Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ingeniería en Ciencias y Sistemas

PROXIMITY MARKETING: ESTUDIO DE UNA NUEVA FORMA DE HACER PUBLICIDAD BASADA EN EL USO DE DISPOSITIVOS DE BLUETOOTH Y CONSTRUCCIÓN DE UN PROTOTIPO FUNCIONAL

Ronmell Ignacio Fuentes Orozco

Asesorado por el Ing. Cristian Eduardo Lavarreda Estrada

Guatemala, noviembre de 2010

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

PROXIMITY MARKETING: ESTUDIO DE UNA NUEVA FORMA DE HACER PUBLICIDAD BASADA EN EL USO DE DISPOSITIVOS DE BLUETOOTH Y CONSTRUCCIÓN DE UN PROTOTIPO FUNCIONAL

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA

FACULTAD DE INGENIERÍA

POR

RONMELL IGNACIO FUENTES OROZCO

ASESORADO POR EL ING. CRISTIAN EDUARDO LAVARREDA ESTRADA

AL CONFERÍRSELE EL TITULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, NOVIEMBRE DE 2010

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Murphy Olympo Paiz Recinos
VOCAL I	Inga. Glenda Patricia García Soria
VOCAL II	Inga. Alba Maritza Guerrero Spínola de López
VOCAL III	Ing. Miguel Ángel Dávila Calderón
VOCAL IV	Br. Luis Pedro Ortíz de León
VOCAL V	P.A. José Alfredo Ortíz Herincx
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Murphy Olympo Paiz Recinos
EXAMINADOR	Ing. Pedro Pablo Hernández Ramírez
EXAMINADOR	Ing. Edgar Estuardo Santos Sutuj
EXAMINADOR	Ing. César Augusto Fernández Cáceres
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

Cumpliendo con los preceptos que establece la Ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de Graduación titulado:

PROXIMITY MARKETING: ESTUDIO DE UNA NUEVA FORMA DE HACER PUBLICIDAD BASADA EN EL USO DE DISPOSITIVOS DE BLUETOOTH Y CONSTRUCCIÓN DE UN PROTOTIPO FUNCIONAL,

tema que me fuese asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, en octubre 2009.

Ronmell Ignacio Fuentes Orozco

Guatemala, 15 de Agosto de 2010.

Ingeniero.

Carlos Alfredo Azurdia Morales.

Coordinador de Privados y

Revisión de Trabajos de Graduación.

Presente.

Estimado Ingeniero Azurdia:

Por este medio me permito informarle que he procedido a revisar el trabajo de graduación titulado **PROXIMITY MARKETING: ESTUDIO DE UNA NUEVA FORMA DE HACER PUBLICIDAD BASADA EN EL USO DE DISPOSITIVOS DE BLUETOOTH Y CONSTRUCCIÓN DE UN PROTOTIPO FUNCIONAL,** elaborado por el estudiante Ronmell Ignacio Fuentes Orozco, carnet 200516010, y que a mi juicio, el mismo cumple con los objetivos propuestos para su desarrollo.

Sin otro particular, me despido de usted,

Atentamente,

Ing. Cristian Eduardo Lavarreda Estrada.

Asesor.

Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 06 de Octubre de 2010

Ingeniero
Marlon Antonio Pérez Turk
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Pérez:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **RONMELL IGNACIO FUENTES OROZCO** carné **2005-16010,** titulado: **"PROXIMITY MARKETING: ESTUDIO DE UNA NUEVA FORMA DE HACER PUBLICIDAD BASADA EN EL USO DE DISPOSITIVOS DE BLUETOOTH Y CONSTRUCCION DE UN PROTOTIPO FUNCIONAL",** y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
INGENIERIA EN CIENCIAS Y SISTEMAS
COORDINACION - TESIS -

ESCUELA DE CIENCIAS Y SISTEMAS

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÌA
ESCUELA DE CIENCIAS Y SISTEMAS
TEL: 24767644

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del asesor con el visto bueno del revisor y del Licenciado en Letras, de trabajo de graduación titulado ***"PROXIMITY MARKETING: ESTUDIO DE UNA NUEVA FORMA DE HACER PUBLICIDAD BASADA EN EL USO DE DISPOSITIVOS DE BLUETOOTH Y CONSTRUCCIÓN DE UN PROTOTIPO FUNCIONAL"****, presentado por el estudiante RONMELL IGNACIO FUENTES OROZCO, aprueba el presente trabajo y solicita la autorización del mismo.*

"ID Y ENSEÑAD A TODOS"

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION DE INGENIERIA EN CIENCIAS Y SISTEMAS

Ing. Marlon Antonio Pèrez Turk
Director, Escuela de Ingenierìa Ciencias y Sistemas

Guatemala, 22 de noviembre 2010

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

DTG. 396.2010

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al trabajo de graduación titulado: **PROXIMITY MARKETING: ESTUDIO DE UNA NUEVA FORMA DE HACER PUBLICIDAD BASADA EN EL USO DE DISPOSITIVOS DE BLUETOOTH Y CONSTRUCCIÓN DE UN PROTOTIPO FUNCIONAL,** presentado por el estudiante universitario **Ronmell Ignacio Fuentes Orozco,** autoriza la impresión del mismo.

IMPRÍMASE:

Ing. Murphy Olympo Paiz Recinos
Decano

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

Guatemala, 23 de noviembre de 2010.

/gdech

ACTO QUE DEDICO A:

A DIOS	Por ser mi vida.
A MIS PADRES	Romeo Fuentes y Lesbia Orozco, por ser padres ejemplares brindándome su apoyo incondicional en todos los aspectos de mi vida en todo momento. “Abuelos”, ustedes, son lo mejor de mi vida.
A MI(S) HERMANO(AS)	A cada uno en particular por toda la ayuda y soporte brindado en cualquier aspecto de mi vida. Siempre serán un ejemplo para mí.
A MIS ABUELITOS	q.e.p.d.
A MI SOBRINO(A)	Gracias por las alegrías brindadas.
A MIS CUÑADOS	Sinceros y leales agradecimientos.
A MIS AMIGOS	El honor, de tener su amistad, es todo mío. Gracias por todos los momentos compartidos, sin duda están escritos sobre piedra y jamás serán olvidados.
A MI ASESOR DE TESIS	Es un honor haber estado bajo su instrucción. Gracias por el apoyo y la confianza puesta en mí.
A MI GUATEMALA	Orgulloso de haber nacido en este suelo. ¡Arriba Guate!

A LA TRICENTENARIA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA, PRINCIPALMENTE A LA FACULTAD DE INGENIERIA Y ESPECIALMENTE A LA ESCUELA DE CIENCIAS Y SISTEMAS.

ÍNDICE

ÍNDICE DE ILUSTRACIONES VII
GLOSARIO IX
RESUMEN XI
OBJETIVOS XV
INTRODUCCIÓN XVII
1. MARCO TEÓRICO 1
1.1. Definición de conceptos de Proximity Marketing 1
1.1.1. Marketing 1
1.1.2. Proximity Marketing 2
1.1.3. Publicidad 4
1.1.4. Formas de hacer publicidad 5
1.1.4.1. Tv/Radio 5
1.1.4.2. Diario y medios escritos 5
1.1.4.3. Internet y medios interactivos 6
1.1.5. Interacción cliente-publicidad 6
1.1.6. Interacción publicidad-publicista 7
1.1.7. Hacia un nuevo marketing 8
1.1.8. Proximity Marketing: Características únicas y ventajas 9
1.1.8.1. Características únicas de Proximity Marketing 9
1.1.8.1.1. Proporciona datos reales del comportamiento humano 9
1.1.8.1.2. Permite analizar los datos obtenidos 9
1.1.8.1.3. Permite la toma de decisiones con base en el análisis de los datos obtenidos 10
1.1.8.2. Ventajas 11
1.1.8.2.1. Se basa en el permiso del cliente 11
1.2. Definición técnica de los dispositivos y conceptos usados en la solución 11

1.2.1. Software 11
1.2.1.1. Servidor de contenidos 11
1.2.1.2. Servidor controlador de enrutadores 12
1.2.1.3. Red de servidores 13
1.2.2. Hardware 14
1.2.2.1. Enrutador de antenas bluetooth 14
1.2.2.2. Antenas bluetooth 14
1.2.2.3. Dispositivos bluetooth para móviles 15
1.3. Definición técnica de la comunicación entre dispositivos de bluetooth y sus conceptos más importantes 15
1.3.1. Comunicación entre dispositivos electrónicos de tecnología Bluetooth 16
1.3.2. Roles de los dispositivos en la comunicación 17
1.3.2.1. Cliente 17
1.3.2.2. Servidor 18
1.3.3. Protocolos 18
1.3.3.1. Obex 19
1.3.3.1.1. Componentes Obex. 21
1.3.3.1.1.1. Obex Session Protocol. 21
1.3.3.1.1.2. Obex Application Framework. 22
1.3.3.2. L2Cap 24
1.3.3.3. Sockets Rfcomm 25
1.3.3.4. IrObex 26
1.3.3.4.1. Tipos de paquetes de IrObex 27
1.3.3.4.2. Características importantes de IrObex 27
1.3.3.4.3. Ejemplos de paquetes IrObex 28
1.3.3.4.3.1. Paquetes: conexión y desconexión 28
1.3.3.4.3.1.1. Paquete de conexión 28
1.3.3.4.3.1.2. Paquete de desconexión 29

1.3.3.4.3.2. Paquetes: put y get 30
1.3.3.4.3.2.1. Paquete put 31
1.3.3.4.3.2.2. Paquete get 32
1.3.4. Servicios 33
1.3.4.1. OPP 34
1.3.5. Controladores 36
1.3.5.1. Blue Z o bluez 36
1.4. Definición técnica de la solución 37
1.4.1. Requisitos del sistema 37
1.4.1.1. Software 37
1.4.1.1.1. Ambiente integrado de desarrollo: IDE 37
1.4.1.1.1.1. ¿Por qué Eclipse Ganimedes? 37
1.4.1.1.2. Lenguaje 38
1.4.1.1.2.1. ¿Por qué C/C++? 38
1.4.1.1.3. Sistema operativo 38
1.4.1.1.3.1. ¿Por qué Linux? 38
1.4.1.2. Hardware 39
1.4.1.2.1. Antenas bluetooth 39
1.4.1.2.2. Enrutador de antenas bluetooth (opcional) 39
1.4.2. Definición de los subsistemas 40
1.4.2.1. Proceso del sistema 40
1.4.2.2. Arquitectura de la solución 42
1.4.2.3. Separación del sistema en módulos 43
1.4.2.3.1. Scanner 44
1.4.2.3.1.1. Simple Scanner 44
1.4.2.3.1.2. Device 46
1.4.2.3.2. Sender 47
1.4.2.3.2.1. Negotiator 47
1.4.2.3.2.2. Simple Sender 48

1.4.2.3.2.3. Packet 48
1.4.2.3.3. Dbmanager 49
1.4.2.3.3.1. Dbaction 49
1.4.2.4. Justificación de los módulos 50
1.4.2.4.1. Scanner 50
1.4.2.4.1.1. ¿Por qué Simple Scanner? 51
1.4.2.4.1.2. ¿Por qué Device? 51
1.4.2.4.2. Sender 51
1.4.2.4.2.1. ¿Por qué Negotiator? 52
1.4.2.4.2.2. ¿Por qué Sender? 52
1.4.2.4.2.3. ¿Por qué Packet? 52
1.4.2.4.3. Dbmanager 52
1.4.2.4.3.1. ¿Por qué Dbaction? 52
1.4.2.4.4. Interface entre los módulos 53
1.4.2.4.4.1. Interface: Scanner-Sender 53
2. MARCO PRÁCTICO 55
2.2. Análisis, diseño y desarrollo de la solución 55
2.2.1. Diseño de la arquitectura 55
2.2.1.1. Arquitectura 56
2.2.1.2. Diagrama de clases 57
2.2.2. Diseño de componentes 57
2.2.2.1. Scanner 57
2.2.2.1.1. Simple Scanner 57
2.2.2.1.1.1. Funciones 58
2.2.2.1.1.1.1. GetNerbyDevices 58
2.2.2.1.1.1.2. HasPushProfile 65
2.2.2.1.1.1.3. InsertDbDevice 70
2.2.2.1.2. Device 71
2.2.2.1.2.1. Funciones 71

