La versión actual de MIDP es la 2.0, pero se apronta la versión 3.0, que ya es considerado a ser el perfil por defecto de los nuevos dispositivos que saldrán al mercado.

Ligadas a la arquitectura J2ME, existen una serie de APIs que se han ido definiendo para cubrir funcionalidades específicas de determinados tipos de dispositivos. La mayoría de ellas pueden emplearse como extensión a cualquier perfil J2ME porque se han desarrollado sobre la parte común de CLDC y CDC, las más importantes son:

- **Java APIs Bluetooth**, 2002, que permite el uso de conexiones Bluetooth y da soporte al protocolo Object Exchange (OBEX).
- **Wireless Messaging** API, 2003 que proporciona un API para el envío y recepción de Short Messaging Service (SMS).
- **Mobile Media API,** 2003, que da soporte a procesado multimedia.

Configuración	Perfil	Dispositivo	Requisitos Mínimos
CLDC	MIDP	Teléfonos móviles, pagers, PDAs de bajo costo.	256 KB ROM 128 KB RAM
	PDAP	PDAs que gestionan información personal y que se sincronizan con PC u otras PDA.	1 MB (ROM + RAM)
CDC	FP	Dispositivos embebidos sin interfaz de usuario	1 MB ROM 512 KB RAM
	PBP	Dispositivos con interfaz de usuario sencilla, electrónica de consumo del hogar: lavadoras, neveras, y de oficina, faxes, impresoras, dispositivos de automóvil. Construido sobre el FP.	2 MB ROM 1 MB RAM
	PP	Dispositivos con una interfaz gráfica más compleja, como set-top boxes, televisiones, PDAs de alto coste. Construido sobre el PBP	2.5 MB ROM 1 MB RAM

Tabla 1 - Configuraciones y perfiles J2ME

3.4 JSR-82

Este API está dividida en dos partes: el paquete javax.bluetooth y el paquete javax.obex. Los dos paquetes son totalmente independientes. El primero de ellos define clases e interfaces básicas para el descubrimiento de dispositivos, descubrimiento de servicios, conexión y comunicación [17].

La API JSR-82 actualmente soporta:

- **SDAP – Service Discovery Application Profile**: servicio orientado a descubrir dispositivos y servicios que trabajen sobre Bluetooth.
- **L2CAP - Logical Link Control and Adaptation Protocol**: protocolo de bajo nivel, se envían y reciben arreglos de bytes.
- **RFCOMM - Serial Cable Emulation Protocol**: protocolo construido sobre L2CAP, los datos se envían y reciben por medio de ficheros de entrada y salida, muy similar a los sockets.
- **GOEP - Generic Object Exchange (OBEX) Profile**: Este protocolo es muy similar a HTTP y es utilizado sobre todo para el intercambio de archivos.

3.5 Trabajos relacionados

Los siguientes trabajos y proyectos de investigación están relacionados a la arquitectura que se obtuvo, por lo que es importante mencionarlos y tenerlos en cuenta. Muchas de sus conclusiones y avances fueron tomados para poder obtener un mejor producto final.

3.5.1 GA P2P network framework

El proyecto GA P2P Network Framework [3] busca ofrecer un framework de comunicaciones y contexto para la construcción de juegos multiusuario en entornos P2P sobre teléfonos celulares. Principalmente se trató de realizar un framework de comunicaciones P2P sobre Bluetooth, para poder hacer esto, se trabajó sobre la capa de aplicación (Modelo OSI) [16], para poder trabajar como red P2P, dado que Bluetooth trabaja sobre una arquitectura cliente servidor a nivel de capa de red. Para la comunicación entre dispositivos se enviaban mensajes XML, el cual fue un verdadero revés para el proyecto dado que para un dispositivo móvil realizar los pasos de crear un XML, serializarlo, enviarlo, recibirlo, des-serializar y regenerar un XML es demasiado costoso. El proyecto actualmente está parado.

Ventajas:

- El tiempo de desarrollo se disminuye ya que hay que implementar menos líneas de código.
- El desarrollador de videojuegos puede dedicarse a la lógica del negocio sin tener que preocuparse en los aspectos de comunicación.
- Agrega algunas funcionalidades al videojuego, las cuales son trasparentes para el desarrollador del juego, como sincronización de objetos o variables comunes en el videojuego, recuperación de la sesión del videojuego.

Desventajas:

- Toda la información que envía este framework es a través del lenguaje XML, lo que provoca un gran costo de procesamiento para las máquinas virtuales de dispositivos móviles. Provocando una lentitud importante al momento de procesar los mensajes.
- La falta de uso de métodos estándares (Se trabajó mucho sobre perfiles Nokia), quitándole portabilidad.
- Se trabajó P2P a nivel de capa de aplicación, lo cual es más costoso por el overhead que este provoca.

3.5.2 Developing a bluetooth-enabled J2ME game

Proyecto de investigación dedicado al diseño y desarrollo de juegos móviles, su implementación se realizósobreJ2ME. El principal énfasis se sitúa en el análisis y la implementación de la tecnología de comunicación inalámbrica bluetooth [8].

El proyecto como resultado final anexa el código de un conocido juego para móviles, del estilo cartas, llamado *Top Trumps* y la descripción de los tópicos que se deben tener en cuenta al momento de desarrollar videojuegos en entornos móviles.

3.5.3 JXTA (Juxtapose)

Es una plataforma P2P que define un conjunto de protocolos basados en XML, inicialmente creada por Sun Microsystems en el año 2001 [4]. JXTA Permite que los dispositivos conectados a una red intercambien mensajes entre sí. JXTA es el framework P2P más maduro que actualmente existe. Además, fue diseñado para permitir que un amplio rango de dispositivos (computadoras, teléfonos móviles, PDAs) se comunique de forma descentralizada.

4. Metodología de trabajo

Debido a la complejidad del tratamiento y análisis, así como también a la no posibilidad de capturar todos los requerimientos de una sola vez, se optó por usar el paradigma evolutivo de software para el desarrollo de la arquitectura. Este paradigma permite desarrollar versiones cada vez más completas del sistema a medida que se descubran nuevas funcionalidades o surjan necesidades de cambiar y mejorar aquellas ya existentes, es decir iterando en cada nueva etapa en que el software se encuentre.

Las librerías, por naturaleza, no tienen un usuario final específico, y el diseño evolutivo ayuda a que una retroalimentación [14] pueda ir mejorando y desarrollando nuevas versiones de acuerdo a las necesidades que vayan surgiendo, a través de un ciclo de vida iterativo.

A nivel global, la arquitectura se descompone en partes, o componentes, y estas van evolucionando independientemente unas de otras, surgiendo iteraciones e incrementos. Con este paradigma de trabajo se logra una evolución paralela del sistema, adaptándose a las dificultades propias del problema.

Para los prototipos, por su estricta dependencia a las funcionalidades de la librería, se usó el paradigma desechable. Cada vez que se haga un cambio importante sobre la arquitectura o una evolución, se estudia si la modificación es válida para implementarla en el nuevo prototipo, y de ser así, se tomaran en cuenta para el desarrollo del prototipo desechable siguiente.

Pensando en las interacciones que el usuario pueda tener con la arquitectura y el prototipo, se usó el paradigma de análisis y diseño orientado a objetos, usando UML como lenguaje de modelamiento, pues describen las especificaciones de los sistemas. En cuanto a la implementación. Sin embargo, como parte del diseño, se define un estándar de creación de interfaces para lenguajes orientados a objetos que puedan comunicarse con la librería, y usar todas sus funcionalidades con otras tecnologías, como Java.

5. Análisis

5.1. Comunicación entre dispositivos

El establecimiento de comunicaciones entre objetos que se ejecuten en distintos procesos (ya sea en un mismo equipo o en Internet) es un objetivo habitual en programación, especialmente a la hora de generar aplicaciones de amplia distribución. Para ello, solía ser necesario un conocimiento profundo no sólo de los objetos a cada extremo de la secuencia de comunicación, sino también de un host de protocolos de nivel inferior, de interfaces para la programación de aplicaciones y de herramientas o archivos de configuración. En suma, se trataba de una tarea compleja que exigía un grado considerable de experiencia y atención. El objetivo principal de los módulos de la arquitectura es abstraer al desarrollador y brindarle herramientas de alto nivel que simplifiquen la complejidad y den una herramienta vital de apoyo. Dado que las tecnologías de las comunicaciones van creciendo y aumentando día a día, se planteó un modelo modular capas de ser extendido y re implementado en el tiempo, siempre con el objetivo de ofrecer mejor y mayor extensibilidad a los video juegos orientados en sesiones compartidas de juegos.

5.1.1. Bluetooth

La razón principal por la que se utilizó esta tecnología como medio de comunicación es el hecho de que bluetooth es una tecnología económica, en la actualidad puede encontrarse en la mayoría de los dispositivos móviles y además trabaja sobre frecuencia de 2.4GHz la cual pertenece a una de las Bandas de Radio Industrial Científica y Médica (ISM) la cual no requiere Licencia y puede ser utilizada con facilidad y de forma gratuita [13].La transmisión de datos se hace mediante canales abiertos, lo que facilita su uso y difusión. Para hacer uso de ella solo es necesario que los dispositivos en cuestión cuenten con el chip interno incorporado.

5.1.2. Búsqueda de servicios y dispositivos

Dentro de los tópicos más importantes de la comunicación entre videojuegos, es necesario tener las herramientas adecuadas para poder encontrar otros dispositivos y a su vez también sesiones de juegos, independiente del medio o protocolo que se tenga.

5.2. Singleton

El patrón Singleton [7] es utilizado cuando se desea asegurar que existe una y sólo una instancia de determinada clase en toda la aplicación. 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 y centralizado.
- Las librerías contarán con una instancia de cada servicio, con el fin de mantener controlada la atomicidad de las operaciones y consistencia del estado de cada servicio, además de un reducido consumo de memoria.

5.3. Diseño modular

Uno de los objetivos principales de la arquitectura fue siempre tener un diseño ordenado y altamente cohesionado de los módulos, donde los roles de cada módulo estén bien definidos en función de lo que realizan. El objetivo final es tener un modelo de arquitectura simplificado, ordenado, altamente funcional, independiente y ofrezca la posibilidad de extenderse, para mantenerse actualizada en el tiempo.

5.4. Extensibilidad

Para poder garantizar la extensibilidad de la arquitectura, se definen clases abstractas que contengan un molde funcional adecuado y claro para implementar, que sean funcionalmente independientes entre módulos y conservar la transparencia de la tecnología utilizada.

5.5. Control del almacenamiento

El poder almacenar y conservar la integridad de la información es un tema muy relevante para cualquier aplicación móvil, ya que los medios que cuentan para poder realizarlos son limitados y requieren de un consumo de energía bien relevante, además que no se cuentan con recursos tan poderosos como los que contienen los computadores, por lo que es necesario definir estrategias necesarias para tratar el almacenamiento en distintos escenarios, necesidades y contextos para poder desarrollar videojuegos de forma optima y transparente e independiente de si el medio de almacenamiento es parte del dispositivo móvil o si es ofrecido por un tercero.

5.5.1. Memoria principal

El tener un mecanismo para poder administrar y controlar la memoria principal se hace importante, dado la capacidad limitada de los dispositivos es necesario tener un ente administrar que tenga la visibilidad completa de esta y su regulación. Sin embargo también

es necesario considerar los elementos tendrán distintos distintas necesidades de permanencia en la memoria e incluso prioridades, por lo cual se hace necesario tener mecanismos para poder establecer prioridades de asignación e importancia en el uso de la memoria.

5.5.2. Memoria persistente

Muchos contenidos de los videojuegos se encontraran almacenados en la memoria persistente del dispositivo, y serán utilizados cuando estos sean necesarios, se tendrá en cuenta un modelo de utilización y acceso a la memoria persistente de forma transparente en su utilización.

5.5.3. Almacenamiento remoto

El acceso a servidores o estaciones de trabajo de terceros ya no es solamente algo que puedan hacer los computadores, los dispositivos móviles ya tienen las herramientas y tecnologías necesarias para poder hacerlo y de cualquier punto. Por lo que es necesario contar con un acceso transparente y claro para acceder a estos recursos.

5.6. Seguridad

5.6.1. Encriptación

Para que la información sea solamente entendida por el destinatario correspondiente, asegurar que no existan intromisiones en los mensajes o información y conservar la privacidad de las sesiones de juego, es que se necesitan medios y mecanismos necesarios para poder generar encriptación de los datos y sesiones de juegos, siguiendo la misma convención de diseño de la arquitectura, fácil acceso e independiente del modelo que se utilice para encriptar.