2.2.2.1.2.1.1. AllocateDevice 71
2.2.2.2. Dbmanager 73
2.2.2.2.1. Dbaction 73
2.2.2.2.1.1. Funciones 73
2.2.2.2.1.1.1. InsertIntoDbDevice 74
2.2.2.2.1.1.2. InsertIntoDbLog 77
2.2.2.3. Sender 78
2.2.2.3.1. Sender 78
2.2.2.3.1.1. Funciones 78
2.2.2.3.1.1.1. SendData 78
2.2.2.3.1.1.2. GetOnePacket 85
2.2.2.3.2. Negotiator 90
2.2.2.3.2.1. Funciones 91
2.2.2.3.2.1.1. MakePacket 92
2.2.2.3.3. Packet 94
2.2.2.3.3.1. Funciones 94
2.2.2.3.3.1.1. Funciones de mutación 95
2.2.2.3.3.1.1.1. SetPacketLength 95
2.2.2.3.3.1.1.2. SetObjectName 96
2.2.2.3.3.1.1.3. SetObjectLength 97
2.2.2.3.3.1.1.4. SetBody 98
2.2.2.3.3.1.2. Funciones de acceso 99
2.2.2.3.3.1.2.1. GetPacketLength 99
2.2.2.3.3.1.2.2. GetObjectName 99
2.2.2.3.3.1.2.3. GetObjectLength 100
2.2.2.3.3.1.2.4. GetBody 100
3. IMPLEMENTACIÓN 103
3.1. 10 pasos para una implementación exitosa de Proximity Marketing 103

CONCLUSIONES 109
RECOMENDACIONES 111
BIBLIOGRAFÍA 115
APÉNDICES 117

ÍNDICE DE ILUSTRACIONES

FIGURAS

1.	Modelo de capas L2Cap	25
2.	Formato de un paquete connect-request	27
3.	Formato de un paquete connect-response	29
4.	Ejemplo de paquetes del tipo connect	29
5.	Formato de un paquete disconnect-resquest	30
6.	Formato de un paquete disconnect-response	30
7.	Formato de un paquete put-request	31
8.	Formato de un paquete put-response	31
9.	Ejemplo de paquetes del tipo put	32
10.	Formato de un paquete get-request	33
11.	Formato de un paquete get-response	33
12.	Arquitectura de la solución	56
13.	Diagrama de clases de la solución	57

TABLAS

I	Analogía protocolo DHCP - IrObex	20
II	Componentes del Obex Application Framework	22
III	Valores promedio de η	107

GLOSARIO

Bilateral Adj. Relativo a ambas partes, contrato en que intervienen dos partes.

BlueZ Controlador de las antenas de bluetooth para sistemas operativos Linux.

booleano (a) Tipo de dato que permite dos valores: verdadero o falso.

Bucle Ciclo de iteraciones que permiten ejecutar sentencias de programación.

BYTE Medida de almacenamiento de datos, un byte es igual a 8 bits.

Casteado (a) Operación de castear o forzar a la conversión de un tipo de dato a otro.

char Tipo de dato, equivalente a character o su traducción: caracter.

Consola Terminal de un sistema operativo.

Customizing Del verbo en Inglés: Customize, ajustar un producto o servicio a las necesidades de cierto cliente

Eficaz Que logra hacer efectivo un intento o propósito.

Eficiencia Capacidad de un fin empleando los mejores medios posibles.

Flags Banderas o parámetros de configuración que son enviados a una función.

Función Segmento de código, de computadora, que ejecuta ciertas órdenes.

HAL Son las siglas para Hardware Abstraction Layer que significa Capa de Abstraccion de Hardware, usada por los

	Sistemas Operativos Linux.
HI	Header Id o identificador de cabecera.
Imprescindible	Que no se puede prescindir de ello, que no se puede evitar.
IrObex	Modificación de Obex para ser usado por bluetooth.
Linux	Sistema operativo de código abierto.
NULL	Valor nulo de una variable.
Obex	Protocolo de intercomunicación de dispositivos de luz infrarroja.
OpCode	Código de Operación.
OPP	Object Push Profile que traducido es: Perfil de Colocación de Objetos.
Query	Código en lenguaje SQL que realiza instrucciones en una base de datos.
Sensitiva(o)	Adecuado a los sentidos del ser humano.
WEB 2.0	Termino acuñado por Tim O' Reilly para definir una nueva generación de contenidos basados sobre escenarios web o World Wide Web.
Wi-Fi	Sistema de envío de información que utiliza ondas de radio en lugar de cables.

RESUMEN

Esta tesis presenta un estudio sobre un tema que, aunque en Guatemala no es muy conocido, internacionalmente está siendo muy desarrollado; esto es Proximity Marketing o mercadeo a proximidades. En este trabajo se expone, desde los conceptos básicos hasta la creación conceptual de una solución de software, así como su implementación. Luego, mediante pruebas, se obtienen resultados que puedan ser analizados y luego proponer mejoras a la solución.

En principio se describen conceptos relacionados al marketing, la publicidad, las formas de hacer publicidad, interacción entre el cliente y el publicador, y se hace ver que existen varias formas de hacer que el cliente reciba correctamente el mensaje que se le está enviando a través de la publicidad.
En primera instancia, se proporcionan algunas referencias a personajes que han escrito materiales de reconocimiento alto y aserción profunda respecto de sus opiniones. Luego, se presenta la forma en la que se deben elegir las herramientas a usar y las tecnologías que se implementarán.
Además, se presenta la forma en la que se debe desarrollar una solución basada en los componentes de software y hardware que se necesitan para desarrollar este concepto (Proximity Marketing).

La idea básica sobre la que se fundamenta este trabajo es:
"*Proveer un estudio del proceso que conlleva la creación de una solución de software, orientada a Proximity Marketing, desde su concepción, construcción y hasta en su implementación"*.

Esta solución está orientada hacia la generación de una nueva idea que persigue incrementar el rango de opciones del mercado para publicitar los contenidos.

Se parte de la necesidad de crear una nueva forma para que las personas, anunciantes de productos y servicios, puedan publicar sus ofertas, precios, entre otros. Luego se muestra la forma en la que se hace la publicidad en la actualidad. Después, se plantea la necesidad de hacer que la nueva forma de hacer publicidad sea efectiva. Para esto es necesario definir el uso de la tecnología que se posee con nuevas objetivos basados en los mercados de proximidades. Luego se relacionan los aspectos más importantes, de Proximity Marketing, como la conectividad, efectividad, personalización, entre otros y se forma una idea de cómo debería ser una nueva forma para que los clientes reciban contenidos publicitarios.

Seguidamente, se analiza la funcionalidad que un dispositivo móvil ofrece, relacionado a la solución que se desea implementar. Se estudian los elementos de una buena comunicación, las relaciones entre los involucrados –clientes, publicitarios, publicidad, forma de publicitar- y lo que la nueva solución debe poseer para ser en realidad una solución efectiva. Luego se plantea la solución, siempre detallando las razones por las cuales se están usando ciertos elementos, dispositivos, componentes.

Luego se forma la idea general de la solución y después, usando la perspectiva sistémica, ésta se descompone, en más subsistemas o componentes que permitirán, con su interacción, completar el objetivo del sistema macro. Luego, se estudia cada componente y se analiza por separado, así también la descripción general de lo que cada sistema deberá hacer, con sus referencias, a fin de proveer una mejor comprensión por parte del usuario.

Finalmente, se estudia la forma en la que un sistema de Proximity Marketing deberá ser implementado en un escenario real. Luego se estudian las condiciones que deben ser parte del ambiente en el cual se está

implementando. Además se presentan algunas recomendaciones para la implementación de un sistema de este tipo.

Estas recomendaciones se hacen con el fin de mostrar al usuario que todo sistema puede y debe ser mejorado con cada iteración que se le haga al mismo, esto basado en la premisa de la integración continua de los sistemas de tecnologías de la información (IT) y lograr, de esta forma, que los sistemas puedan ajustarse, cada vez más, a los requisitos que el cliente desea. Ya que éstos deben ser cumplidos por cualquier solución de software que sea construida.

OBJETIVOS

GENERAL

Realizar un estudio acerca de una nueva forma de hacer publicidad, usando dispositivos de Bluetooth para teléfonos móviles celulares; proporcionando una solución mediante software, conocido como Proximity Marketing.

ESPECIFICOS

- Inferir el impacto del uso de la tecnología de Bluetooth, en la publicidad de un producto o servicio, basado en el impacto que otras tecnologías han tenido sobre ésta.

- Definir el marco teórico de la solución de Proximity Marketing.

- Proponer un prototipo funcional de una solución de software para contribuir a la realización de Proximity Marketing.

- Recomendar las mejores prácticas en la forma de hacer publicidad, utilizando Proximity Marketing.

INTRODUCCIÓN

Publicidad, un concepto desarrollado formalmente, desde los inicios, en la concepción de los productos y comercios en la antigüedad, cuando las personas necesitaban anunciar sus productos o servicios basados en sus habilidades. Pero quizás la publicidad nace desde que la comunicación fuese concebida como el medio de interrelación para las sociedades. Según lo menciona Wikipedia, existe en una tablilla de arcilla, con orígenes Babilónicos, en la que se tiene contenida inscripciones de un comerciante de ungüentos, un escribano y un zapatero, ésta data del 3,000 a. C [1].Se puede ver que la publicidad, entonces, se remonta hasta épocas antiguas muy lejanas a la nuestra, sin embargo el concepto de publicidad ha sufrido muchos cambios.

Los cambios a la publicidad están basados en que los publicadores desean que la misma tenga más efecto positivo en los clientes potenciales; se dice que toda persona es un cliente, y es importante que, eventualmente, la persona pueda ejercer el poder de compra del bien o servicio que se está anunciando.

Actualmente, como lo menciona la Profesora Silvia Sivera –Universidad Oberta de Catalunya- estamos expuestos a más de 2,000 mensajes publicitarios al día; prestamos algún tipo de atención a 52; leemos, vemos o escuchamos con cierta disposición unos 24; nos gustan 10; y recordamos de forma positiva cuando mucho unos 4 [2].Se puede ver que existe una saturación total de publicidad. Sin embargo, existe un bajo índice de asimilación de los contenidos que la publicidad anuncia. Lo anterior nos lleva a una pregunta: ¿Cómo se puede hacer más efectiva la publicidad?, para esto Marshall Mcluhan se refiere en su libro *"understanding media" –publicado en 1,999-* que más importante que el

[1] http://es.wikipedia.org/wiki/Publicidad#Historia
[2]SIVERA, Silvia, Marketing Viral, 2008. Editorial UOC

mensaje, es el medio sobre el cual este mensaje es enviado hacia las personas, esto es una referencia a que, si se desea hacer que el mensaje que se está enviando sea mucho más efectivo, se debe conseguir medios mucho más efectivos.

La publicidad necesita ser eficaz, para esto debe adecuarse a cada cliente, esto generalmente se conoce como "*customizing*" o amoldarla hacia cada cliente. Eficacia es lo que define a una publicidad exitosa actualmente.

Tecnología, probablemente el concepto al que se está más ligado en los últimos tiempos. Desde su concepción primaria, tecnología es casi todo lo que nos rodea y digo "casi todo" dado que se refiere a que, lo que la naturaleza brinda sin antes ser procesado por algún artefacto creado por el hombre, tiene carácter de no ser tecnología. Por otro lado, todo lo demás es denominado tecnología o producto de una tecnología que en su idea original viene a ser una tecnología; con base en que lo que es producto final de algunos es producto inicial o materia prima para otros. Es notable que nuestro mundo moderno está construido sobre las bases de la tecnología, las que proveen la infraestructura que permite alcanzar nuevos niveles en la evolución humana permitiendo, de esta forma, la sobrevivencia de nuestra especie así como mejores expectativas sobre la forma de satisfacción de nuestras necesidades.

La biblioteca del conocimiento, Wikipedia, define a la tecnología de la siguiente forma: "***Tecnología*** *es el conjunto de conocimientos que permiten construir objetos y máquinas para adaptar el medio y satisfacer nuestras necesidades*" [3]. Aquí se puede ver que, así como Ben Schneiderman en su libro "*leonardo's laptop*", propone que la tecnología debe adecuarse a las necesidades humanas, la mayoría de conceptos relacionados a la tecnología se orientan a la búsqueda

[3] http://es.wikipedia.org/wiki/Tecnologia

constante, mediante el conocimiento, de productos que se adecuen y estén destinados a satisfacer las necesidades humanas y a colaborar con las tareas para la satisfacción de las mismas.

Una buena pregunta que viene a la mente luego de leer los conceptos de “tecnología” y “publicidad” es ¿Cuál es la relación entre ambas? Muchos de los lectores pensarán que no existe relación alguna, otros pensarán que existe una relación intrínseca entre ambos. Por otro lado estarán los que prefieren no emitir juicio respecto de esto. Sin embargo es implacable la relación entre ambos, dado que se usa tecnología para hacer publicidad y se hace publicidad de tecnología, algo parecido a una relación bilateral por parte de estos conceptos. Se puede ver que, de hecho, gran parte de los avances tecnológicos han sido necesidad de la publicidad, desde que se hacía publicidad sobre radios, televisiones, con afiches, hasta las denominadas vallas publicitarias e internet y su famosa web 2.0, todo esto ha ido evolucionando poniendo a la tecnología en una constante búsqueda para encontrar nuevas formas de hacer publicidad y contribuir ella misma (la tecnología) a su publicidad.

Ahora que se sabe que ambos conceptos están íntimamente ligados, se procede a estudiar cómo estos conceptos llegaron a tener este tipo de relación. Y es que la publicidad necesita de varias áreas que le ayuden a hacer su trabajo. Como ya se ha mencionado uno de los objetivos de la publicidad es ser eficaz. Es bajo este objetivo, que la publicidad ha iniciado una constante evolución en su lucha por encontrar nuevas formas que le permitan alcanzar este objetivo. En este punto es donde la tecnología entra en acción, proveyendo los medios necesarios, efectivos que ayudan a que la publicidad sea más sensitiva. Esto permite alcanzar mejores expectativas por parte de los clientes que ven publicidad y que eventualmente serán los que adquieran los productos o servicios que son ofrecidos mediante la publicidad.

Visto desde cualquier punto de vista, ya sea del consumidor o del productor, la publicidad juega un papel importante en los servicios o bienes que se ofrecen y aún más, la tecnología como medio, se hace imprescindible en su efectividad. Tomando en cuenta lo anterior se puede inferir que una tecnología cambiante ayuda, a que la publicidad sea también cambiante.
Así también, una tecnología que es novedosa y que parece interesante al usuario o cliente, va a generar una forma novedosa e interesante de hacer publicidad.

Es necesario, entonces, no solo poseer una publicidad acerca de un bien atractivo y bueno para el consumidor, sino también tener un buen medio por el cual la publicidad pueda llegar a los personas que eventualmente se convertirán en clientes compradores de lo que se este anunciando.

El reto es encontrar nuevas tecnologías que permitan, proveer nuevas y novedosas formas de hacer publicidad, permitiendo llegar a muchos más mercados y obtener mejores resultados, por medio de la adecuación correcta y particular de las expectativas de cada cliente potencial.

1. MARCO TEÓRICO

1.1. Definición de conceptos de Proximity Marketing

Para poder identificar mejor el contexto en el que se desea desarrollar esta solución, a continuación se definen los conceptos más sobresalientes dentro del Proximity Marketing.

1.1.1. Marketing

Este término se usa generalmente para describir el proceso que se ejerce entre dos o más personas, por medio del cual, unas personas satisfacen las necesidades de las otras mediante el intercambio de los objetos que pueden ser bienes y servicios, y así cada uno cubre sus necesidades; unos su necesidad de lucro y otros su necesidad de compra.

"Los mercados se forman de conversaciones" [4], esta premisa fue la base fundamental de 95 tesis que formaron el **Manifiesto Cluetrain [5],** donde, en sus inicios se trataba de enfocar las empresas a algo más que una simple relación cliente-empresa, hacia una relación mucho más particular, con otro enfoque basado en el cliente como persona y no solo como objeto de generación de ganancias. Luego de esta convención, se han generado una serie de evoluciones de lo que el marketing se refiere, sin embargo la mayoría apunta a una relación, entre el cliente y la empresa, mucho más particular. Aquí se puede ver, entonces, que el Manifiesto Cluetrain muestra, prácticamente, la concepción original del marketing, sin embargo la nueva perspectiva sobre este marketing es enfocar los productos, que son ofrecidos por los vendedores o empresas que prestan servicios, a una mejor forma de complacer o satisfacer

[4] Cortés, Marc. Bienvenido al nuevo marketing, Claves del Nuevo Marketing, 2009.
[5] LEVINE, Rick y otros, El Manifiesto Cluetrain, Ediciones 2000.

las necesidades del cliente. Este concepto se genera mediante la concepción de que el cliente no es simplemente un objeto de generación de lucro, sino un ser humano y que como tal merece los mejores productos y servicios que alguien puede anunciar. Inherente al marketing esta la publicidad, proceso por el cual se anuncia un servicio o bien, la cual juega un papel sumamente importante en la comunicación de personas y su relación de comercializar, tanto de productos como servicios. A continuación también entran conceptos importantes como la segmentación de mercados y alcances de las estrategias de marketing, formas de publicidad: efectivas y eficaces, entre otras. Aquí también aparece el concepto del uso de la tecnología, para poder alcanzar a los mercados. De esta forma se logra que la publicidad pueda llegar a los mercados de una forma más precisa y particularizada, dependiendo de los sectores y preferencias de cada sector. Se puede ver que parte importante de un marketing es conocer el comportamiento de sus clientes potenciales y este enfoque se debe seguir.

1.1.2. Proximity Marketing

Dado que cada vez se exige la creación de nuevas formas de hacer publicidad y tomando en cuenta que la tecnología avanza rápidamente; uno de los conceptos novedosos es el uso de la tecnología inalámbrica para contactar a los clientes y enviar contenidos de publicidad que sean precisos. Esto permite la generación de resultados efectivos en el comportamiento del cliente respecto de su adquisición de los bienes o servicios que se publicitan. Usando la tecnología inalámbrica se puede contactar a los dispositivos y hacer envíos de información a cada uno de ellos (PDAs, celulares, laptops conectadas a internet, etc.).

Se puede notar que el marketing usa publicidad, y, usando la segmentación de mercados, se obtiene el marketing a proximidades o Proximity Marketing.

Existen dos tipos de Proximity Marketing, uno basado en dispositivos bluetooth y otro basado en mensajería corta o *Wi-Fi*. Este documento se enfoca en el Proximity Marketing que se hace basado en los dispositivos de bluetooth.

A continuación se presenta una definición que captura la mayoría de definiciones de Proximity Marketing para formalizar una definición.

Proximity Marketing, o por su traducción que puede hacerse a: mercadeo de proximidad o mercadeo a proximidades, es una forma de marketing que usa la publicidad mediante la tecnología inalámbrica de Bluetooth para poder contactar a las personas –clientes potenciales- y enviarles contenidos publicitarios, a fin de que estas personas puedan poseer esta información, valiosa para la toma de sus decisiones en un futuro, y ejercer el poder de adquisición del bien o producto que se anuncia.

Al usar Proximity Marketing hay que tener en cuenta que es un tipo de marketing particularizado a ciertos mercados, de hecho el concepto se refiere a usar, en su base literal, el concepto de mercados cercanos para la publicidad que se desea enviar. Esto hace que Proximity Marketing tenga resultados particulares y precisos del comportamiento de las personas en ciertos espacios de comercios.

A diferencia de otras formas de hacer publicidad, en Proximity Marketing se puede enviar publicidad en forma de amplia difusión a todos los dispositivos, y también permite enviarse de forma particular a un dispositivo en específico. Esto permite un mejor manejo de la aceptación de los contenidos publicitarios por parte de las personas. Lo importante, de este aspecto último, es la retroalimentación que brindan los clientes o personas que reciben estos contenidos al aceptar o rechazar los mismos. Esto se debe a que permite el aprendizaje sobre las preferencias de cierto grupo de personas, y luego fomentar la toma de decisiones acertadas con base en análisis previos hechos

sobre los datos del comportamiento humano proporcionados por Proximity Marketing.

1.1.3. Publicidad

Figura como una técnica que se encarga de anunciar o difundir contenidos que dan información al público acerca de un bien o servicio con el objetivo de motivar al público a ejercer una acción de consumo del bien o servicio anunciado.

Gran parte de las personas que se refieren a la publicidad hacen referencia, unas, a que la publicidad es un arte de presentar contenidos para convencer a las personas, y otras, que la misma es una disciplina científica usada para persuadir a las personas hacia la compra de un bien o servicio anunciado. En ambos caso se concluye que lo que se desea generar es una acción de las personas por adquirir lo que se esta anunciando. De hecho se puede concluir que es un arte y ciencia al mismo tiempo, dado que como arte se comprende la creatividad para generar contenidos atractivos a difundir y como ciencia la estructura de pasos no solo para crear los contenidos sino para anunciarlos mediante cualquier medio del cual se haga uso.

La publicidad, entonces, debe encontrar un nicho de mercado sobre el cual se pueda enviar o ejercer. La misma cuenta con elementos como:

- El receptor.
- El mensaje.
- El medio del mensaje o canal.
- El emisor.
- La retroalimentación que cada persona desea hacer hacia lo que se le esta enviando como contenido publicitario.

Estos elementos permiten hacer un estudio particularizado sobre cada persona que recibe el mensaje publicitario. A partir de esto se pueden hacer modelos del comportamiento, que permitan predecir o en un alto porcentaje los gustos de un cliente en especifico o de un conglomerado, según sea el caso.

Como idea global se puede ver que, a diferencia de la propaganda, en la que se desea que una persona adopte un pensamiento u opinión de cierta persona, de la que se hace propaganda, en la publicidad lo que se busca es persuadir a que las personas ejerzan el poder de compra o adquisición del bien o producto en mención.

1.1.4. Formas de hacer publicidad

1.1.4.1. Tv/Radio

Son de los medios que se han tenido presente en el siglo pasado, como vitales para la difusión de publicidad, estos medios proporcionan un ambiente unidireccional para la difusión de contenidos, poseen la ventaja de que pueden ser ampliamente distribuibles por la efectividad de su alcance. Sin embargo poseen la desventaja, de ser medios de una sola dirección, es decir proporcionan monólogos entre un publicista y una persona del público en general.