5.6.2. Control de sesiones

Las sesiones de juego siempre pueden estar expuestas a cualquier ataque o violación por un medio externo que quiera aprovecharse de estas, es necesario contar con mecanismos necesarios para poder conservar la integridad de las sesiones y sus participantes.

5.7. Requerimientos y restricciones de la arquitectura

5.7.1. Restricciones

En la siguiente tabla están las restricciones del proyecto.

Código	Restricción
R1	Como protocolo de comunicación para trabajar debe ser Bluetooth.
R2	El lenguaje de programación para desarrollar debe ser J2ME.
R3	Las librerías deben ser soportadas por los teléfonos celulares.
R4	Se debe tener un grado de modularidad acorde a la funcionalidad de cada módulo.

Tabla 2 - Restricciones

5.7.2. Requerimientos funcionales

Representan todas las funcionalidades básicas que debe cumplir la arquitectura, las cuales se encuentran enumeradas en la siguiente tabla:

Código	Requerimiento
RF1	La arquitectura debe contemplar las necesidades básicas para comunicarse y conectarse con otros dispositivos.
RF2	La arquitectura debe contemplar la creación de sesiones de juego y preservar su integridad.
RF3	La arquitectura debe permitir que un posible jugador pueda buscar una sesión de juego a través de la búsqueda de servicios y dispositivos.
RF4	Las sesiones deben asegurar el ingreso y salida de jugadores, como a su vez la continuidad de esta.
RF5	La arquitectura debe permitir a la opción de rechazar jugadores que

	ingresen a la sesión de juego.
RF6	La arquitectura debe permitir aceptar jugadores que ingresen a una sesión de juego.
RF7	La arquitectura debe proveer una serie de mecanismos de sincronización de los recursos que administre en la sesión de juego (variables y objetos).
RF8	La arquitectura debe notificar sobre las fallas ocurridas.
RF9	La arquitectura debe permitir encapsular las direcciones físicas.
RF10	La arquitectura debe notificar al jugador si su dispositivo móvil no contiene los requerimientos técnicos mínimos para poder ínter operar con otros jugadores
RF11	La arquitectura debe soportar los cambios repentinos que tengan los jugadores dentro de la sesión de juego.
RF12	La arquitectura debe tener un servicio de mensajería para poder comunicar jugadores de la sesión de juego.
RF13 *	La arquitectura debe tener una entidad que sea mediadora entre los jugadores, con el objetivo de que estandarice algunas particularidades de los dispositivos.
RF14	Se debe poder modificar la capacidad máxima de jugadores en una sesión de juego.
RF15	Se necesita una entidad que controle y administre las solicitudes de comunicación.
RF16	La arquitectura debe contar con una entidad única para poder administrar y controlar el acceso de los videojuegos a cualquier servicio proporcionado por esta.
RF17	Se necesita una entidad que tenga el conocimiento, control y sepa operar con los distintos tipos de memoria de los dispositivos móviles.
RF18	Se requiere de una entidad encargada de ofrecer servicios de encriptación y seguridad para las operaciones básicas de comunicación y control de las sesiones de juego.

RF19	Se requiere un componente que almacene y ofrezca moldes o esqueletos de lógicas e IA para cualquier tipo de juegos.
RF20	Los servicios principales deben ser ofrecidos bajo el patrón de diseño Singleton [7]

Tabla 3 - Requerimientos funcionales

R13: Como la mayoría de los dispositivos móviles tienen distintas características (como los tamaños de pantalla), la arquitectura tendrá una entidad que solicite a los dispositivos involucrados en la sesión de juego, sus características, para que la entidad pueda evaluarlas y ver si dadas las características de ambos dispositivos móviles son aptas para poder establecer un juego.

5.7.3. Requerimientos no funcionales

Son los que brindan soporte a todos nuestros requerimientos funcionales, los a continuación serán enumerados:

Código	Requerimiento	Tipo	Métrica
RNF1	Los componentes de la arquitectura deben tener documentación asistida por javadoc.	Usabilidad	Una especificación por método y atributos.
RNF2	La arquitectura no debe permitir mensajes que no pertenezcan a su sesión	Seguridad	El UUID debe pertenecer al servicio de cada sesión.
RNF3	Tasa de envió de mensajes debe tener una velocidad fluida, y el refresco debe ser casi inmediato.	Comunicación	Cantidad de frames divididos por segundos.

Tabla 4 - Requerimientos no funcionales

5.8. El ciclo principal de los videojuegos

Desde los principios de los videojuegos surgían las dudas de cómo controlar el ciclo principal (la vida del videojuego) y qué condiciones debían ocurrir para que el juego refrescara el cambio o cuando debiese terminar.

La respuesta a esta interrogante era simple, formar un bucle (90% de las veces un while) que dentro contuviese las acciones realizadas por los participantes del videojuego, el procesamiento de estas, coordinación entre los participantes, envió de la información y actualización del escenario en cuestión.

La condición de este bucle estaba representaba generalmente por las siguientes condiciones: “mientras el juego continué”, “mientras el jugador esté vivo” o “mientras aún queden objetivos por cumplir”. Si estas premisas no se cumplían se generaba el quiebre del bucle y muchas veces el termino del juego o el reinicio de este.

A continuación en la figura 6 se aprecia el bucle principal de los videojuegos.

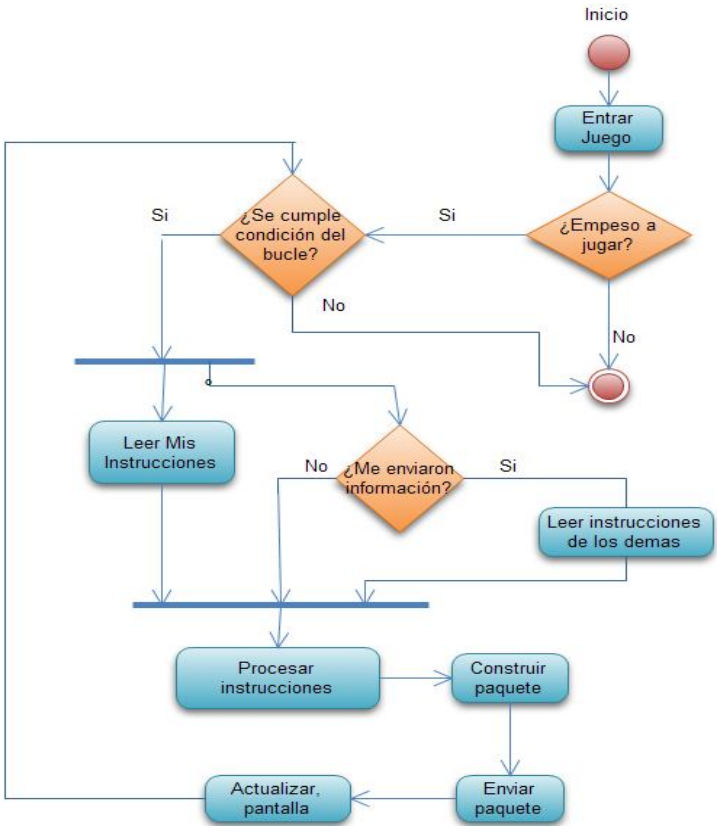

Figura 6 - Ciclo principal de los videojuegos

5.9. Definición de la arquitectura

La presente arquitectura se define como un conjunto de librerías y paquetes que brinden, a través de un conjunto de módulos con funcionalidades específicas, un soporte y ayuda directa a aquellos desarrolladores de video juegos con soporte para multijugador en dispositivos móviles.

Diferentes funcionalidades se encontrarán al momento de usar esta arquitectura, dentro de las más importante se encuentran la administración y búsqueda de servicios, la gestión de la comunicación, sincronización y la creación de sesiones de juego para la interacción de dos o más dispositivos, también el procesamiento de los juegos o configuración de estos, y el desarrollo de la lógica, entre otros.

A través de esta arquitectura los desarrolladores tendrán que dedicarle menos tiempo a la programación y de esta manera ganar un mayor tiempo en el diseño de mejores interfaces más enfocadas al usuario de juegos de la actualidad.

5.10. Módulos

5.10.1. Módulo de procesamiento de juegos

El módulo de procesamiento de juegos, o más conocido como el núcleo propio de la arquitectura, fue el módulo más importante de esta arquitectura, debido a que se encarga de la configuración general de cada uno de los juegos que son montados sobre la arquitectura y de esta misma.

Este módulo controla las partidas o sesiones que se creen una vez cargados los juegos, al mismo tiempo se encargara de comunicarse internamente con otros módulos de la arquitectura y recibir información pertinente a las tecnologías soportadas por los móviles, un ejemplo claro de esto es el tipo de protocolo de comunicación, por lo cual el módulo de procesamiento debe ser capaz de solicitar información al módulo de comunicación acerca de los protocolos soportados para los dispositivos que quieran ingresar a las partidas, los cuales podrían ser Bluetooth, Wi-Fi (IEEE 802.11), o Wi-Max(IEEE 802.16)

Las funciones principales del módulo de procesamiento se encuentran en administrar los servicios ofrecidos por la arquitectura, así se puede llevar un control de estos y mantener sus estados intangibles a entidades externas. El módulo también cuenta con clases y librerías utilitarias para ayudar en las labores de programación.

5.10.2. Módulo de persistencia

El módulo de Persistencia será el encargado de la administración del almacenamiento de información trascendental para los juegos en el dispositivo, a través de la memoria interna del dispositivo móvil, persistente y también externa por distintos medios de comunicación.

Entre las principales funcionalidades de este módulo se encuentra el manejo de objetos representativos y comunes dentro de los videojuegos (automodificables y configurables) como tablas de record de puntajes, estadísticas de tiempos de partidas, la posibilidad de guardar escenarios y animaciones con personajes completas en el dispositivo, incluso música, dentro y fuera del dispositivo.

5.10.3. Módulo de lógica

Este módulo está ligado directamente con los juegos desarrollados y su lógica, es decir, se encargara de la gestión y control de las características más importantes que tienen los juegos y a su vez los diferencian unos de otros. Dependiendo el tipo de juego que el programador desarrolle, es como trabaje este módulo, debido a que debe tener un control de los niveles que los juegos brinden, la cantidad de vidas o muertes que pueden tener eventualmente los personajes de un juego, entre otros.

Existe una gran gama de tipos de juego, y ahora en la actualidad se han desarrollado muchos para dispositivos móviles, dentro de estos tipos de juego encontramos algunos como los de disparo (Shoot'em Up), juegos de estrategia, de ingenio, de rol, de aventuras, entre otros.

Los paquetes de librerías serán enfocados al apoyo de la construcción de éstos distintos tipos de videojuegos y la arquitectura brindará un soporte para el desarrollo de esta gran gama de juegos, y de esta manera, pre estableciendo configuraciones que en diferentes tipos de juegos varíen mucho, como por ejemplo en un tipo de juegos shoot'em up, validar el número de vidas que un personaje puede sobrevivir, o en un juego de cartas, controlar el monto que un jugador pueda apostar, o un juego de rol cuando un jugador puede morir, o contabilizar el número de las vidas que un personaje lleva en un juego específico.

5.10.4. Módulo de seguridad

Este módulo se encargará de la seguridad al momento de jugar, como sucede en las aplicaciones de escritorio que se producen problemas de este tipo, en aplicaciones móviles puede pasar lo mismo. Es por esto que este módulo debe encargarse de resolver sistemas de seguridad para los diferentes tipos de juegos que se deseen desarrollar.

Dentro de las funcionalidades específicas que soportará este módulo deben existir sesiones con contraseña, es decir juegos que necesiten identificarse previamente. También juegos que accedan a servicios web o acceso a Internet cuenten con un nivel de seguridad aceptable, la arquitectura no permita el ingreso a una sesión de un personaje no invitado, en

otras palabras validar que quien dice ser emisor sea el emisor y quien dice ser el receptor sea el receptor.

En base a lo definido anteriormente podría establecerse una jerarquía de módulos, esto quiere decir que algunos módulos dependerán directamente unos de otros, es por esto que él envió de información parametrizada entre módulos es esencial, este proceso hace fuerte uso de la modularidad e interconexión entre módulos.

5.10.5. Módulo de comunicación

El módulo de comunicación es el comienzo de esta arquitectura, se define como el módulo encargado de asignar la configuración de comunicación entre los distintos dispositivos móviles que sincronicen una partida de juego especifica.

Bajo este módulo se especificará el protocolo por el cual todos los dispositivos deberán conectarse a una partida, estos protocolos podrían variar, dentro de los protocolos más conocidos encontramos Bluetooth, que es aquel protocolo elegido para las pruebas en los prototipos, pero a este se podría agregar el soporte para Wi-Fi o Wi-Max, entre otros protocolos, extendiendo la especificación.

La finalidad es que sin importar el protocolo que se elija la arquitectura brinde un soporte masivo, y de cierta forma funcione como una caja negra con los demás dispositivos, envíe y reciba información hacia los otros módulos, y que dentro de sus parámetros en los mensajes se incorpore el protocolo utilizado para una partida determinada, en otras palabras, se quiere un verdadero Middleware [2] entre dispositivos móviles.

El módulo de comunicación dentro de sus funcionalidades tendrá un soporte estandarizado, sin importar el protocolo que se utilice, para la búsqueda de dispositivos y servicios, en un área determinada. También debe encargarse de brindar eficiencia al momento de transmitir la información, es por esto que debe contar con algoritmos de serialización de información, y de esta forma trabajar a nivel de lenguaje máquina, para hacer de esta función un proceso rápido y prácticamente inmediato.

Además el modulo de comunicación debe tener el control de las sesiones en las partidas, donde una sesión comienza desde el momento en que los dispositivos utilizan un juego en una partida multijugador. Es por esto que en este módulo se deben definir los roles de servidor y de una lista de clientes, los cuales se irán agregando en colas a medida que vayan ingresando a la partida.

Como trabajo futuro se espera el desarrollo de módulos que brinden mayor funcionalidad al desarrollo de video juegos, y amplíen las capacidades para los desarrolladores al momento de crear juegos para móviles.

En la siguiente figura se establecen los paquetes de módulos de la arquitectura, donde el paquete de Procesamiento figura como entidad central, encargado de comunicar a los demás módulos y recibir las peticiones del videojuego para devolvérselas a los demás.

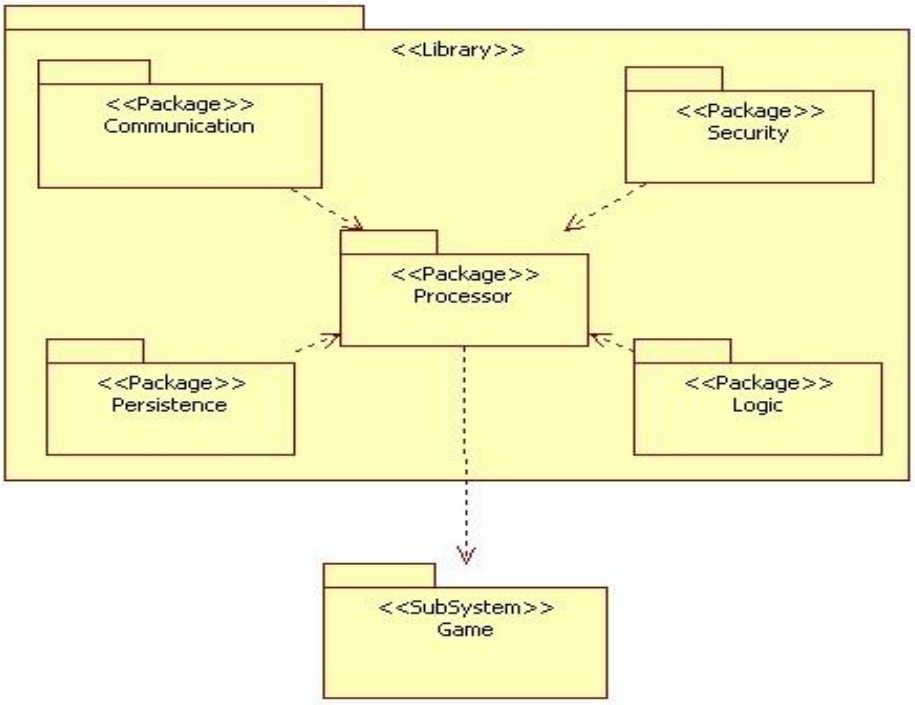

Figura 7 - Diagrama de paquetes

5.11. Integración de la arquitectura

En este proyecto solamente integraremos lo que se desarrolle de ARQlet en el bucle antes definido, dado que es aquí donde mayor se explota la funcionalidad del juego en todos sus aspectos. Tomando en cuenta el bucle anterior e inyectando la participación de ARQlet, obtendremos la siguiente figura con la participación de los módulos.

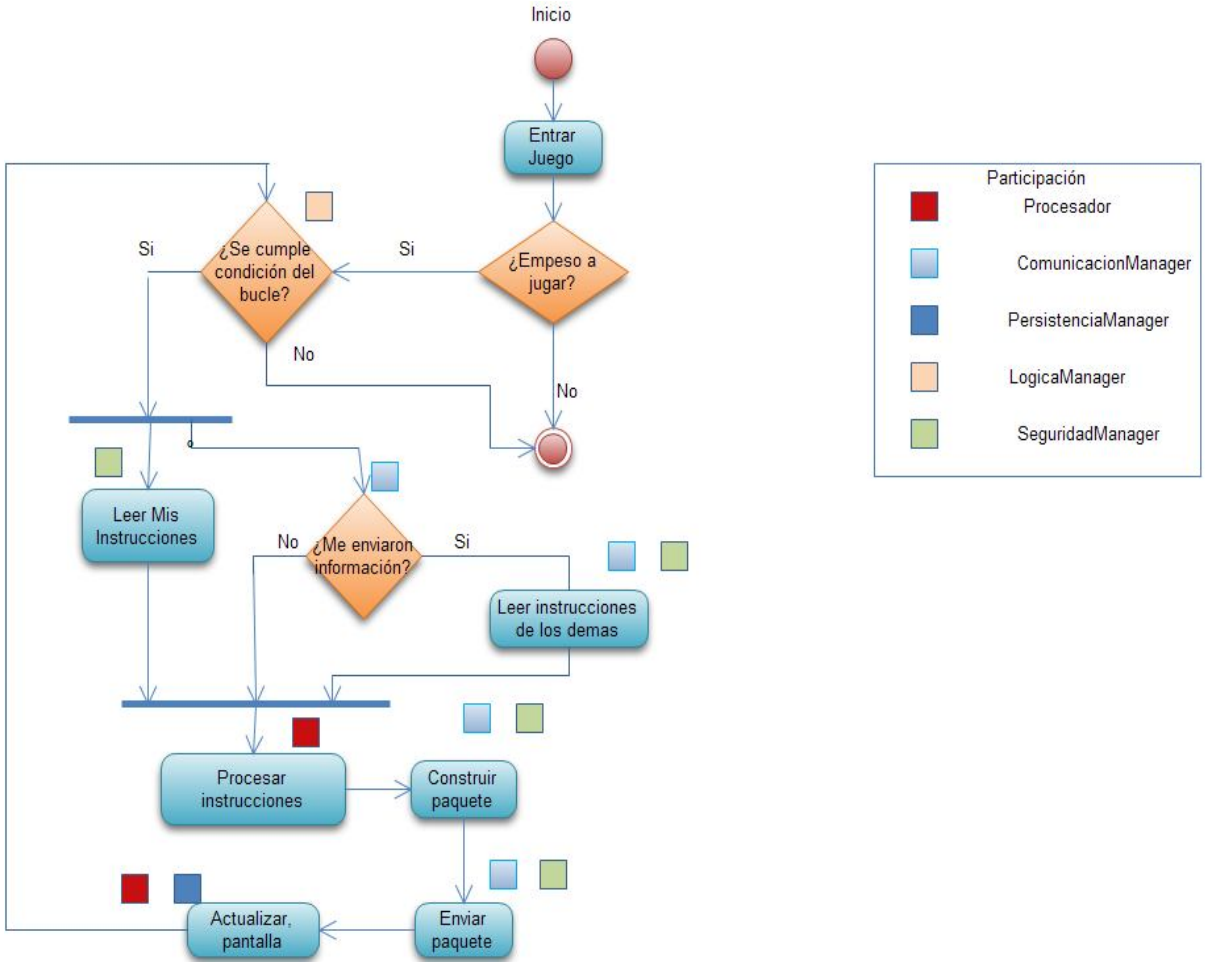

Figura 8 - Ciclo principal de videojuegos con ARQlet

6. Diseño

En este capítulo se detallará el diseño general de la arquitectura planteada en este trabajo, enfocándose en cada uno de los módulos que esta contiene y especificando sus funcionalidades, con el objetivo de lograr un desarrollo eficiente.

Cada uno de los módulos ha sido diseñado con el objetivo de que luego su código fuese reutilizable y pudiese soportar ampliaciones de su implementación, es decir, extensibilidad de funcionalidades.

Para lograr esta tarea en cada uno de sus módulos se diseñaron plantillas o clases abstractas, a las cuales se les denominó ***Manager***. Cada módulo contiene un Manager que encapsula sus principales funcionalidades, las cuales pueden ir de funcionalidades genéricas a métodos muy específicos que dependen de los dispositivos móviles con los que se encuentre trabajando el desarrollador.

Además a cada Manager implementado o extendido se le llama *Driver*, los cuales tienen un uso muy similar a los driver que ocupan las bases de datos (por ejemplo JDBC), los cuales se cargan y utilizan su propia funcionalidad a través de una interfaz estándar [1]. Este trabajo postula algo muy similar, debido a que cada módulo puede contener distintos drivers, los cuales son la implementación de cada uno de los Manager cuando estos han sido cargados.

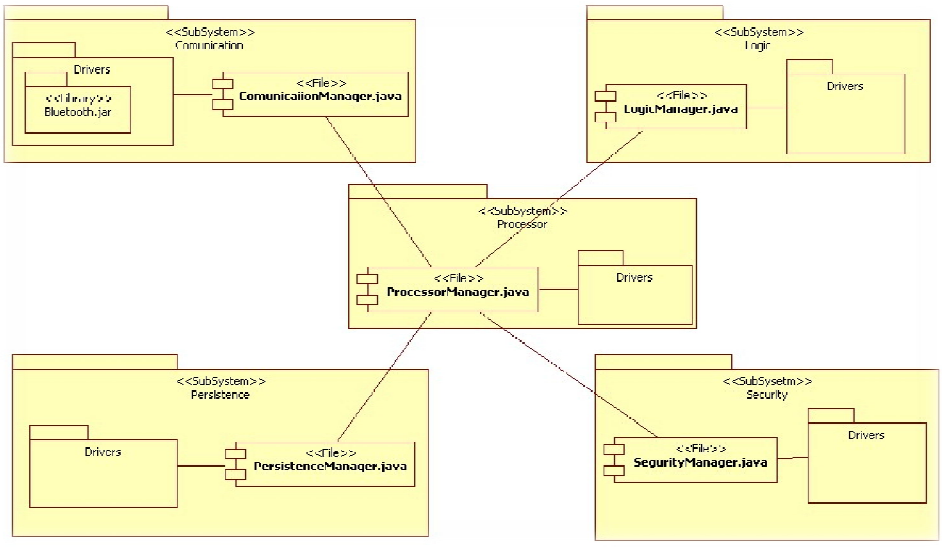

Figura 9 - Diagrama de componentes

6.1. Módulo de comunicación

La orientación principal del módulo de comunicación es brindar un conjunto de funcionalidades orientadas a la formación de redes, la comunicación entre los nodos de esta y su administración por parte del anfitrión. En función del tiempo y del marco del trabajo propuesta al proyecto, se realizaron las siguientes funcionalidades, véase figura 10.