1.1.4.2. Diario y medios escritos

Es la forma base de la publicidad, inicia desde que las primeras personas tuvieron la necesidad de anunciar sus productos así también los servicios que ofrecían. Desde que las personas iniciaron el trueque, tanto de los objetos como de los servicios, que cada uno poseía y deseaba algo a cambio, en este sentido los medios escritos fueron los primeros en ser usados para que la gente

conociera lo que las otras personas anunciaban o deseaban dar a conocer. Hoy en día los diarios, periódicos, volantes, afiches, entre otros, son medios que suelen ser comunes para anunciar todo tipo de eventos, esto incluye también la publicidad de venta de bienes o prestaciones de servicios.

1.1.4.3. Internet y medios interactivos

Constituyen los medios de mayor efectividad en los últimos años, sobre todo porque proporcionan un enlace bilateral entre el anunciante y el público. Crean el ambiente propicio para la efectividad de la publicidad y dan mejores datos en lo que a retroalimentación se refiere. Entre los medios interactivos es donde se sitúa la publicidad que se hace mediante Proximity Marketing basado en dispositivos de bluetooth.

1.1.5. Interacción cliente-publicidad

El cliente siempre desea tener el control de los sucesos, es por esto que en publicidad siempre las técnicas están enfocadas hacia una mejor interacción entre el cliente y la publicidad, esto es lo que mejora una publicidad y la hace efectiva en sus resultados.

Una de los aspectos críticos, generalmente olvidados, es lo que se denomina base-en-permiso, o lo que en inglés es *permission-based*, y hace referencia a que el cliente siempre debe ser el único que decida, si aceptar o no, un contenido publicitario. Este ha sido uno de los problemas que se perfilan últimamente en la publicidad, dado que un cliente puede no desear recibir cierta publicidad y la misma le llega sin aceptación alguna. Este último hecho es lo que en términos generales se llama *spam*; en donde no existe una pregunta hacia el cliente si desea recibir o no los contenidos, que están a punto de ser enviados hacia su persona, y simplemente le son enviados. Esto ocasiona, en

la mayoría de casos, el disgusto del cliente, formando una predeterminación de repulsión hacia los contenidos publicitarios.

Es por esto que, para proponer un modelo efectivo de publicidad sobre los mercados y el público en general, se debe proveer un modelo que ofrezca control al cliente y le de el derecho de aceptar o no, un contenido de publicidad que se intenta difundir; esto, precisamente, es lo que proporciona Proximity Marketing.

1.1.6. Interacción publicidad-publicista

Este tipo de interacción se basa en la forma en la que la persona, que genera publicidad, realiza este proceso. Para esto se deben proporcionar herramientas de tecnología que permitan estandarizar los procesos y que guíen al publicista en la forma de realizar los mismos. Es notable ver que en la actualidad existen muchos programas de computadoras que permiten a un agente publicista, de forma fácil y rápida, realizar contenidos y luego difundirlos a las personas que éste desea. Esto muestra que la relación “publicidad-publicista” se ha cambiado a una expresión, quizás la más acertada, la cual figura como interacción “herramienta-publicista”. Tomando a una herramienta como un programa de computadora o software especial que permite a un publicista crear publicidad.

En el ámbito de las herramientas de software, estas deben proporcionar formas fáciles y amigables que interactúen con el publicista y que lo ayuden a cumplir sus objetivos de forma precisa y efectiva. En esta tesis se presentan una serie de buenas prácticas para la construcción de una solución de software que permita hacer este tipo de proceso interactivo entre una herramienta de software y un publicista.

1.1.7. Hacia un nuevo marketing

Luego de que se haya visto los conceptos referentes a marketing se puede decir que, para que un marketing pueda ser efectivo, teniendo un nuevo enfoque basado en el cliente [6], debe tener, entre otros, los siguientes principios.

- Escuchar a los mercados: esto permite escuchar a los clientes, de una forma más cercana, distinta a la concepción antigua de necesidades del cliente.
- Construcción participativa: esto permite que el cliente sea quien pida o realice, mediante su participación, la transformación de técnicas de mercadeo para que puedan ajustarse mejor a sus expectativas.
- Mejorar la Visibilidad: mejorar las conversaciones, voluntad de participar, construir los medios para que exista expresión de los clientes y hacer caso a esta participación.
- Eficiencia publicitaria: el uso de nuevas herramientas que permitan que el mensaje llegue con mejor disposición a los mercados a los cuales esta dirigido el mensaje.

Estas premisas están relacionadas al enfoque que se hace sobre el marketing, en este caso las mismas tienen un enfoque sobre la forma de hacer que el marketing sea mucho mas particularizado hacia el cliente, esto es tomar características que el cliente prefiere y ajustar el tipo de marketing hacia estos clientes.

[6] Cortés, Marc. Del marketing 1.0 al 2.0: 10 + 1 tendencias, Claves del nuevo marketing. 2009.

1.1.8. Proximity Marketing: Características únicas y ventajas

Proximity Marketing otorga ciertas ventajas en su uso, y posee características que ninguna otra forma de hacer publicidad posee. A continuación se definen las ventajas y características.

1.1.8.1. Características únicas de Proximity Marketing

1.1.8.1.1. Proporciona datos reales del comportamiento humano

Se puede ver que, a nivel del sistema de software, se pueden obtener datos reales del comportamiento humano. Esto es así, porque cuando a un cliente se le envía contenido a su dispositivo y luego éste decide aceptarlo, en efecto se sabe que el cliente ha aceptado el contenido y, cuando no lo acepta, se sabe que lo ha denegado, no es una suposición. Esto proporciona el camino para la estadística, dado que si se obtiene estos datos simples, luego pueden ser analizados y así poder inferir a partir de estos. Sin embargo no serían proyecciones o conjeturas basadas en datos imprecisos, sino conclusiones estadísticas basadas en datos precisos del comportamiento humano.

1.1.8.1.2. Permite analizar los datos obtenidos

Luego de haber recogido los datos que se tienen, pueden ser analizados. Esto permite que se puedan inferir comportamientos, sin embargo un punto importante de estos datos es que, estadísticamente, no son una proyección de lo que se piensa que se tendrá, sino más bien es lo que realmente se tuvo cuando se realizó el proceso. A su vez para cualquier persona que desee formular modelos de estimación de estos datos, puede tomarlos como una base sobre la cual realizar la estimación. Esto estaría representando el estudio de

mercado para un nuevo producto, la apertura de un nuevo comercio y poder predecir lo que las personas preferirán en su visita a cierto centro comercial.

1.1.8.1.3. Permite la toma de decisiones con base en el análisis de los datos obtenidos

Un aspecto importante de Proximity Marketing es que permite ver las preferencias de un cliente en determinado lugar donde sea implementado. Como un ejemplo, se presenta el escenario de un centro comercial donde existan una gran cantidad comercios destinados a la venta de artículos de vestir, unos pocos destinados a vender comida y otros destinados a oficinas administrativas; suponiendo que en un día normal de la semana, por medio del escaneo del lugar, es encontrada gran cantidad de dispositivos. Se emite un mensaje de 10% de descuentos en artículos de vestir de determinados comercios, así también 10% de descuento en comidas y 10% de descuento en servicios de oficinas administrativas y asesorías. Pensemos, ahora en que de estos dispositivos escaneados –que pertenecen a cada persona- el 70% prefirió canjear su 10% de descuento en comidas, un 25% prefirió canjear su descuento del 10% en artículos de vestir y un 3% prefirió canjear su descuento en servicios administrativos y asesorías. Aquí se puede ver que existe una gran preferencia, de parte de los clientes, de este centro comercial, por las comidas antes que por los servicios administrativos y los artículos de vestir. Esto proporciona información real, acertada de lo que son las preferencias de los clientes que visitan este centro comercial, esto hace posible que la toma de decisiones futuras por parte de los administradores del mismo, sean mucho más acertadas y por consiguiente con mejores resultados para ellos.

1.1.8.2. Ventajas

1.1.8.2.1. Se basa en el permiso del cliente

Como se había mencionado anteriormente, una de las ventajas es que en el Proximity Marketing se pregunta por el consentimiento de las personas, si desean recibir los contenidos que se están intentando intercambiar (permission-based), esto permite que la publicidad no se vuelva molesta –como actualmente es el *spam* en celulares- y permite el interés de las personas contactadas, por desear saber a que se refieren los contenidos que están recibiendo. Claro está que si alguien no desea recibirla simplemente no aceptará la negociación para intercambiar contenidos y en este punto se termina el proceso.

Dado que uno de los principios de Proximity Marketing es el permiso del cliente, se obtiene como ventaja el hecho de no ocasionar que el cliente pueda tomar como *spam* los contenidos que se le están enviando. Esto para no ocasionar disgusto en el cliente y poder ver de una mejor forma los contenidos, asimilando lo que se le está tratando de decir con éstos, y provocar, de esta forma, el poder de compra del mismo sobre lo que se está anunciando.

1.2. Definición técnica de los dispositivos y conceptos usados en la solución

En este punto, se muestran los principales conceptos y dispositivos que son de interés en esta solución, esto, tomando en cuenta que la solución esta basada en fomentar Proximity Marketing, que se realiza en dispositivos inalámbricos de bluetooth.

1.2.1. Software

1.2.1.1. Servidor de contenidos

Un servidor común, hace referencia a un dispositivo de software, montado en una computadora, el cual se dedica a ejecutar tareas para las cuales fue

diseñado o construido, es decir ésta es su tarea; ejecutar las tareas que se le solicitan que son del mismo género a las que fue construido.
Exactamente con esta definición se puede mencionar que, el servidor de contenidos, en este caso, se enfoca a un dispositivo de software, montado sobre una computadora, el cual permite que los administradores de la publicidad o publicistas puedan hacer tareas específicamente relacionadas a publicidad esto es:

- Subir contenidos de publicidad
- Bajar o borrar contenidos cuando ya no sea necesario tenerlos disponibles.
- Cambiar fechas de disponibilidad de contenidos
- Modificar los distintos contenidos y sus respectivas características.
- Enviar contenidos a dispositivos
- Seleccionar grupos de dispositivos, o en su defecto dispositivos individuales para envío de contenidos. Entre otros.

Se puede ver que este servidor, puede ser mucho más fácilmente mapeado, análogamente, a un dispositivo de software o programa de computadora que permite administrar los contenidos que serán distribuidos como publicidad, así también las tareas que permiten que estos contenidos puedan ser entregados a los distintos dispositivos que serán el mercado potencial.

1.2.1.2. Servidor controlador de enrutadores

Para este caso, tomando la anterior definición de un servidor, se dice que un servidor de control de enrutadores, es un producto de software que se encarga de administrar los distintos enrutadores de las antenas de Bluetooth. Esto se hace mediante el reconocimiento de las mismas cuando son conectadas.

También este servidor se encarga del manejo de los enrutadores para que los algoritmos que se prevén, puedan funcionar correctamente, ajustándose a las condiciones actuales sobre las cuales el programa de *software* esta corriendo. En general, este programa ayuda a que la aplicación pueda estar funcionando sin que exista la necesidad de ser nuevamente establecida con nuevos parámetros y esto requiera intervención humana. En esta solución, este dispositivo de *software* (servidor controlador) ya esta incluido en el algoritmo del Scanner como parte de su trabajo.