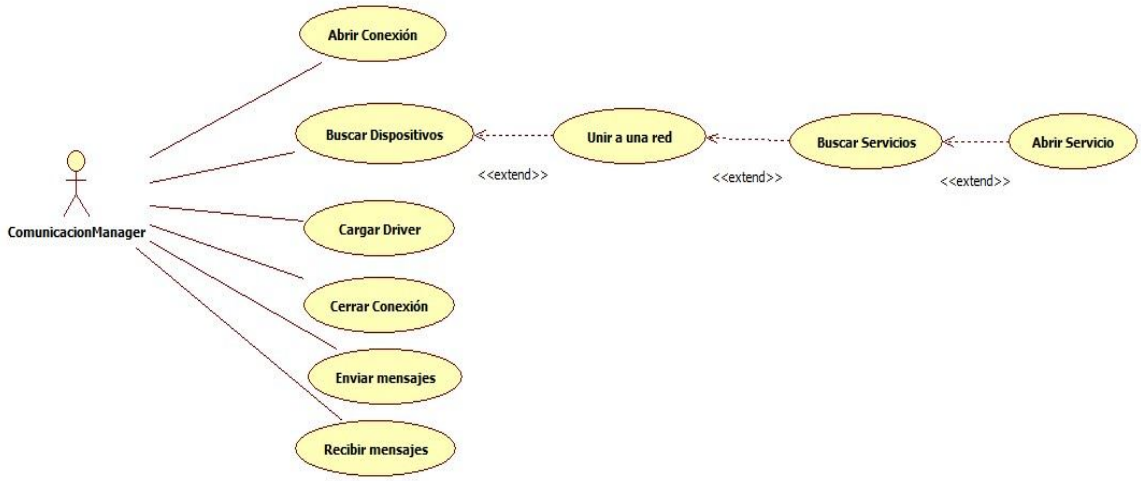

Figura 10 - Funcionalidades de ComunicacionManager

- **Abrir conexión:** Se activa y enciende el dispositivo para trabajar con un protocolo determinado.
- **Buscar dispositivos:** Se buscan dispositivos emisores que trabajen con el mismo protocolo con que se realiza la búsqueda.
- **Unir a una red:** De la lista de dispositivos se escoge uno de estos y se pide una solicitud de ingreso.
- **Buscar servicios:** Se buscan los servicios que ofrece el dispositivo anfitrión de la red.
- **Abrir servicio:** Se abre el servicio escogido con anterioridad y se procede a consumir el servicio.
- **Cargar driver:** Se carga un driver con la implementación con del protocolo que se desea ocupar.
- **Cerrar conexión:** Se cierra la conexión y se desconecta de la red.

- **Enviar menajes:** Se envían mensajes entre los nodos de la red, los cuales pueden ser de tipo: *unicast, multicast* y *broadcast*.

- **Recibir mensajes:** Se recepcionan y procesan los mensajes de la red.

Estas son funcionalidades básicas que se deben realizar dentro de una red, pero claves para luego crear funciones de más alto nivel que cumplan con los requerimientos más orientados al control y, como en este proyecto se centra en una primera aproximación.

6.2. Driver de bluetooth

Para poder cumplir las exigencias del módulo de comunicación se diseño un driver que implementara el manager de comunicación, este se basa en el protocolo de comunicación Bluetooth.

Teniendo en cuenta que Bluetooth trabaja sobre una arquitectura cliente-servidor, se planteo el siguiente modelo del dominio (véase figura 11*)* para poder abordar una solución coherente con las restricciones de trabajo del protocolo.

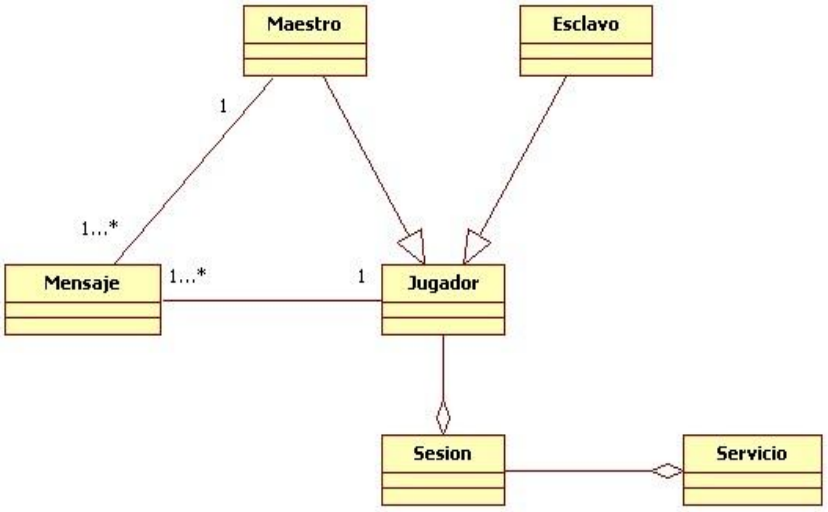

Figura 11 - Modelo del dominio bluetooth

- **Jugador:** Representa a un dispositivo dentro de la red bluetooth, este puede enviar y recibir mensajes a los demas jugadores de la sesión.
- **Maestro:** Es el jugador anfitrión de la sesión el cual ofrece el servicio , acepta a los jugadores *esclavos*que deseeen unirse y administra la sesión que sostiene la red.
- **Esclavo:** Es el jugador que se integra a la sesión de juego, para esto primero debe buscar el servicio solicitado y despues pedir su ingreso.

- **Mensaje:** Unidad con la cual se comunican y coordinan los jugadores.
- **Sesión:** Guarda todos los jugadores, el servicio y sirve de aduana para el envio y recepción de mensajes.
- **Servicio:** Identifica a la sesión y sus características, a través de esta se puede buscar e ingresar a la sesión.

6.2.1. Implementación del driver bluetooth.

En la figura 12 se presenta un diagrama de clases que corresponde al modelo grafico que se implemento en la fase de desarrollo de la arquitectura, se debe destacar que BluetoothDriver es una clase que hereda de ComunicacionManager, y en cada uno de sus metodos este implementa el protocolo bluetooth.

Con el fin de probar y tener una implementación más completa se han implementado dos protocolos de comunicación que trabajan sobre bluetooth L2CAP y RFCOMM , la gran diferencia es que L2CAP trabaja con flujos de byte y RFCOMM es de más alto nivel y trabaja a nivel de sockets. Más adelante se abordan estos protocolos con más detalle.

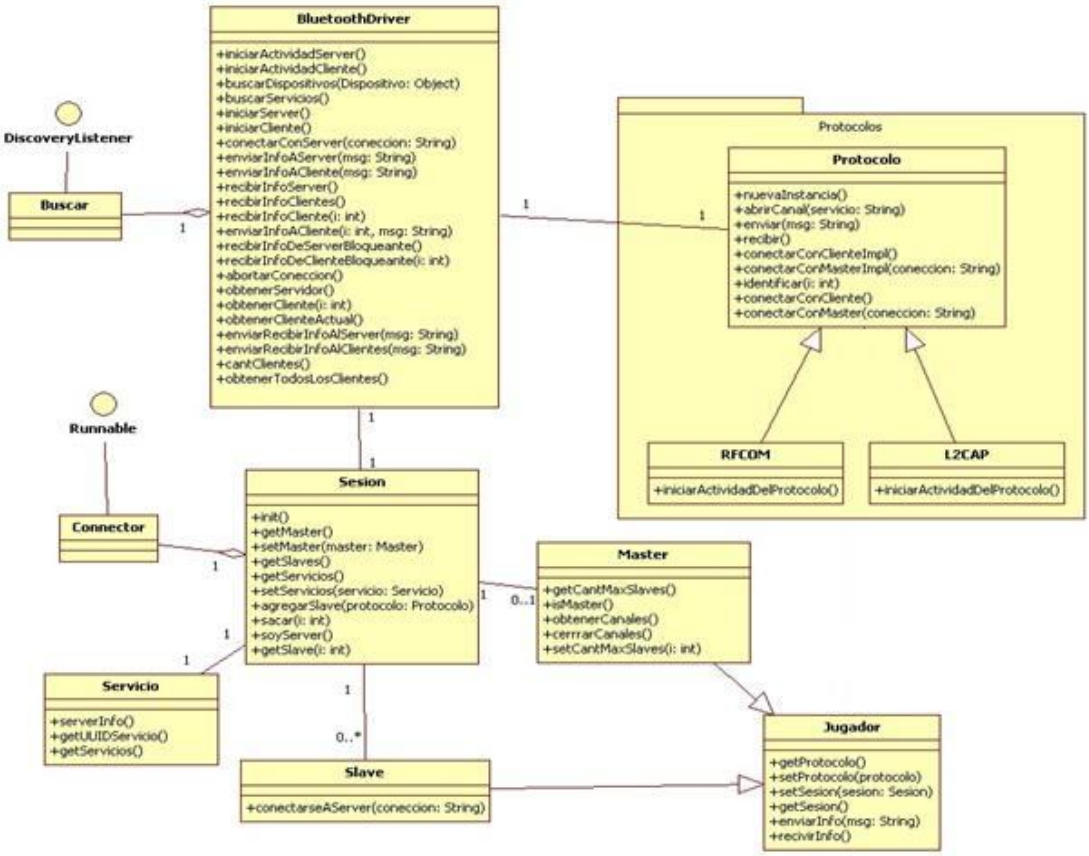

Figura 12 - Modelo de clases bluetooth driver

6.3. Sistema de búsqueda

La librería de bluetooth sobre java (JSR-82) ofrece mecanismos para la busqueda de dispositivos y servicios disponibles, pero al momento de realizar una búsqueda esta se ejecuta sobre una hebra (thread) distinta a la de nuestro programa principal, por lo que al terminar la busqueda o al obtener un resultado está hebra desaparece. El principal problema de este esquema es que no todos los dispositivos tienen un mismo tiempo de busqueda o time out, ocasionando muchas veces la descordinación entre el programa principal y la hebra de busqueda. Dado este problema se plantea una solución ocupando *monitores* [10] por cada entrada a una busqueda hasta la entrega de su resultado

Al entrar en una busqueda la hebra de busqueda de bluetooth bloquea el paso de otras hebras sobre los resultados de la busqueda (en este caso al programa principal), mientras no se termine la busqueda o esta sea interrumpida. Gracias a este enfoque de busqueda con monitores se logra una sincronización correcta entre el programa principal y las busquedas de bluetooth (véase Figura 13).

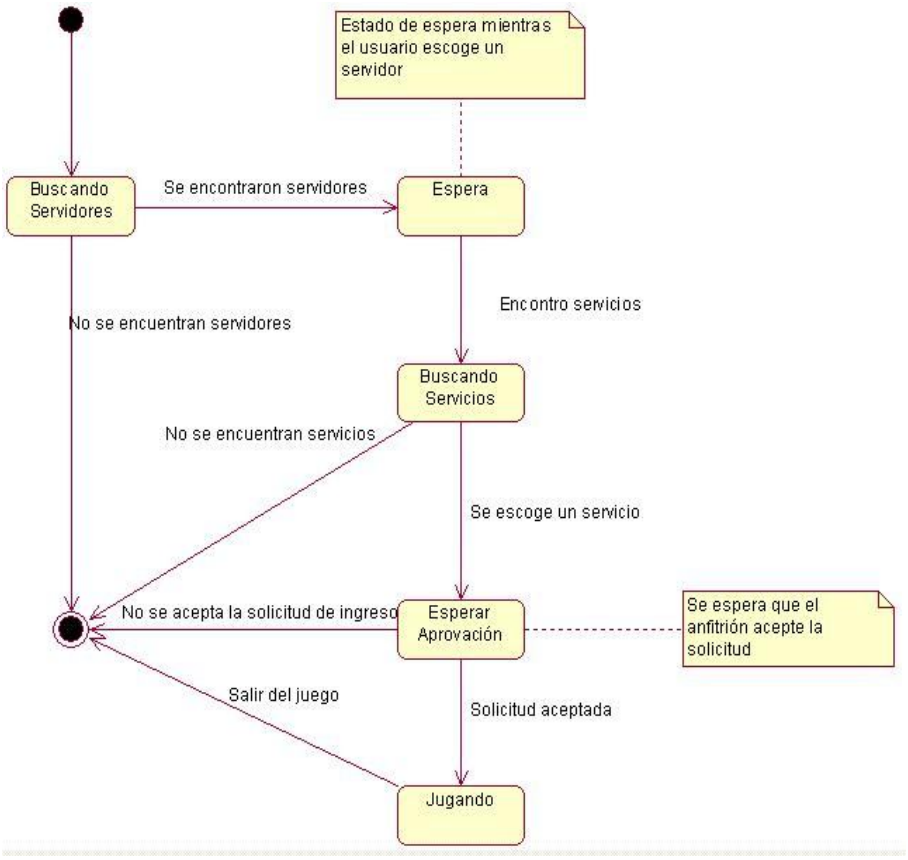

Figura 13 - Sistema de búsqueda

6.4. Creación de una sesión

Una sesión se crea a partir de un anfitrión o master, este a través de la arquitectura inicializa los parametros de la sesión como el protocolo con que va a operar, los servicios ofrecidos, la cantidad de cupos disponibles en la red e incluso si es necesaria una previa autentificación para poder ingresar.
Posteriormente se inicia el servidor, este crea la sesión como tal y la configura dado los parametros antes descritos, despues se crea la unidad maestro sobre la sesión la cual serie el mismo, se crea el servicio y éste se publica. Luego el maestro de la sesión abre los canales para que se puedan ocupar o ser consumidos por posibles usuarios o slaves, por cada uno de estos que se unan se crea una hebra Connector, que los inicializa en la sesión y luego los pone al corriente de esta misma.

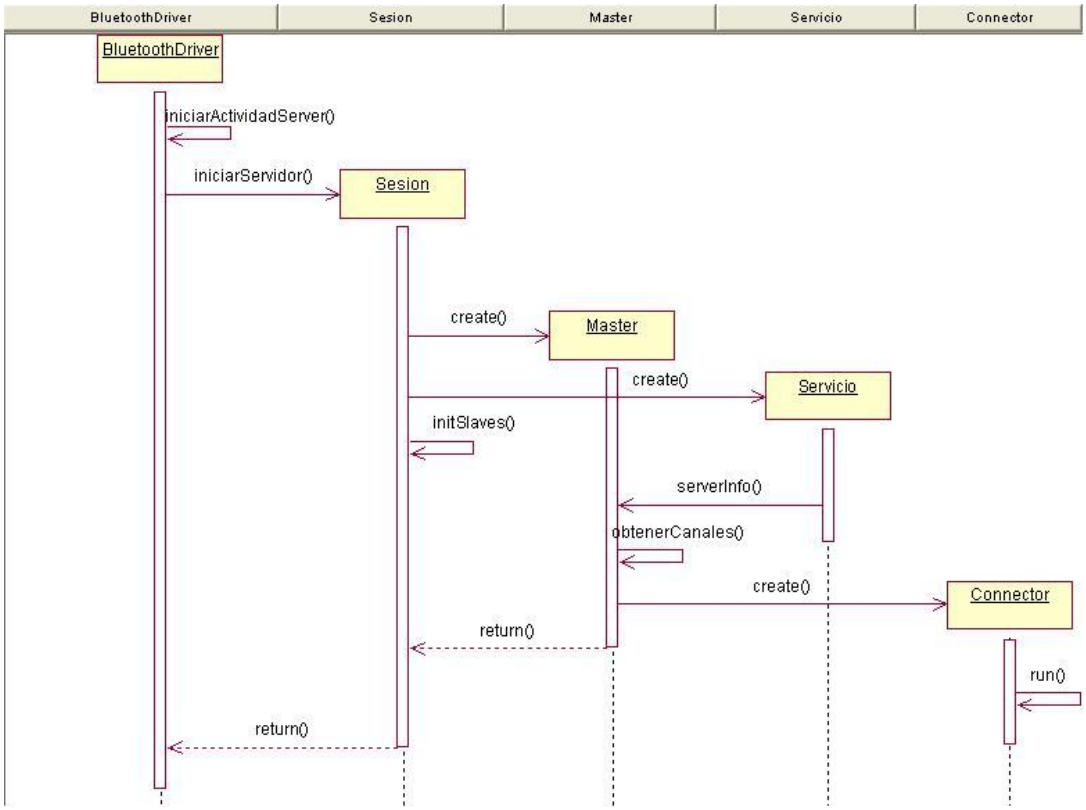

Figura 14 - Crear sesión

7. Desarrollo

En este punto del trabajo se detalla la implementación de la arquitectura propuesta, específicamente el desarrollo del módulo de comunicación, las clases principales del modulo con sus métodos principales, los prototipos desarrollados bajo la arquitectura y sus resultados.

7.1. Protocolos Bluetooth

En este trabajo se plantea el uso de dos de los más importantes protocolos de comunicación en bluetooth: se refiere a los protocolos RFCOMM y L2CAP, la diferencia entre ambos es el mecanismo que utilizan para el envío y recepción de la información, es decir el cómo viajan los datos entre los dispositivos, también en la cantidad de overhead que traen los paquetes de datos.

7.1.1. Protocolo RFCOMM

RFCOMM abreviatura de Radio Frequency Communication (Comunicación por radio frecuencia). El protocolo RFCOMM es un conjunto simple de protocolos de transporte, construido sobre el protocolo L2CAP, es decir se conecta a las capas inferiores de la pila de protocolos bluetooth; y que proporciona 60 conexiones simultáneas para dispositivos bluetooth emulando puertos serie RS-232. El protocolo está basado en el estándar ETSI TS 07.10 [16].

RFCOMM es a menudo denominado *emulación de puertos serie*. El puerto serie de Bluetooth está basado en este protocolo. En base a esto RFCOMM transmite la información con flujos de datos o **datos en serie**, utilizando los métodos InputStream y OutputStream

7.1.2. Protocolo L2CAP

Son las siglas de *Logical Link Control and Adaptation Protocol* (Protocolo de control y adaptación del enlace lógico). Es utilizado dentro de la pila de protocolos de Bluetooth. L2CAP se utiliza para pasar paquetes con y sin orientación a la conexión, los datos los envía y recibe serializados (arreglos de bytes). Las funciones de L2CAP incluyen:

- Segmentación y re ensamblado de paquetes. Acepta paquetes de hasta 64KB de sus capas superiores.
- Multiplexación de varias fuentes de paquetes, comprobando el protocolo de las capas superiores para así adaptarlo antes del re ensamblaje.
- Proporcionar una buena gestión para la transmisión unidireccional a otros dispositivos bluetooth.
- Gestión de la calidad de servicio (QoS) para los protocolos de las capas superiores. En esta fase negocia el tamaño máximo del campo de datos de las tramas. Con ello,

evita que algún dispositivo envíe paquetes tan grandes que puedan desbordar al receptor.

7.2. Singleton

Para garantizar que solamente exista una instancia de cada Manager sobre la ejecución de la arquitectura, se utiliza el patrón de diseño **Singleton,** el cual está diseñado para entregar y garantizar que solamente se tenga una instancia de un objeto o valor dado [7], véase figura 15.

```
/**
 *
 * @author Francisco Reyes C
 * INF-PUCV
 */
public class Singleton {
    // El constructor privado no permite que se genere un constructor
    // por defecto
    private static Singleton instance;
    private Singleton() {}

    public static Singleton getInstance() {
        if (instance ==null)
            instance = new Singleton();
        return instance;
    }
}
```

Figura 15 - Código singleton

7.3. Paquete procesador

7.3.1. Clase ProcesadorManager

Clase encargada de almacenar los Manager de los demás módulos de la Arquitectura permitiendo la intercomunicación entre estos y a través de este se puede acceder a los demás Managers.

7.3.2. Clase Transformador

Clase que contiene métodos estáticos que convierten objetos enmascarados (objetos de la clase Object) en tipos de datos básicos o primitivos.

7.4. Paquete comunicación

7.4.1. Clase CommunicacionManager

Clase abstracta que sirve de molde para implementar cualquier tecnología de comunicación y sus funcionalidades con solamente extenderla e implementar sus métodos abstractos.

7.4.2. Interfaz ObtenerClienteListener

Interfaz orientada al tratamiento de los dispositivos entrantes a la sesión. La clase que la implemente es capaz de controlar la entrada de los nuevos clientes a la sesión.

7.5. Paquete BluetoothDriver

En este paquete se implemento el Driver de bluetooth y todas las clases necesarias para poder trabajar con este protocolo, sistema de búsquedas (de dispositivos y servicios) y administrar la comunicación de los participantes en la red.

7.5.1. Clase BluetoothDriver

Clase que implementa los métodos de la clase ComunicacionManager y a través de sus métodos y un conjunto de clases e interfaces logra implementar y administrar redes en el protocolo bluetooh. BluetoothDriver funciona en base a la arquitectura cliente servidor. Los protocolos de bluetooth implementados para la transferencia de datos son RFCOMM y L2CAP. Cabe mencionar que a través de esta clase se realizan todas las operaciones, tales como:

- Iniciar, editar y cerrar sesiones.
- Búsqueda de dispositivos y servicios.
- Envió y recepción de mensajes.
- Información sobre la red y sus participantes.

7.5.2. Clase Sesión

Clase dedicada a contener a los participantes de la red bluetooth, ósea un servidor y sus respectivos clientes. Esta clase los administra, los comunica y contiene métodos para mantener la integridad de la sesión y sus participantes.

7.5.3. Clase Master

Clase que alberga y representa al servidor de la red, en este caso el que mantiene conexión directa con todos los clientes (slave) de la sesión.

7.5.4. Clase Slave

Clase que representa a un cliente de la sesión, está tiene un enlace dedicado con el servidor de la sesión.

7.5.5. Clase Protocolo

Clase abstracta que encapsula toda la gestión del envío y recepción de datos sobre protocolos que trabajen con bluetooth, pudiendo ser los siguientes tipos: RFCOMM o L2CAP.

7.6. Métodos

A continuación se listan los métodos más significativos de la arquitectura.

public static void crearInstanciaDeComunicacion(String driver, int protocolo)
Crea una nueva instancia de comunicación, donde el primer parámetro driver especifica la tecnología de comunicación que se utilizará (Bluetooth, Wi-Fi, Wi-Max) y el parámetro protocolo indicará el protocolo (L2CAP, RFCOMM) que trabajará sobre el medio de comunicación escogido.

public static ComunicacionManager obtenerInstaciaDeComunicacion()
Retorna la instancia de comunicación almacenada, si no la tiene retorna null.

public ArrayList buscarDispositivos()
Método que busca dispositivos de cualquier tipo que tengan la opción de Bluetooth habilitada, devuelve una lista con todos los dispositivos encontrados. Cabe mencionar que este método es bloqueante, ósea mientras busca dispositivos no se puede realizar ninguna otra operación hasta que se termine de buscar. Retorna un ArrayList con la lista de dispositivos encontrados.

public ArrayList buscarServicios(Object o)
Método que busca servicios en un dispositivo localizado el cual es recibido como parámetro de forma enmascarada con la clase Object. Retorna un ArrayList con la lista de servicios encontrados.

public void iniciarActividadServer()
Método que inicia el agente descubridor para el servidor estableciendo el tipo de visibilidad para los clientes.

public void iniciarActividadCliente()
Método que inicia el agente descubridor para el cliente estableciendo el tipo de visibilidad para los dispositivos clientes y el servidor.

public boolean enviarInfoAServer(String msg) throws IOException
Método que envía un mensaje al servidor

public boolean enviarInfoCliente(String msg) throws IOException
Método que ejecuta un broadcast, enviando un mensaje desde el servidor a todos los clientes.

public String recibirInfoServer() throwsIOException
Método que ejecuta el cliente para leer la información que le envía el servidor.

public String recibirInfoClientes() throws IOException
Método que ejecuta el servidor para recibir la información que le envían todos los clientes dentro de la red.

public void init()
Método que inicializa la lista que albergara a los slaves de la sesión.

public Master getMaster()
Método que retorna el dispositivo master de la sesión actual, si aun no se inicia la comunicación, retornara nulo.

public UUID getSessionUUID()
Método que retorna el identificador único de la sesión.

public ArrayList getSlaves()
Método que retorna la lista que contiene a todos los slaves de la sesión actual.

public void ArrayList estandarizaResolucionPantalla()
Método que contiene la funcionalidad de las medidas de la resolución de las medidas de la pantalla de cada uno de los dispositivos conectados a las partidas de juego, se dedica a calcular un tamaño intermedio para estas y de esta forma estandarizarlas. De esta forma dándole un plus a la dificultad que representa el desarrollo para las distintas gammas de dispositivos móviles.

public Slave getActualSlave()
Método que retorna al slave con que últimamente se está trabajando.

public Slave getActualSlave()
Método que retorna al slave con que últimamente se está trabajando.

public void sacar(int id)
Método que saca al slave i de la lista, por ende de la sesión, puede arrojar excepciones si el

índice sobrepasa el límite de la lista.

public void agregarSlave(Protocolo p)
Método que agrega un slave, este le llega un objeto protocolo que contiene los canales de comunicación del nuevo slave hacia el master de la sesión.

public Servicios getServicios()
Método que retorna un objeto servicios que contiene la configuración del servicio ofrecido por el master o si es cliente, sería el servicio que desea buscar.

public Master(Sesion s, Protocolo protocolo)
Constructor que inicializa al master asignándole la respectiva sesión que administrara y el protocolo que usara para comunicarse.

public Slave getSlave(int i)
Se obtiene el slave i de la lista, de ser el índice i mayor que la lista puede arrojar excepción de desborde del arreglo.

public boolean soyServer()
Método que le devuelve al dispositivo que pregunta en que rol cumple en la sesión.

public int getCantMaxSlaves()
Método que devuelve la cantidad de slaves que puede llegar a soportar el master.

public void obtenerCanales()
Método que genera un thread para gestionar todas las solicitudes de los slaves, a medida que acepta una petición va incorporando los nuevos slaves y dispara el evento ObtenerClienteListener**.**

public void cerrarCanales()
Método que destruye el thread que gestiona las conexiones entrantes y por ende nadie más puede entrar o hacer conexión con la sesión iniciada por el master.

public abstract boolean enviar(String msg)
Método que envía el mensaje al destinatario correspondiente.

public abstract String recibir()
Método que recibe un mensaje del dispositivo con que estableció conexión.

public abstract Protocolo conectarConClienteImpl() throws IllegalAccessException
Método que al implementarse debe realizar toda la lógica de conexión de un master con el cliente con el protocolo que la extiende.

public abstract void conectarConMasterImpl(String coneccion) throws IllegalAccessException
Método que al implementarse debe realizar toda la lógica de conexión de un cliente con el master con el protocolo que la extiende.