1.2.1.3. Red de servidores

Se propone una red de servidores, en proyectos que puedan ser destinados a centros comerciales o mercados mucho mas grandes que los convencionales, esto tomando en cuenta que deben existir varios servidores que se reparten entre ellos el área total del mercado o centro comercial. Para esto se necesitará una forma inteligente de saber que un dispositivo de Bluetooth, está siendo contactado varias veces por publicidad (lo cual podría ser molesto para el usuario del dispositivo relacionado) y recibiendo la misma. Entonces, la red de servidores funcionaría de una forma tal, que los mismos se comunicarían entre ellos a fin de saber cuales dispositivos han recibido suficiente publicidad anteriormente, y no volver a enviarles. Esto se hace para no incurrir en molestias de parte de los usuarios, por demasiada publicidad emitida.

Otro aspecto que es muy importante en la red de servidores es, la repartición de carga para cada uno; propiciando que no pueda ser saturado un servidor con demasiada carga mientras otro servidor esta sin carga relativa. Es entonces necesario que exista una red de servidores para que la carga pueda ser balanceada y obtener un mejor rendimiento de esta forma.

1.2.2. Hardware

1.2.2.1. Enrutador de antenas bluetooth

Son conocidos como *hubs* de antenas bluetooth, la idea principal de estos dispositivos de hardware, es que exista un dispositivo que permita agrupar un número de antenas y que luego, estas puedan tener una comunicación en común para no usar demasiado recurso. En este caso las antenas de Bluetooth, poseen conexión de puerto tipo USB, pensando en el escenario de 6 antenas para una computadora, se estaría hablando de que es necesario que dicha computadora debiera soportar 6 puertos USB listos para ser usados por las antenas, lo cual presenta una desventaja para la búsqueda de los dispositivos.

Ahora bien si se desea usar 8 antenas, se debe hacer especificaciones sobre los 8 puertos de USB al proveedor de equipo a fin de poder cumplir con los requerimientos, es por esto que se usan los enrutadores; de esta forma se puede tener varias antenas conectadas a un enrutador y la cantidad de enrutadores como puertos USB lo permitan.

Generalmente estos dispositivos (enrutadores) proveen 4 puertos para antenas, además de poseer conectividad, del dispositivo hacia la computadora, de puertos USB, así que se puede tener varias antenas conectadas a un mismo dispositivo y luego éste ultimo comunicarse directamente a la computadora mediante puertos USB.

1.2.2.2. Antenas bluetooth

Las antenas bluetooth, son dispositivos de bluetooth, usadas como dispositivos individuales, es decir que no dependen de otros componentes. Son dispositivos fáciles de usar, como los dispositivos convencionales que son usados como componentes adicionales de las computadoras.

Estas antenas permiten la comunicación con otros dispositivos y por su conexión USB pueden ser conectadas a computadores como un dispositivo periférico (*plug and play*) y luego de ser reconocidas por la computadora, ser usadas. La idea principal, de usar estas antenas, es que puedan ser portables y poder ser usadas cuando sean necesitadas, permitiendo que un dispositivo de Bluetooth, que este contenido en celulares o dispositivos móviles, laptops e incluso otras antenas, entre otros, puedan tener comunicación entre ambas usando los protocolos y tecnología provista por los dispositivos de bluetooth.

1.2.2.3. Dispositivos bluetooth para móviles

Los dispositivos de Bluetooth para celulares móviles son antenas que previamente el fabricante de los celulares ha ensamblado directamente al celular, para que pueda ser usada en comunicación con otros dispositivos. Estas antenas poseen una característica particular; que no pueden ser removidas cuando el usuario lo desee y luego conectadas cuando el mismo lo desee también, esto debido a que el fabricante del celular, o dispositivo al cual esta relacionado, desde su manufactura la ensambló para que sea de uso exclusivo de este dispositivo, es por esto que estos dispositivos no permiten que sus componentes de Bluetooth, puedan ser usados en otros dispositivos.

1.3. Definición técnica de la comunicación entre dispositivos de bluetooth y sus conceptos más importantes

La humanidad siempre ha tenido la necesidad de intercambiar todo tipo de objetos, entre objetos tangibles como alimentos, vestido, útiles, enceres, entre otros. Así también los objetos intangibles son parte importante del intercambio diario entre personas.

Sin embargo para el intercambio de objetos intangibles como información o, simplemente, datos, se necesitan de procesos más complejos como el lenguaje humano (tomando en cuenta los distintos lenguajes actualmente disponibles). Además, es necesario que exista una forma definida de intercambiar este tipo de objetos.

Para el intercambio de información, entonces, existen formas que definen nuevos aparatos o medios de comunicación y una de ellas es la comunicación inalámbrica desde dispositivos electrónicos.

Parte de estos dispositivos electrónicos poseen estructuras o modelos de los pasos que se deben seguir cuando se desea intercambiar información entre dos entes que deciden hacerlo. A su vez los entes que deciden intercambiar objetos (de cualquier tipo) juegan papeles importantes en este proceso, papeles que son de mucho interés cuando se desea plantear una solución de software que represente este proceso de una forma automatizada o estandarizada mediante programas de computadora. A continuación se describen los aspectos más importantes dentro de la comunicación entre dispositivos electrónicos de tecnología Bluetooth.

1.3.1. Comunicación entre dispositivos electrónicos de tecnología Bluetooth

Se da cuando un dispositivo decide hacer intercambio de información con otro dispositivo que está disponible y acepta hacer el intercambio de la información con el primer dispositivo.

Para que la comunicación pueda ser efectuada deben existir reglas que establezcan la forma correcta de intercambio de información. A estas reglas generalmente se les conoce como el protocolo de comunicación, además también deben existir formas a un nivel más inferior que permitan la

manipulación de la información que se esta enviando y/o recibiendo. Adicionalmente, la parte del hardware se encarga del nivel más inferior que existe en la comunicación.

1.3.2. Roles de los dispositivos en la comunicación

En la comunicación entre dispositivos de bluetooth existen dos roles necesarios para que se dé el intercambio de información estos son:

- Cliente
- Servidor

La clasificación del rol que le corresponde a cada dispositivo lo determina la forma en la que la comunicación se hace, es decir se basa en quién es el primero que intenta comunicarse.

1.3.2.1. Cliente

En este caso, el dispositivo que desempeña el papel del cliente, es el dispositivo que se encarga de iniciar la comunicación, es el primero que hace la petición al servidor para iniciar un proceso de intercambio de información o comunicación. Dejando claro que la comunicación entre los dispositivos puede ser de ambas vías o de una sola vía, es decir cuando los dos envían y reciben información el uno del otro o cuando solo un dispositivo envía información y el otro recibe la misma, respectivamente.

No existe diferencia entre una antena simple de Bluetooth o un dispositivo de Bluetooth que este previamente ensamblado en un dispositivo móvil celular, aquí los roles son ejecutados por la sincronización, es decir cualquier dispositivo de Bluetooth puede ser el cliente, siempre que sea éste el que se encargue de iniciar la conversación o comunicación entre los dispositivos relacionados. **Ver tabla II: Componentes del Obex Application Framework.**

1.3.2.2. Servidor

Es el rol que desempeña el dispositivo que es contactado por el otro dispositivo (cliente) para iniciar una comunicación en la que se intercambia información. Es este rol quien se encarga de esperar a que un cliente desee conectarse al mismo, para poder iniciar un intercambio de información.

Este rol es desempeñado por cualquier dispositivo, no es necesario que sea un dispositivo de Bluetooth de celulares o una antena en específico, sino cualquier dispositivo que sea el contactado y no el que esté contactando. **Ver tabla II: Componentes del Obex Application Framework.**

1.3.3. Protocolos

Por definición, un protocolo es un conjunto de reglas que hacen que una comunicación o proceso de comunicación sea desarrollado en una forma específica. Para la comunicación entre los dispositivos de Bluetooth el protocolo define la forma en que los dos dispositivos, -cliente y servidor- se ponen de acuerdo en la forma en la cual se comunicarán. Existen protocolos como OBEX que proporcionan la estructura de comunicación, así también proporcionan un modelo de objetos sobre los cuales los paquetes de la comunicación serán enviados/recibidos, además de la utilería que proporciona las formas y los soportes para que la comunicación, entre los dispositivos, pueda ser sostenida correctamente.

1.3.3.1. Obex

Es el acrónimo para Protocolo de Intercambio de Objetos (por sus siglas en ingles: *Object Exchange Protocol*), desarrollado inicialmente para los dispositivos de transmisión infrarroja, este protocolo define la forma de comunicación para los dispositivos de este tipo. La asociación de Datos Infrarrojos –IrDA- (por sus siglas en inglés: *Infrared Data Association*) propone el protocolo OBEX para estructurar una forma de comunicación por la cual dos dispositivos de luz infrarroja deben comunicarse a fin de intercambiar información.

Para este caso se usa una modificación al protocolo OBEX denominada IrObex, dado que es un protocolo que se parece en gran parte al protocolo DHCP (más adelante se presenta analogía) y es más fácil mapear la funcionalidad de uno a otro.

Se usa, entonces IrObex, para estandarizar la forma en la que se generará la comunicación entre los dispositivos que estén relacionados. Se inician con paquetes de descubrimiento de dispositivos, haciendo que un dispositivo, que se encuentra en un área que esta cubierta por la señal de la antena que invita a la negociación, pueda contestar enviando un paquete de respuesta al paquete de descubrimiento previamente enviado, a partir de esto luego se inician las negociaciones de los parámetros de la comunicación, esto es: puerto de comunicación, tamaño de los paquetes, comunicación asíncrona, directa, entre otros. Luego de esto se usan paquetes PUT para enviar un contenido y paquetes *GET* para obtener paquetes a petición de los clientes. El envío de los paquetes de tipo *PUT* y *GET* es análogamente parecido al protocolo HTTP con sus paquetes *request-response*. Con cada envío de un paquete se responde con otro; en éste se indica si se recibió el paquete correctamente o no.

Generalmente, el cliente es quien envía los paquetes de tipo *PUT* dado que es él quien se encarga de iniciar la conversación con un servidor (ver 1.3.2 Roles de los dispositivos en la comunicación), luego, si el servidor desea saber alguna información o dado el caso, un paquete que el cliente envió y que el servidor nunca recibió, el servidor puede contestar con un paquete de tipo *GET* para indicarle al cliente que le vuelva a enviar dicho paquete dado que en algún punto de la comunicación el mismo fue perdido y no pudo ser entregado a su destinatario.

A continuación, se presenta una analogía del protocolo DHCP y el Protocolo IrObex.

Tabla I. Analogía Protocolo DHCP – IrObex

DHCP	**IrObex**	**Descripción**
Discovery	Obex Discovery	Se envía un paquete para que los dispositivos cercanos contesten. Esta operación se hace en modo *Broadcast* (dirigido a todos los dispositivos cercanos).
Offer	Obex Response	Los dispositivos cercanos responden a la dirección que está haciendo el *Discovery.*
Request.	Obex Connect-request.	Envía el paquete de conexión hacia el servidor, para iniciar la conexión.
Acknowledge	Obex Connect-esponse.	El servidor responde y acepta o deniega la conexión.
Desconexión.	Obex Disconnect. (request-response).	En este punto se envían paquetes para finalizar la conexión y el servidor

		contesta aceptándola.

A continuación, se describen los componentes del Protocolo OBEX dado que la especificación de componentes de éste protocolo no varía en relación con el protocolo IrObex, el cual es usado por dispositivos de bluetooth.

1.3.3.1.1. Componentes Obex.

La especificación de OBEX consiste en dos grandes partes: el protocolo (OBEX Session Protocol) y el marco de aplicación (OBEX Application Framework) [7], en donde ambos son estructuras que permiten la comunicación entre dos dispositivos, también poseen un modelo que permite representar objetos a fin de ser intercambiados entre dos dispositivos. Su principal propósito es facilitar la interoperabilidad entre dos dispositivos.

1.3.3.1.1.1. Obex Session Protocol.

Este componente proporciona el modelo para representar objetos y el protocolo en sí para proveer la estructura de comunicación entre los dos dispositivos.

El modelo es quien lleva toda la información acerca de los objetos que están siendo enviados así como contener los objetos mismos, esta construido con cabeceras completamente analizado en su estructura sintáctica, como cualquier cabecera basada en HTTP.

[7] Infrared Data Association, OBEX™ version 1.3.

Así también, el protocolo es quien proporciona la forma básica de comunicación entre los dos dispositivos, está basado en paquetes binarios del tipo cliente/servidor y usando el modelo *request-response*.

1.3.3.1.1.2. Obex Application Framework.

Es la base para un conjunto de servicios que son provistos por OBEX así como los requerimientos del intercambio de objetos, es el marco de trabajo de la aplicación quien propone los distintos objetos y/o roles que se pueden ver dentro de OBEX, dentro de los elementos podemos ver lo siguientes:

Tabla II. Componentes del Obex Application Framework

Elemento	**Descripción**
Cliente OBEX	Es la entidad que inicia la conexión de transporte hacia un servidor OBEX e inicia las operaciones
Servidor OBEX	Es la entidad que responde a las operaciones OBEX, éste espera por un cliente para que se pueda iniciar una comunicación
Servidor Default OBEX	Este es el servidor propio, es análogo al "*localhost*" usado en los protocolos HTTP, con el puerto 80
Conexión de Transporte OBEX	Es la capa de transporte de la conexión que carga o lleva el protocolo OBEX.
Conexión OBEX	Es un enlace virtual entre dos aplicaciones o servicios, es iniciada al enviar un paquete CONNECT, una vez iniciada la conversación todos los paquetes son enviado

	usando esta conexión.
Conexión directa	Es un tipo de conexión en donde los paquetes OBEX contienen información acerca de su destino, esta información es usada por el protocolo OBEX para ser entregada al correcto destinatario.
El buzón de entrada de la conexión	Es un tipo de conexión hecha al servidor default, donde los paquetes no contienen información acerca del destinatario ya que es conocido. Se puede tener acceso a un gran número de servicios a través del buzón de entrada de la conexión.
Buzón de entrada	Es el destinatario al que se envían los objetos (Paquete PUT). No necesariamente debe ser un tipo de dispositivo de almacenaje, mas conceptualmente puede ser un método que permita a un cliente colocar objetos en este recipiente sin necesidad de preocuparse sobre los detalles de cómo se maneja el almacenaje de los mismos.
Aplicación	Una aplicación OBEX, es una aplicación que se encarga de comunicar, usando métodos conocidos solo por su fabricante, a dos dispositivos mediante el protocolo OBEX. Es quien pone en práctica el protocolo con sus elementos.
El servicio	Un servicio es quien hace la comunicación, usando procedimientos especificados en un estándar disponible.
La capacidad del servicio	Es usada para encontrar información acerca del servidor OBEX, incluyendo información del dispositivo, tipos de

	objetos soportados, perfiles de objetos y aplicaciones soportadas.

Fuente: IrDA, OBEX Especification, versión 1.3, 2003.

Como se mencionó anteriormente, la analogía con DHCP hace mejor la comprensión de este protocolo, usando objetos basados en paquetes que son enviados y que los destinatarios pueden analizar sintácticamente para poder obtener información de los mismos, esto para lograr el objetivo de la entrega de la información de un dispositivo a otro.

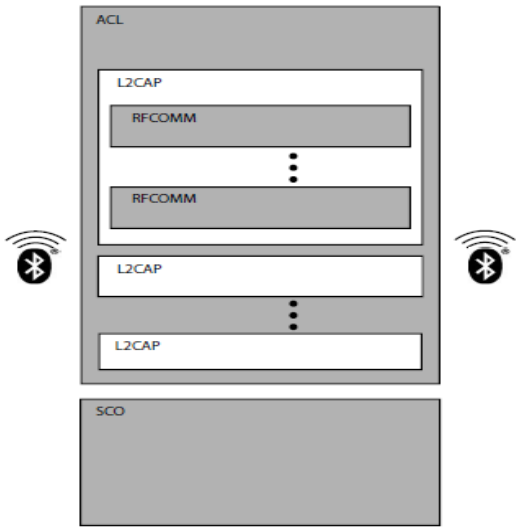
A continuación, se presenta la arquitectura sobre la cual Obex se establece con el fin de tener una mejor comprensión.

1.3.3.2. L2Cap

El Protocolo de enlace lógico, control y adaptación por sus siglas en inglés: *Logical Link Control and Adaption Protocol,* es un protocolo basado en paquetes con varios niveles de confiabilidad.

Es una forma de comunicación entre dos dispositivos, generalmente usada para conectarse a un dispositivo y poder iniciar una comunicación entre ambos. Éste ayuda a pasar los paquetes entre el sistema manejador y los dispositivos sobre los cuales se esta haciendo la comunicación, este es usado por RFCOMM para poder hacer las respectivas entregas de los paquetes, bidireccionalmente, a los dispositivos de Bluetooth relacionados en una comunicación.

Figura 1. Modelo de capas de L2Cap



Fuente: Albert S. Huang, Larry Rudolph. Bluetooth Essentials for programmers. Cambridge. 2009.

1.3.3.3. Sockets Rfcomm

Denominado así por sus siglas en inglés: *Radio Frequency Communications Protocol*, es un protocolo basado en flujo de información lo que en inglés se conoce como *streams-bassed*. Provee casi los mismos servicios y garantías que el protocolo TCP.

Los sockets RFCOMM son objetos que se encargan de abrir las comunicaciones entre dos dispositivos, estos son los encargados de generar las sesiones entre dos dispositivos y manejar varias sesiones por cada socket.

Como socket, estos son parte del modelo de objetos que proporciona el protocolo IrObex, este concepto está mucho más familiarizado en las comunicaciones entre redes de computadoras, sin embargo el principio que se sigue es el mismo: tener un programa que intercomunique a los procesos del mas bajo nivel con los procesos del alto nivel como las aplicaciones de usuario,

es entonces, el socket, quien se encarga de manejar las conexiones de las comunicaciones entre dos dispositivos.

Algunos protocolos son usados para manipulación de objetos en comunicaciones entre dispositivos de luz infrarroja y otros en dispositivos de Bluetooth, la idea es poder usar los protocolos predefinidos para Bluetooth -L2capp y Rfcomm- pero con ciertos cambios basados en el protocolo Obex –comunicación infrarroja- con el fin de mejorar la forma de comunicación entre los dos dispositivos. A continuación, se presenta las capas entre Rfcomm y L2cap para una mejor comprensión.

1.3.3.4. IrObex

Es un protocolo de sesiones, designado por la Asociación IrDA. Este protocolo es ahora usado por la tecnología Bluetooth, haciendo posible que las aplicaciones puedan usar tanto la tecnología de Bluetooth como la tecnología infrarroja de IrDA. Ambos (Bluetooth e IrDA) son usados para comunicaciones inalámbricas de corto rango. Poseen ciertas diferencias relacionadas a las capas bajas de los protocolos. IrObex, por lo tanto, es mapeado sobre las capas bajas de los protocolos que son adoptados por Bluetooth.

IrObex es el punto en el cual se interceptan Bluetooth y Obex. La IrDA presenta la modificación al protocolo OBEX como IrObex para proporcionar las ventajas de comunicación en bajo rango que proporciona OBEX hacia las funcionalidades que proporciona Bluetooth. Éste protocolo, que funciona como una modificación al protocolo OBEX para ser usado en tecnología Bluetooth, proporciona las reglas de comunicación entre dos dispositivos de Bluetooth (cliente-servidor) con los paquetes del tipo request-response que proporciona por default OBEX.

1.3.3.4.1. Tipos de paquetes de IrObex

IrObex proporciona los siguientes tipos de paquetes:

- Connect-Request: paquete para iniciar la conexión.
- Connect-Response: respuesta del servidor al paquete Connect-Request.
- Put-Request: sirve para colocar contenidos en el servidor.
- Put-Response: sirve para notificar que se ha recibido el paquete correctamente.
- Get-Request: sirve para obtener contenidos del servidor o del cliente.
- Get-Response: sirve para notificar que se ha recibido el paquete correctamente, de tipo get y se procede a enviarlo.
- Disconnect-Request: para pedir la finalización de la conexión.
- Disconnect-Response: para afirmar que la conexión se finaliza.

1.3.3.4.2. Características importantes de IrObex

Este protocolo provee un modelo que maneja la información de los objetos y los objetos en sí mismos. Cada paquete posee características. Estas características son vistas como cabeceras, las más importantes son:

- *Count*: contador de paquetes.
- *Name*: nombre del paquete.
- *Type*: tipo del paquete.
- *Description*: pequeña descripción del paquete.
- *Target*: el objetivo o destinatario hacia el que va dirigido el paquete.
- *Body*: define el cuerpo del paquete.
- *End of Body*: define el fin del cuerpo del paquete. Entre otros.

Como ya se ha mencionado, el protocolo IrObex, maneja paquetes del tipo *request-response*. Esto permite tener paquetes de distinto tipo para poder manejar las conexiones.

1.3.3.4.3. Ejemplos de paquetes IrObex

A continuación, se presentan los tipos de paquetes de IrObex; los paquetes para abrir una conexión, cerrarla y para enviar contenidos, así como hacer peticiones de paquetes.

1.3.3.4.3.1. Paquetes: conexión y desconexión

A continuación, se presentan los formatos y ejemplos de los paquetes manejados en IrObex/Obex. De aquí en adelante, para las referencias del *Opcode* en los formatos y ejemplos, se debe ver el apéndice 2. Sección 5.2: Códigos de Operación (OpCode), de esta tesis. También puede referirse al libro: IrDA, Obex Exchange Protocol, versión 1.3. 2003, pagina 25 o también al apéndice 1 de esta tesis.

1.3.3.4.3.1.1. Paquete de conexión

La sesión Obex es iniciada cuando una aplicación envía un paquete del tipo Connect-request (conexión); el formato es el siguiente:

Figura 2. Formato de un paquete connect-request

Byte 0	Bytes 1 and 2	Byte 3	Byte 4	Bytes 5 and 6	Byte 7 to n
0x80 (opcode)	Connect request packet length	OBEX version number	Flags	Maximum OBEX packet length	Optional headers

Fuente: BLUETOOTH SPECIFICATION Version 1.1 page 426 of 1084, IrDA Interoperability, página 8.

Cuando el servidor recibe este paquete, debe contestar en el siguiente formato:

Figura 3. Formato de un paquete connect-response

Byte 0	Bytes 1 and 2	Byte 3	Byte 4	Bytes 5 and 6	Byte 7 to n
Response code	Connect response packet length	OBEX version number	Flags	Maximum OBEX packet length	Optional headers

Fuente: BLUETOOTH SPECIFICATION Version 1.1 page 426 of 1084, IrDA Interoperability, página 8.

Un ejemplo de este tipo de paquetes es el siguiente:

Figura 4. Ejemplo de paquetes tipo connect

Client Request:	bytes	Meaning
Opcode	0x80	**CONNECT**, Final bit set
	0x0011	packet length = 17
	0x10	version 1.0 of OBEX
	0x00	flags, all zero for this version of OBEX
	0x2000	8K is the max OBEX packet size client can accept
	0xC0	HI for **Count** header (optional header)
	0x00000004	four objects being sent
	0xC3	HI for **Length** header (optional header)
	0x0000F483	total length of hex F483 bytes
Server Response:		
response code	0xA0	SUCCESS, Final bit set
	0x0007	packet length of 7
	0x10	version 1.0 of OBEX
	0x00	Flags
	0x0400	1K max packet size

Fuente: IrDA, Obex Exchange Protocol, versión 1.3. 2003, pagina 28.