7.7. Prototipos desarrollados

En el transcurso del desarrollo del proyecto, se han desarrollado prototipos de videojuegos los cuales hacen uso de la arquitectura con la intención de validar los requerimientos, en cada una de las fases de construcción de ésta misma.
A continuación se listan estos dispositivos, especificando una breve descripción del juego, el objetivo de porque fueron desarrollados, y los resultados obtenidos luego de su construcción.

7.7.1. Prototipo 1: Snake

Descripción: Juego mono usuario, basado en el conocido juego de la víbora para consolas de juego. Donde una serpiente recorre un mapa buscando *elementos* para alimentarse y aumentando su tamaño por cada elemento que consuma, el jugador pierde cuando choca contra las parades del juego o cuando choca contra si mismo

Objetivo del Desarrollo: El juego fue desarrollado con el fin de conocer y utilizar el framewotk que presenta J2ME para el desarrollo de juegos.

Resultados: Se obtiene el conocimiento suficiente para implementar y esbozar los primeros componentes de ARQlet sobre J2ME.

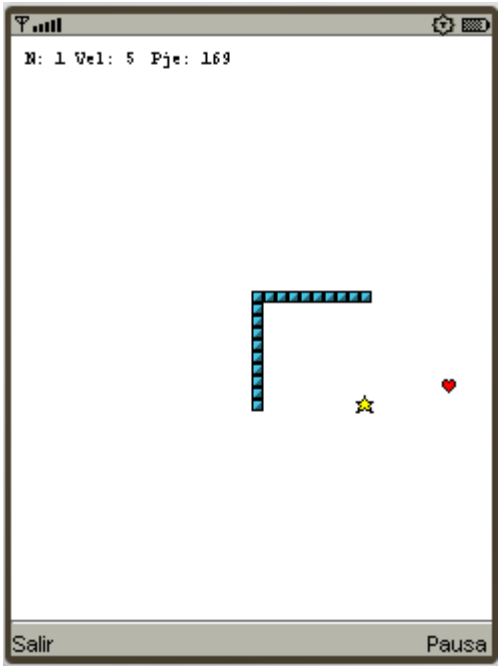

Figura 16 - Prototipo snake

7.7.2. Prototipo 2: Pong

Descripción: Juego para dos jugadores como máximo, basado en el conocido juego pong de atari. Donde dos jugadores hacen rebotar una pelota con el objetivo de anotar en el extremo contrario. Aquel jugador que logre hacer chocar la pelota en la pared de su oponente marca un punto. Gana aquel que obtenga la mayor cantidad de puntos anotados o logre anotar el total de puntos establecidos en la configuracion del juego.

Objetivo del Desarrollo: El juego fue desarrollado con el objetivo de incorporar la librería JSR-82 (bluetooth sobre java) y de esta manera establecer una comunicación entre dos dispositivos móviles mediante protocolos de comunicación bluetooth y despues con la propia implementación de BluetoothDriver de ARQlet. Este prototipo marca el inicio de la programacion multihilo y con una comunicación ya establecida a través de bluetooth en este proyecto.

Resultados: Se crea gran parte de la infrastructura y cuerpo de los paquetes de comunicación y procesador, se crean más formas de enviar/recibir mensajes para organizar las acciones (Se utiliza RFCOMM).

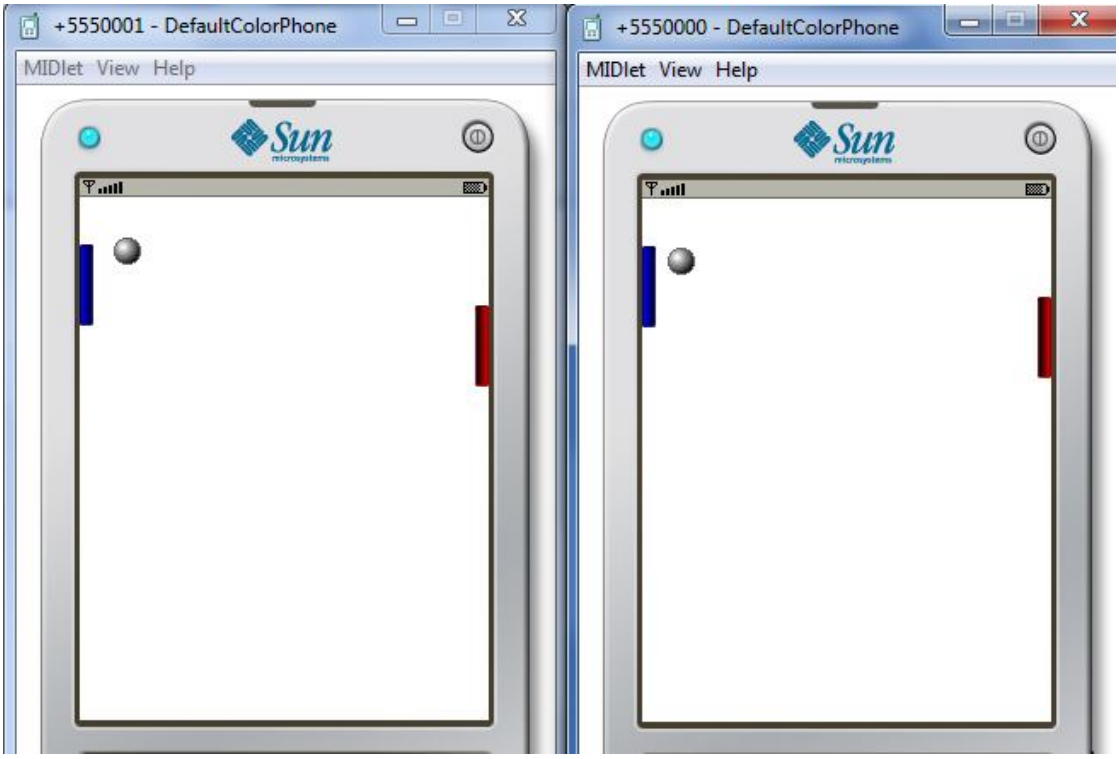

Figura 17 - Prototipo pong

7.7.3. Prototipo 3: Rigth 2 Death

Descripción: Prototipo final desarrollado en base a la arquitectura, este prototipo incorpora una plataforma para multijugador, permitiendo de esta forma interactuar hasta 4 jugadores en una misma partida o sesion de juego.

El juego entra en la categoria shooting y trata de un personaje que debe ir avanzando por un mapa ficticio con el objetivo de ir aniquilando zombies, antes de que ellos lo aniquilen a él.

A medida que el juego avanza y logre continuar con las etapas, el nivel de dificultad también aumentará, incrementando la resistencia de sus atacantes y su velocidad.

Pero durante el mapa el jugador podra ir obteniendo “premios”, los cuales consisitiran en kits medicos con la opcion de que el jugador recupere en parte su energia vital.

Este prototipo solo cuenta con una etapa, en donde solo aumenta la dificultad, pero da soporte para la conectividad entre 4 jugadores donde es el master aquel que crea la partida y slaves aquellos que se unen a esa partida.

Resultados: La comunicación se puede realizar a través de los protocolos de comunicación en bluetooth: L2CAP y RFCOMM . Al utilizar el protocolo RFCOMM y en partidas donde se conectaban mas de dos jugadores ocurría un problema de overhead en la comunicación, ocasionando que el rendimiento de los dispositivos baje. En conclusión podria decirse que la transmision de datos via RFCOMM nos es conveniente cuando el flujo de la

información es muy elevado, en este caso 1 dispositivo master intentado comunicarse con 3 dispositivos slaves.

Por otro lado al implementar la comunicación vía L2CAP el resultado obtenido fue otro, debido a que este protocolo utiliza un mecanismo de serialización de los datos (la informacion viaja a nivel de bytes) y además de ser un protocolo de bajo nivel, se evitan considerablemente los problemas de overhead permitiendo así aumentar el rendimiento en la red, y estableciendo una partida de juego mas fluida, excenta de problemas de comunicación.

Nota: Debido a que el procesamiento se realiza por el lado del servidor, en algunas ocasiones los clientes que se conectaban a las partidas de juego obtenian resultados con desfase en el pintado de las pantallas, debido a problemas de sincronizacion en el protocolo.

A pesar de que los resultados no son del 100% optimos, se plantea como inquietud y para un trabajo futuro la incorporacion de **relojes vectoriales**, los cuales brinden un procesamiento mas ordenado en el accionar de los threads de los juegos, independiente del protocolo de comunicación que se utilice, esto permitirá evitar estas desincronizaciones o los famosos lags que sufren dispositivos en las partidas de juego.

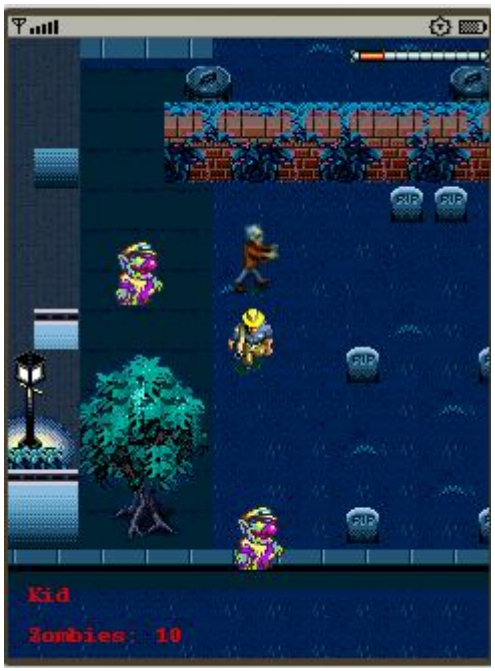

Figura 18 - Prototipo Rigth 2 Death (1 jugador)

Figura 19 - Prototipo Rigth 2 Death (4 jugadores)

8. Pruebas

Las pruebas de software respaldan y validan el resultado de la arquitectura. Las pruebas se dividieron en dos categorías:

- **Funcionalidad:** asegura la calidad de cada una de las funcionalidades desarrolladas.
- **Rendimiento:** asegura el buen uso de los recursos computacionales de los dispositivos móviles, y de esta manera obteniendo resultados eficientes.

Se comienza por diseñar un caso de prueba para una función en particular, por ejemplo, los parámetros y salidas, luego se implementa la función en el lenguaje. Una vez que compila sin errores ni ***warnings,*** se procede a probar usando el caso de prueba ya creado. Si no supera la prueba, se analiza si las condiciones del escenario son las correctas, si la implantación del prototipo es adecuada para el dispositivo móvil o si el error es por parte del sistema operativo del dispositivo móvil.

8.1. Metodología de las pruebas

Para la realización de las pruebas se realizaron los siguientes pasos:

- Se instala el prototipo sobre el teléfono celular.
- Se procede a establecer el escenario de pruebas.
- Se ejecutan las aplicaciones.
- Se comprueba la veracidad de las pruebas.