Luego de este paquete, las aplicaciones que contesten, podrán entablar conversación con la aplicación que los contactó.

1.3.3.4.3.1.2. Paquete de desconexión

Este paquete es enviado para finalizar la conexión, debe cumplir con el siguiente formato:

Figura 5. Formato de un paquete disconnect-request

Byte 0	Bytes 1 and 2	Byte 3
0x81	Packet length	Optional headers

Fuente: BLUETOOTH SPECIFICATION Version 1.1 page 426 of 1084, IrDA Interoperability, página 9.

El servidor, para finalizar por completo la conexión, contesta con el siguiente formato:

Figura 6. Formato de un paquete disconnect-response

Byte 0	Bytes 1 and 2	Byte 3
0xA0	Response packet length	Optional response headers

Fuente: BLUETOOTH SPECIFICATION Version 1.1 page 426 of 1084, IrDA Interoperability, página 9.

Para el ejemplo anterior del paquete, solo es necesario cambiar el *Opcode* para finalizar la conexión y el paquete lucirá de la misma forma.

1.3.3.4.3.2. Paquetes: put y get

Una vez la comunicación se haya establecido, los paquetes se vuelven del tipo *put* y *get*. Se usan estos paquetes como distintivos para enviar o recibir datos relacionados al mensaje o a la información que se desea enviar. El tamaño máximo que establece el protocolo para el envío de paquetes es 64 Kbyte – 1.

1.3.3.4.3.2.1. Paquete put

Se usa este paquete para colocar contenidos en el servidor, éste debe tener el siguiente formato:

Figura 7. Formato de un paquete put-request

Byte 0	Bytes 1 and 2	Byte 3
0x02 (0x82 when Final bit set)	Packet length	Sequence of headers

Fuente: BLUETOOTH SPECIFICATION Version 1.1 page 426 of 1084, IrDA Interoperability, página 9.

El servidor contesta en el siguiente formato:

Figura 8. Formato de un paquete put-response

Byte 0	Bytes 1 and 2	Byte 3
Response code	Response packet length	Optional response headers

Fuente: BLUETOOTH SPECIFICATION Version 1.1 page 426 of 1084, IrDA Interoperability, página 9.

Un ejemplo del paquete de put es el siguiente:

Figura 9. Ejemplo de paquetes del tipo put

Client Request:	Bytes	Meaning
opcode	0x02	**PUT**, Final bit not set
	0x0422	1058 bytes is length of packet
	0x01	HI for **Name** header
	0x0017	Length of **Name** header (Unicode is 2 bytes per char)
	JUMAR.TXT	name of object, null terminated Unicode
	0xC3	HI for **Length** header
	0x00001000	Length of object is 4K bytes
	0x48	HI for Object **Body** chunk header
	0x0403	Length of **Body** header (1K) plus HI and header length
	0x.....	1K bytes of body
Server Response:		
response code	0x90	CONTINUE, Final bit set
	0x0003	length of response packet
Client Request:		
opcode	0x02	**PUT**, Final bit not set
	0x0406	1030 bytes is length of packet
	0x48	HI for Object **Body** chunk
	0x0403	Length of **Body** header (1K) plus HI and header length
	0x.....	next 1K bytes of body
Server Response:		
response code	0x90	CONTINUE, Final bit set
	0x0003	length of response packet

Fuente: IrDA, Obex Exchange Protocol, versión 1.3. 2003, pagina 28.

Estos paquetes se mantienen durante la comunicación, para realizar el envío de todos los datos que se desean intercambiar.

1.3.3.4.3.2.2. Paquete get

Se usa este tipo de paquete para pedir contenidos al cliente, por parte del servidor, éste debe tener el siguiente formato:

Figura 10. Formato de un paquete get-request

Byte 0	Bytes 1 and 2	Byte 3
0x03 (0x83 when Final bit set)	Packet length	Sequence of headers starting with Name

Fuente: BLUETOOTH SPECIFICATION Version 1.1 page 426 of 1084, IrDA Interoperability, página 10.

El cliente debe contestar en el siguiente formato:

Figura 11. Formato de un paquete get-response

Byte 0	Bytes 1 and 2	Byte 3
Response code	Response packet length	Optional response headers

Fuente: BLUETOOTH SPECIFICATION Version 1.1 page 426 of 1084, IrDA Interoperability, página 9.

Una vez se haya enviados todos los paquetes de información, se procede a terminar la comunicación con el envío de un paquete de desconexión, definido anteriormente.

1.3.4. Servicios

Los servicios son programas de computadora o software que permiten o proveen el desempeño de una tarea en particular sin mezclarse con otra. Son muy específicos en el desarrollo de las tareas para cumplir un objetivo, es un servicio el que permite subir un contenido web a un servidor web así como es

un servicio el que permite verificar el correo en cualquiera de los dominios sobre el cual se posee. Uno de los servicios que los dispositivos de Bluetooth proporcionan en su mayoría y que será de mucha utilidad cuando se hace conexión inalámbrica entre dos dispositivos para intercambiar información, es el servicio de colocación de objetos, conocido como Opp. A continuación se describe este servicio.

1.3.4.1. OPP

Definido de esta forma, por sus siglas en ingles: *Object Push Profile*, que traducido es: Perfil de Colocación de Objetos. Este servicio, es prestado por la mayoría de dispositivos de Bluetooth, permite que un cliente pueda conectarse a otro dispositivo sin tener que negociar un *password* o palabra de autenticación de identidad.

La mayoría de veces en una comunicación entre dos dispositivos, inicialmente se negocian los parámetros de comunicación. Parámetros tales como: tipo de comunicación, velocidad de transmisión de datos, tiempos de espera, entre oros. Uno de estos parámetros generalmente antes de iniciar la comunicación es: la contraseña o palabra clave, la cual permite la autenticación entre varios entes involucrados en un intercambio de información.

Sin embargo este proceso a veces se vuelve tedioso, dada la cantidad de veces que se inicia una comunicación entre dos dispositivos, es entonces donde OPP hace su aparición.

Generalmente al proceso de autenticación de usuarios mediante la negociación de una palabra clave o contraseña se le conoce como *Pairing***.**

Destinado a ser un servicio que no genere mas tareas que las que resuelve, OPP está encaminado a tener un dispositivo como “dispositivo de confianza”, es decir cuando un dispositivo A quiera conectarse con un dispositivo B para hacer un intercambio de información o contenidos, el dispositivo A simplemente debe enviar el contenido o información que desee al dispositivo B, en este caso el dispositivo B solo recibirá un mensaje de confirmación de que el dispositivo A desea enviarle información al dispositivo B y preguntando si desea recibir el contenido, en este caso el dispositivo B puede aceptar o denegar la conexión y de esta forma, iniciar o cancelar el envío de la información que el dispositivo A deseaba hacer.

Como se puede ver en este caso no fue necesario que el dispositivo A negociara una contraseña con el dispositivo B para que se autenticara la conexión entre ambos y así efectuar el envió de información. Esto fue proporcionado por el servicio de OPP.

Entonces el servicio de OPP del dispositivo B le pudo haber permitido –en caso de aceptación de la conexión- iniciar la recepción de información por parte del dispositivo A, sin tener que escribir una palabra de conexión para autenticación de los usuarios.

Un punto interesante de la funcionalidad proporcionada por OPP es que permite evitar la negociación de contraseña haciendo transparente la comunicación entre dispositivos de confianza. Por ejemplo: suponiendo el escenario en el que el propietario del dispositivo A se encontraba en una habitación vecina a la habitación de donde se encontraba el dispositivo B, sin embargo el propietario del dispositivo B se encontraba en una reunión importante y no podía ser interrumpido, además el propietario de A debía enviarle esta información al propietario de B dado que era información de suma importancia para lo que se estaba discutiendo en la reunión del propietario del dispositivo B, una pregunta podría ser ¿Cómo compartir esta información tomando en cuenta las

restricciones del escenario? Se puede ver que en la reunión el propietario de B no podía ser interrumpido, por lo tanto el propietario de A no podría decirle la palabra clave de conexión entre ambos dispositivos, haciendo mucho más difícil el envió de la información, y quizás hasta permitiendo que la comunicación fuese fallida debido a que nunca se pudo generar la autenticación de los dos dispositivos de comunicación.

Este escenario puede perfectamente ser resuelto por OPP, tomando en cuenta que B solo tendría que responder si acepta o no la información que A esta enviando y la comunicación se genera, permitiendo que A cumpla su envío y B la recepción y el objetivo general de compartir la información está completo.

1.3.5. Controladores

Un controlador es un componente de software que se encarga de traducir las instrucciones del procesador hacia instrucciones propias del dispositivo a bajo nivel. Cada vez que un programa o aplicación cliente esté usando un dispositivo de hardware, existen operaciones a bajo nivel que necesitan ser hechas; la aplicación cliente le envía las instrucciones, a su vez el procesador se encarga de usar el controlador de cada dispositivo para ordenarle al dispositivo de hardware a que realice cierta tarea, esto basado en que los controladores son los intermediarios entre una aplicación cliente y el dispositivo de hardware.

1.3.5.1. Blue Z o bluez

Este controlador es el más usado para las aplicaciones que usan Bluetooth, en un sistema operativo como Linux. Este conjunto de utilerías, proporcionan una conectividad rápida entre las computadoras convencionales y los dispositivos de Bluetooth en sistemas como los unix/based, su mayor objetivo es

proporcionar un conjunto de estándares de implementación de dispositivos inalámbricos de Bluetooth sobre Linux, soportando todos los protocolos y capas que pueden existir en las comunicaciones de Bluetooth.

1.4. Definición técnica de la solución

1.4.1. Requisitos del sistema

Parte importante para que una solución de software pueda ser planteada son los requisitos del sistema, los cuales deben adaptarse a lo que estamos pensando como solución, a continuación presento los principales requisitos del mismo.

1.4.1.1. Software

En cuestión de Software se definen los siguientes requisitos desde un punto de abstracción relativamente alto, a fin de proporcionar un ambiente con mejor rendimiento.

1.4.1.1.1. Ambiente integrado de desarrollo: IDE

El IDE, por sus siglas en inglés: *Integrated Development Environment*, es la herramienta o programa de desarrollo que permite la escritura, de programas de computadora, en un ambiente mucho más cómodo para el programador. También es conocido como: Ambiente Integrado de Diseño. Para este caso se usa un IDE que es fácilmente integrable con las librerías del controlador de Bluetooth. Este IDE es: Eclipse.

1.4.1.1.1.1. ¿Por qué Eclipse Ganimedes?

Porque es el IDE que mejor se adapta a la programación sobre los lenguajes que se usan en la construcción de la solución. Además es fácilmente integrable con el Sistema Operativo que se usa y proporciona un ambiente de trabajo conveniente para el programador.

1.4.1.1.2. Lenguaje

Para este tipo de soluciones basadas en software se necesita un lenguaje que tenga capacidad de ser altamente efectivo y poderoso, esto tomando en cuenta el soporte que provee así también que nos dé la libertad de poder manipular los dispositivos en su bajo nivel programático. Los controladores permiten usar sus librerías de desarrollo para que puedan, los dispositivos, ser manipulados usando las mismas, mediante instrucciones previamente establecidas en los mismos.

1.4.1.1.2.1. ¿Por qué C/C++?