8.2. Herramientas

Para la realización de las pruebas se utilizaron los siguientes materiales:

- Sun Java Wireless Toolkit for CLDC.
- 2 dispositivos móviles Nokia 5200.
- J2ME Unit Test.
- Netbeans 6.7.1.

Figura 20 - Casos de pruebas

8.3. Pruebas de funcionalidad

Cada vez que se desarrolla un software, es fundamental asegurarse que se comporte como se espera, y realice las tareas que se definieron en las fases anteriores, como por ejemplo, el diseño. Para este proyecto se decidió usar pruebas unitarias (o ***unittesting***) para asegurar que cada función se comporte como se espera, y que cada parámetro de salida y entrada sea acorde con lo que se establece en el manual de referencia. Una unidad es la parte ***testeable*** más pequeña de una aplicación, como métodos o funciones. Si bien esta técnica es muy usada para lenguajes orientados a objetos, el concepto es aplicable a cualquier paradigma, y, por ende, a la programación estructurada.

Al decidir utilizar esta técnica por sobre otras, se tomó en cuenta la naturaleza de cambio propia de una librería, y la complejidad de la tecnología con la que se trabaja. Al construir una prueba unitaria, la funciona probar debe seguir un comportamiento estrictamente definido para pasarla. Estas pruebas facilitan los cambios en el software, pues al crear una prueba, esta se puede reutilizar tantas veces como sea necesario, y se puede probar si los cambios hechos a una función la satisfacen. Otro beneficio al usar pruebas unitarias es que, al diseñar los casos de pruebas, se puede trabajar en conjunto con el manual de referencia de la API, asegurando así que concuerden.

Existen muchos *frameworks* para pruebas unitarias en distintos lenguajes, y se optó por utilizar J2ME UnitTest [5].

Con la ayuda de de este framework se establecieron conjuntos de pruebas (denominadas *test case*) agrupadas en un solo proyecto. Las test case se separaron según sus funcionalidades en distintos paquetes, los cuales son:

- AbrirCerrarConeccion: Contiene las pruebas necesarias para poder realizar conexiones en ambientes L2CAP y RFCOMM, también para las desconexiones.
- Buscar Dispositivos: Contiene las pruebas necesarias para poder buscar y acceder a dispositivos móviles.
- Buscar Servicios: Contiene las pruebas necesarias para poder buscar servicios dentro de un dispositivo móvil previamente encontrado.
- EnviarRecibirMensajes: Contiene las pruebas de las funciones que sirven para la recepción y emisión de todos los tipos de mensajes creados en el módulo de comunicación y sin importar el protocolo.

8.3.1. Resultados

Los resultados obtenidos fueron positivos en escenarios donde la distancia entre dispositivos móviles fuese entre el rango de 0 a 8 metros como máximo, después de eso las pruebas tienden a fallar. Los casos de prueba están agrupados por los paquetes definidos anteriormente, los cuales fueron:

- AbrirCerrarConexion:
    - Precondición para abrir conexiones:
        - Dispositivo móvil con el programa cargado e intacto.
        - Test AbrirConexionL2CAP: Pass.
        - Test AbrirConexionRFCOM: Pass.

Sobre Wireless Toolkit:

Estableciendo una prueba de estrés con 100 iteraciones sobre AbrirConexionL2cap, se logra ver que no ocurre ningún solo percance, y que ni siquiera la memoria se ve afectada manteniéndose en 92 kilobytes app, véase figura 21.

Sobre Nokia 5200:

Solamente se pudieron realizar 10- 35 pruebas consecutivas, luego el celular se "reinicia", sin arrojar ningún indicio por pantalla de lo que sucedió. Generalmente este celular se reinicia cuando encuentra una anomalía o un servicio de este entra en conflicto, por lo que se presume que al activar y desactivar continuamente bluetooth puede ocasionar en un cierto momento el cuelgue de la prueba. Se cataloga como aceptable el resultado dado que el inicio de conexiones no se hará de forma concurrente.

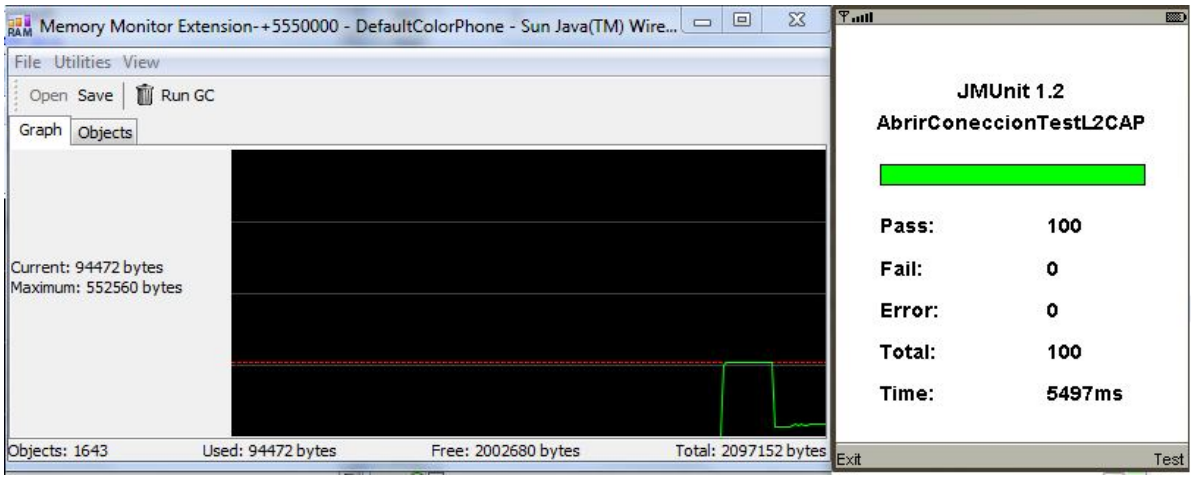

Figura 21 - Simulación de caso: abrir conexión con L2CAP

- Precondición cerrar conexiones:
  - Tener una conexión abierta.
  - Test CerrarConexionL2CAP: Pass.
  - Test CerrarConexionRFCOM: Pass.

Sobre Wireless Toolkit:

Estableciendo una prueba de estrés con 100 iteraciones sobre CerrarConexionRFCOM, en promedio se obtienen 4 – 5 errores cada cien pruebas, nuevamente la memoria no es factor en la prueba, véase figura 22.

Sobre Nokia 5200:

No es posible obtener ningún resultado positivo, a través de una investigación se logra deducir ; cuando se inicia el servidor este inicia un thread encargado de gestionar las conexiones entrantes y como en la prueba se cierra muy de golpe sin esperar que ni siquiera el thread que gestiona las conexiones entrantes se inicie correctamente, ocasiona una excepción.

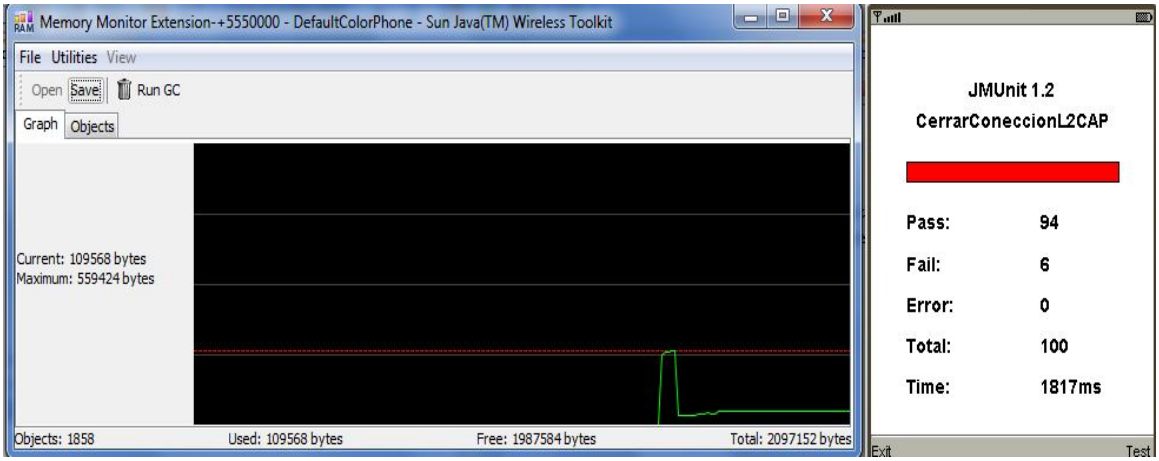

Figura 22 - Simulación de caso: cerrar conexión con L2CAP

- Buscar Dispositivos:
    - Precondiciones:
        - Tener 2 dispositivos con una conexión abierta.
        - Test BuscarDispositivosL2CAP: Pass.
        - Test BuscarDispositivosRFCOM: Pass.

Sobre Wireless Toolkit:

Estableciendo una prueba de estrés con 100 iteraciones sobre Buscar Dispositivos L2CAP, no se logran conseguir errores, se confirmo que las pruebas sobre Wireless Toolkit los resultados obtenidos no son realistas aunque incluso se asignaran parámetros parecidos a las características de un dispositivo móvil estos seguirán arrojando resultados positivos, véase figura 23.

Sobre Nokia 5200:

La búsqueda de dispositivos es un proceso lento en bluetooth, demora en promedio 25 – 40 segundos variando mucho en distintos dispositivos que se encuentran en el mercado. Las pruebas fueron satisfactorias, pero engorrosas dado que la demora de la búsqueda retrasaba el flujo normal de las pruebas, por lo que se validaron las pruebas de búsquedas de dispositivos sobre 10 casos. Se obtiene que la distancia promedio en que funciona es de 7-8 metros y no 10 metros como se dice en la norma.

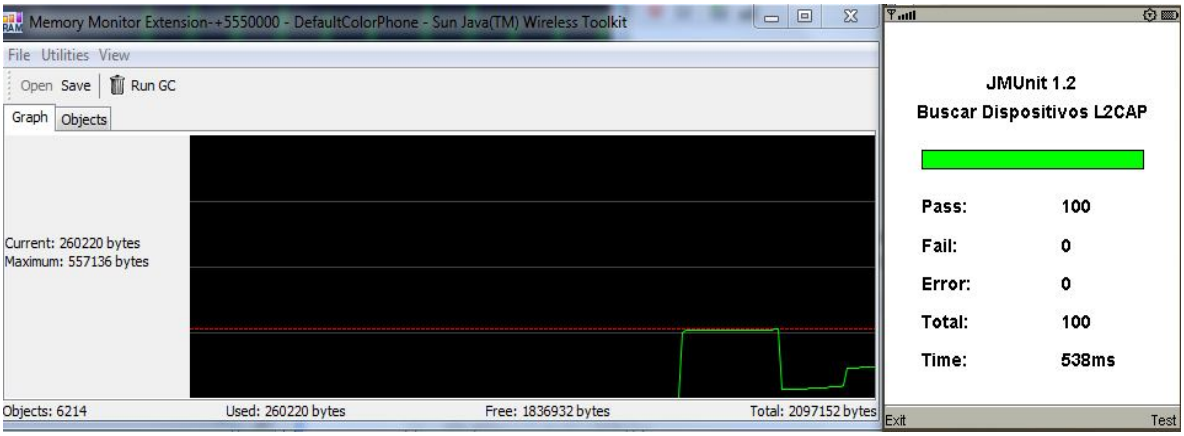

Figura 23 - Simulación de caso: buscar dispositivo L2CAP

- BuscarServicios:
    - Precondiciones
        - Tener 2 dispositivos con una conexión abierta.
        - Un dispositivo debe haber encontrado a otro par.
        - El dispositivo servidor debe tener algún tipo de servicio para ofrecer.
        - Test BuscarServiciosL2CAP: Pass.
        - Test BuscarServiciosRFCOMM: Pass.

Sobre Wireless Toolkit :

Estableciendo una prueba de estrés con 100 iteraciones sobre Buscar Servicios con RFCOMM, como se menciono antes las búsquedas de dispositivos son lentas sobre bluetooth, tanto así las búsquedas de dispositivos como de servicios sobre estos, en la misma prueba se logra ver este retraso demorando casi 11 minutos en terminar las 100 iteraciones. Otro punto importante a destacar de esta prueba es que es la primera que desborda la memoria hasta 1.5 megabyte, esto se debe principalmente que se abren canales de comunicación, se traen dispositivos móviles e incluso algo de envió de mensajes., véase figura 24.

Sobre Nokia 5200:

Como en los casos anteriores la batería de pruebas fue reducida a 10 iteraciones dadas las limitancias ya evidenciadas por el celular, sin embargo logra realizar las pruebas satisfactoriamente dejando entrever un dato importante, que mientras más participantes tenga la sesión de juego nos es más difícil conectarnos (tiempos de espera).

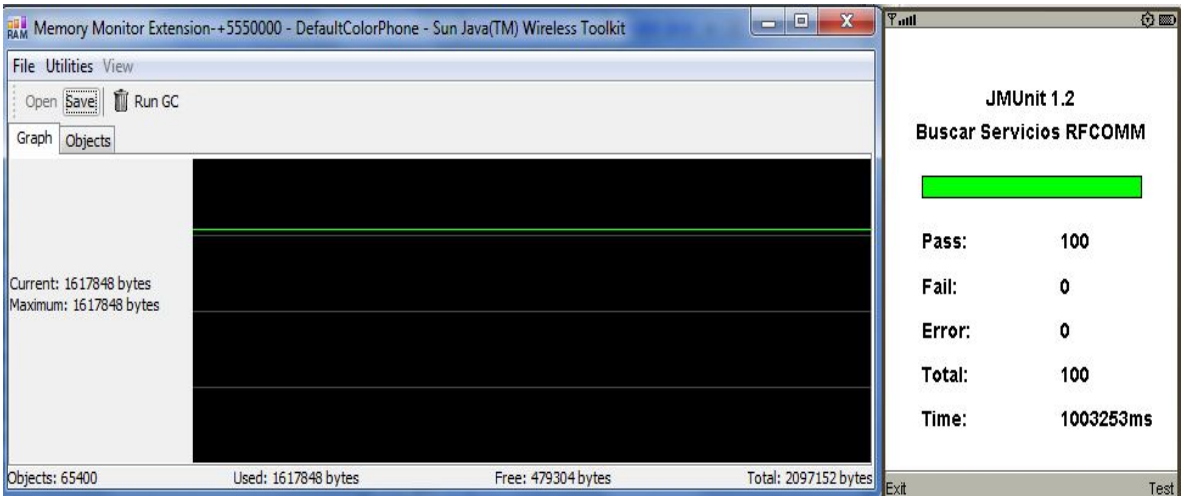

Figura 24 - Simulación de caso: buscar servicios RFCOMM

- EnviarRecibirMensajes:
  - Precondiciones:
    - Tener dos dispositivos con una conexión abierta.
    - Un dispositivo debe haber encontrado a otro par.
    - El dispositivo servidor debe tener algún tipo de servicio para ofrecer.
    - El dispositivo cliente debe haber aceptado e ingresado al servicio ofrecido.
    - Test EnviarRecibirClienteMensajesL2CAP: Pass.
    - Test EnviarRecibirClienteMensajesRFCOM: Pass.
    - Test EnviarRecibirServidorMensajesL2CAP: Pass.
    - Test EnviarRecibirServidorMensajesRFCOM: Pass.
    - Métodos bloqueantes:
    - Test EnviarRecibirClienteBloqueanteMensajesL2CAP: Pass.
    - Test EnviarRecibirClienteMensajesBloqueanteRFCOM: Pass.
    - Test EnviarRecibirServidorMensajesBloqueanteL2CAP: Pass.
    - Test EnviarRecibirServidorMensajesBloqueanteRFCOM: Pass.

Sobre Wireless Toolkit :

Estableciendo una prueba de estrés con 100 iteraciones sobre Enviar recibir cliente L2CAP, solamente las primeras 4 iteraciones se ejecutaron correctamente, se intento repetir, pero en 5 ocasiones se estancaba entre 3-4 casos de éxito, esto se debe porque en ARQlet se tiene configurado por defecto que el máximo de clientes posibles en una sesión sean de hasta 4, ya que el overead generado no permite que la red se puede jugar de una forma correcta, véase figura 25.

Sobre Nokia 5200:

Nuevamente la búsqueda de servicios y Dispositivos retrasan el objetivo final de la prueba, pasando estos *baches*, el efecto anterior ocurrido en Wireless Toolkit se repite, se puede llegar hasta solamente 4 conexiones dada las restricciones de ARQlet sobre sus sesiones. Cabe mencionarse que el envió y recepción de mensajes es bastante mejor que los procesos de búsqueda que presenta bluetooth.

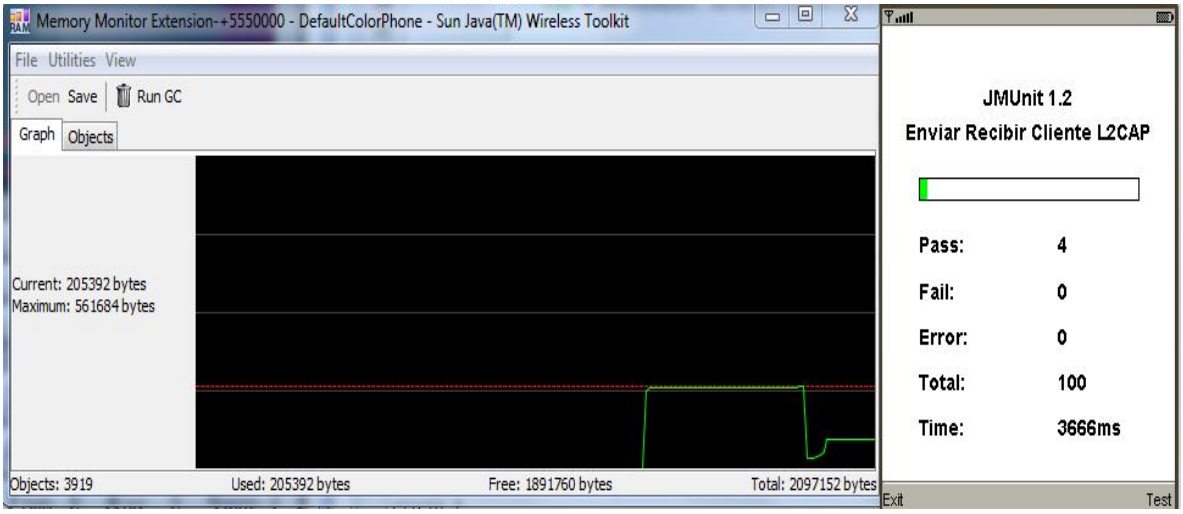

Figura 25 - Simulación de caso: enviar-recibir cliente L2CAP

8.4. Problemas encontrados

Los principales problemas detectados en las pruebas fueron los siguientes:

- El radio efectivo de 10 metros, bluetooth: para una comunicación efectiva, el verdadero rango óptimo es entre los 0 y 7 metros, sobre 9 metros se pierde velocidad en la transmisión de datos, con posibilidades de perder la conexión.

- Arquitectura cliente servidor: dado que para una sesion de juego donde hayan más de dos jugadores, se hace pesar el nivel de sobrecarga de mensajes sobre la entidad maestra de la red, incidiendo mucho en el rendimiento de la red y la comunicación.

- Mensajes bloqueantes no garantizan el orden: quedó demostrado con las pruebas que al formar redes grandes, los mensajes bloqueantes no garantizan el orden de los eventos, pudiendo ocasionar la perdida del orden de los eventos en los dispositivos.

- El tratamiento lento de los threads: mucho de los casos de error se debian a que los threads sobre los dispositivos celulares no se iniciaban bien, entonces al momento de el programa principal liberar recursos estos quedaban casi como demonios o no totalmente inicializados, causando muchas de las fallas en las pruebas.

9. Conclusiones

El contar con una herramienta de fácil integración de tecnologías de comunicación con distintos protocolos en aplicaciones de dispositivos móviles presenta sin duda una gran ventaja para los desarrolladores de videojuegos colaborativos.

Se ha presentado un modelo de arquitectura modular para juegos enfocado a dispositivos móviles, y que a su vez permitan emular partidas de juego multiplayer implementando un modulo de comunicación bajo el protocolo inalámbrico bluetooth.

Este modelo de arquitectura ha pretendido plasmar las necesidades esenciales en el desarrollo de videojuegos en plataformas móviles, y cada uno de los módulos, o entidades de la arquitectura han sido diseñadas con el fin de brindar soporte a los desarrolladores.

La presente investigación ha contemplado la implementación del módulo de Comunicación y de Procesamiento, agregando un modelo de diseño para los módulos de Lógica, Persistencia y Seguridad en dispositivos móviles.

Existiendo tantos protocolos y tecnologías de comunicación, se ha establecido para esta arquitectura la red inalámbrica personal bluetooth, por el hecho de ser una red con canales abiertos, gratuita y prácticamente existente en cada uno de los nuevos dispositivos que el mercado ofrece, también ha influido la compatibilidad para el desarrollo en la tecnología java, haciendo de esta arquitectura un software muy portable y estándar para los dispositivos que la implementen.

Una serie de diagramas estáticos y dinámicos ilustran el diseño e implementación del modulo de comunicación, el cual ha sido desarrollado con implementación para dos de sus protocolos: L2CAP y RFCOMM.

Tras la implementación de estos protocolos pueden establecerse claras comparativas en cuanto a prestaciones y rendimiento, las cuales pueden ser medidas en tiempos de respuesta en la red, velocidad en la transmisión de los datos, y sincronismo en las partidas de juego.

En conclusión luego del desarrollo de prototipos desechables y la implementación de los protocolos de bluetooth los resultados dejan a la vista la ventaja que tiene el protocolo L2CAP por sobre RFCOMM, por el hecho de los mecanismos de transmisión entre los dispositivos, mientras que el protocolo RFCOMM transmite los datos en serie, el protocolo L2CAP serializa la información permitiendo un envío más rápido(a nivel de bytes), y por ende mejores tiempos de respuesta y sincronismo en las partidas de juego.

Sin duda alguna una de las complejidades en una red cliente-servidor como lo es bluetooth es la sincronización de los mensajes entre dispositivos, la concurrencia entre los dispositivos esclavos es un punto a tratar a fondo. Soluciones aptas pueden ser algoritmos de sincronismo avanzados, como lo son los relojes vectoriales o monitores.

Además puede decirse que claramente la principal problemática obtenida al momento de desarrollar en distintos plataformas de dispositivos móviles ha sido la extensibilidad de la arquitectura, la cual permita el desarrollo de juegos de tipo homogéneos, que puedan ser implementados de forma similar en distintos dispositivos móviles permitiendo el uso de estos en plataformas multijugador sin importar en la gama de dispositivo a la que estos pertenezcan.

Debido a la gran diversidad de marcas y modelos, la programación en dispositivos móviles es ciertamente compleja, ya que mucha de la codificación de los proyectos no es estándar para todas las gamas de dispositivos existentes.

10. Referencias

[1] Ana Mas - Agentes Software y Sistemas Multiagente, Pearson Prentice Hall.

[2] Adrew S. Tanenbaum - Redes de Computadoras. Prentice-Hall.

[3] Andrés Castaño, Juan Hincapié - Framework de comunicaciones y contexto para la Construcción de juegos multiusuario. Universidad EAFIT 2007.

[4] Brendon J. Wilson - JXTA. First Edition, New Riders, June 2002.

[5] Brunno Silva, Carl Meijer - JMUnit 1.2 Java Micro Unit, August 2008.

[6] Cay S. Horstmann, Gary Cornell - Core Java 2. Volume II.

[7] Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Professional Computing Series.1995.

[8] Jamie McHale. Developing a bluetooth-enabled J2ME game, University of Glasgow, September 2007.

[9] Li, S., Knudsen, J. Beginning J2ME: From Novice to Professional, Third Edition, Apress Editorial, 2005.

[10] Mobile Information Device Profile (MIDP), JSR 37, JSR-118 Overview, 2005.

[11] Luis Llana D. Programación Concurrente en Java. Departamento de Sistemas Informáticos y Programación. Universidad Complutennse de Madrid, March 2006

[12] Mark Weiser. The computer for the 21st century. Scientific American, 265, 3, 94-104.

[13] Roberto Cacho B. http://beta.redes-linux.com/manuales/bluetooth/8-Bluetooth.pdf.gz

[14] Roger S. Pressman - Ingeniería de Software, un enfoque práctico", 5ta Edición Editorial McGraw-Hill.

[15] Sun - Mobile Information Device Profile (MIDP), JSR 37, JSR 118 Overview. 2005.

[16] Williams Stallings Data and Computer Communications, 5th Edition

[17] http://www.jsr-82.com - Especificación del estándar Bluetooth sobre java.