En este caso se decide usar los lenguajes C y C++ debido a que son lenguajes que poseen un alto soporte, sus librerías son fácilmente adaptables a un sistema operativo, además los controladores (ver 1.3.5 Controladores) de los dispositivos de Bluetooth, en este caso *bluez*, permiten usar el potencial en su bajo nivel de los dispositivos, usando lenguajes C/C++. Esto permite que el programador o desarrollador del software pueda programar usando el mas bajo nivel permitido por los controladores a fin de tener la libertad de manipular en la forma optimizada las instrucciones de cada dispositivo de antena de Bluetooth.

1.4.1.1.3. Sistema operativo

El sistema operativo de una computadora es el conjunto de programas de computadora o conjunto de paquetes de software que permiten la manipulación de los recursos que la computadora posee, estos también se encargan de controlar y manejar los componentes de las computadoras así como permitir gestionar tareas a un usuario que desea hacer uso del equipo.

1.4.1.1.3.1. ¿Por qué Linux?

Para este caso se usa Linux, tomando en cuenta que es un sistema que proporciona libertad al programador para utilizar los recursos de manera optima

en cada programa. Además de flexibilidad en sus procesos para acceso a los registros que proporciona el sistema operativo, para lectura de la información o datos provistos por los distintos dispositivos conectados al computador. El sistema operativo, en este caso, debe poseer acceso rápido a los dispositivos, además de hacer reconocimiento automático de cada dispositivo que sea conectado. También debe permitir que el software, que controla los dispositivos de Bluetooth, pueda tener acceso a los registros que proporciona el sistema operativo y que son llenos por la información devuelta por cada dispositivo, para que pueda ser leída esta información y luego manipulada de la forma en la que se desee para cumplir el objetivo de la solución.

Para esta solución se propone el uso del sistema operativo Linux en su distribución *Centos* usando su versión 5.1.3.

1.4.1.2. Hardware

1.4.1.2.1. Antenas bluetooth

Las antenas bluetooth son dispositivos que usan la tecnología Bluetooth para establecer la conexión entre un dispositivo y otro, las mismas poseen terminales como una computadora para poder tener acceso a ellas y hacer ordenes directamente sobre las mismas. Estas son las encargadas de hacer las búsquedas de dispositivos que tengan a su alcance con el fin de iniciar un intercambio de información.

1.4.1.2.2. Enrutador de antenas bluetooth (opcional)

Son dispositivos que permiten agrupar un conjunto de antenas bluetooth para poder interactuar con ellas, esto se hace dado que la mayoría de antenas poseen puertos de conexión del tipo USB y estas, en la mayoría de casos, son conectadas directamente al ordenador. Se puede ver que en el escenario de

que existan 10 antenas Bluetooth para interactuar con ellas, se necesitaría 10 conexiones de tipo USB hacia la computadora para poder interactuar con el conjunto de las 10 antenas de Bluetooth, esto proporciona una restricción en cuestión de número de puertos disponibles.

Para esto los enrutadores de tecnología Bluetooth se encargan de agrupar un conjunto de cadenas como un dispositivo externo a la computadora y luego permitir conectar las antenas bluetooth hacia estos dispositivos, una vez hecho esto, los dispositivos (enrutadores) son conectados hacia la computadora, esto permite que pueda existir una gran cantidad de antenas conectadas a la computadora usando un puerto de conexión en común. Así podemos tener 12 antenas bluetooth conectadas a una computadora usando tan solo 3 conexiones de USB entre los dispositivos y la computadora, esto, pensando en el caso de que un enrutador soporte 4 antenas conectadas al mismo.

Para esta solución este tipo de enrutadores son opcionales dado que pueden estar o no, esto depende de la cantidad de antenas con las que se desea interactuar y la cantidad de conexiones de tipo USB con las que una computadora pueda contar.

1.4.2. Definición de los subsistemas

Con el fin de tener un mejor aprendizaje y una mejor concepción de la solución de software que se va a construir, es necesario el concepto de cómo el sistema trabajará y las relaciones entre componentes. Para esto se define el proceso del sistema, con el objetivo de entender luego la arquitectura que se propone como parte del mismo.

1.4.2.1. Proceso del sistema

El proceso del sistema se puede reducir a los siguientes pasos:

a. Carga de los parámetros de configuración para el correcto funcionamiento del sistema
b. Inicio de log o archivos de registros, bases de datos y módulos necesarios para la carga del sistema
c. Preparación de las antenas.
d. Escaneo para identificar las antenas que se tienen disponible para el correcto funcionamiento del software
e. Escaneo para encontrar los dispositivos que se tienen a la par
    - Obtener información del dispositivo como:
        - Nombre del dispositivo
        - Dirección MAC (MAC ADDRESS)
        - Posee o no el servicio de OPP
            - Si posee OPP: numero de puerto de conexión
            - No posee OPP: se termina la comunicación.
        - Otros datos deseados.
    - Inserción a la base de datos para llevar un control de los dispositivos encontrados en el escaneo.
f. Obtener de la base de datos los dispositivos que fueron encontrados
g. Filtrar los dispositivos que poseen el servicio de OPP para iniciar la negociación de envío de contenidos.
h. Iniciar el proceso de envío de contenidos.
    - Se obtiene la dirección MAC del dispositivo
    - Se procede a negociar la comunicación enviando el contenido hacia el dispositivo previamente seleccionado, preguntando si acepta la comunicación
        - Si la respuesta es SI
            - Se procede a enviar los contenidos
                - Separación en paquetes de tamaños adecuados para ser enviados

- Se envían los paquetes
- Mientras no se termine el envío de los paquetes de un contenido, se sigue hasta terminar.

- Si la respuesta es NO:
  - Se procede a terminar la negociación con el dispositivo que denegó la comunicación.

- Independientemente de que el dispositivo destino, acepte o deniegue la comunicación, se inserta a la base de datos, lo registros de las operaciones de los dispositivos que aceptaron la comunicación y los que no. Esto también sucede en log del sistema para mejor control.

i. Repetir proceso hasta terminar el número dispositivos que el escaneo del área permitió tener registrados en la base de datos.
j. Fin

En este proceso de pueden identificar dos tareas macro, las cuales son el objetivo de dos principales componentes de la solución; cada uno posee una. A su vez debe existir una relación entre las dos tareas, la misma relación que debe existir entre los dos componentes de software principales de la solución, estos componentes son descritos más adelante en este documento.

1.4.2.2. Arquitectura de la solución

La arquitectura de la solución es la infraestructura sobre la cual la solución estará montada, esto es los niveles de interacción de cada componente en su correcto funcionamiento. Generalmente la arquitectura de una solución es la columna vertebral sobre la cual se van construyendo las relaciones, interfaces, componentes, entre otros. Esta proporciona un perfil de mayor visibilidad para

poder representar lo que la solución será cuando esté construida completamente. En este caso para la arquitectura de esta solución, se consideran factores importantes como:

- Acceso a datos
- Disponibilidad
- Comunicación asíncrona
- Mayor rendimiento.

Mas adelante en este documento se detalla la forma que la arquitectura de esta solución tendrá, argumentando la razón por la que se construye de esta forma.

1.4.2.3. Separación del sistema en módulos

“Divide y vencerás” es una premisa que ejemplifica la ventaja que se proporciona un proceso a ser desglosado en sus componentes con el fin de poder trabajar por separado en cada uno de ellos y luego integrarlos a manera de obtener un producto u objeto completo, construido a base de sus partes o subcomponentes con el objetivo de una mejor comprensión y mejor distribución del trabajo.

La idea de la separación es iniciar a ver la solución mediante la perspectiva sistémica, (todo es un sistema) de esta forma podemos definir los sistemas macro y luego los subsistemas, todos interactuando para desarrollar las tareas con el fin de cumplir con el objetivo. El uso de la abstracción es parte importante de este proceso ya que ayuda a ver las particularidades de ciertas tareas a fin de agruparlas y asignarles fronteras para evitar la no delimitación de las funcionalidades correspondientes para cada uno de los módulos pensados para esta solución. Tomando en cuenta también que un sistema es mucho mejor

como un todo, comparado con la suma de sus subsistemas de manera individual se busca la integración de cada subsistema de manera optima para su correcto funcionamiento.

A continuación, se presentan los principales módulos pensados en el proceso que se esta solucionando y separados de esta forma, debido a la funcionalidad o tipo de tarea que cada modulo ejercerá dentro del sistema completo.

1.4.2.3.1. Scanner

Componente de software a nivel superior que permite hacer la búsqueda de dispositivos que se tienen alrededor a las antenas de comunicación, con el objetivo de iniciar una comunicación. Este también permite la búsqueda de servicios por dispositivo además de comprobar términos de negociación, por ejemplo puertos de conexión, velocidad de conexión así también cierta información del dispositivo al cual se esta haciendo la conexión.

Este componente a su vez puede ser dividido en pequeños subsistemas los cuales contribuyen al desarrollo de las tareas macro, generando entradas y salidas unos con otros.

1.4.2.3.1.1. Simple Scanner

Componente de software que hace un escaneo del área en busca de dispositivos al alcance y los almacena en registros que luego permitirán ser usados para la negociación y envío de información. Existen básicamente las siguientes tareas:

- Obtener Antenas disponibles para el proceso:
  Para poder trabajar correctamente la solución debe poseer antenas propias para poder hacer la comunicación entre dispositivos, para esto existe un proceso propio de este componente que se encarga de hacer

una búsqueda de antenas disponibles para el uso del sistema. Esto se hace pensando en que el sistema debe ser lo suficientemente inteligente, como para saber optimizar los procesos basado en los recursos que tiene a su disposición, de esta forma se puede saber cuantas antenas están conectadas y usar el mejor algoritmo que sea preciso en el mejor uso de estos recursos.

- Obtener dispositivos de ubicación cercana:
  Esta funcionalidad al igual que la anterior, permite encontrar dispositivos en las cercanías, con la diferencia de que esta funcionalidad permite encontrar únicamente aquellos dispositivos a los cuales nos vamos a conectar para hacer el envío de información o contenidos. Esto se hace tomando en cuenta que una antena del sistema no puede ser tomada como un dispositivo para envío de contenidos dado que si esto sucediera, no se estaría cumpliendo con el objetivo de la solución o proyecto, enfocando los contenidos a los clientes. Existe un proceso que se encarga de seleccionar y usar una de las antenas disponibles, para generar una búsqueda de los dispositivos que se encuentren al alcance de la misma, generando, de esta forma, un arreglo de dispositivos encontrados que luego serán filtrados y consultados para el envío de la información que se desea.

- Obtener información del dispositivo:
  Una vez el proceso anterior haya encontrado los dispositivos se debe entablar la comunicación con el dispositivo y preguntar cierta información de importancia para que el proceso pueda ser completado con éxito. Particularmente este componente también se encarga de preguntar información referente a la dirección MAC del dispositivo así como su nombre, esto permite que los datos puedan ser almacenados usando el

componente “*Device*”, (descrito posteriormente) para luego ser procesados.

- Obtener respuesta de si, o no, existe el servicio de OPP:
  Junto con la tarea de obtención de información del dispositivo, es necesaria la obtención de los servicios que este dispositivo ofrece. Aquí es donde se consulta al dispositivo acerca del servicio de OPP. Para esto entablamos una comunicación con el dispositivo previamente encontrado en el área, y luego hacemos una consulta que nos permita saber si el dispositivo posee el servicio de OPP. Aquí se usan las los procesos de consulta de servicios que proporcionan los controladores, así también las estructuras proporcionadas por los mismos. Luego estas estructuras son manipuladas para obtener la información que es necesaria a fin de saber si existe el servicio que es deseado.

Para esto este componente hace uso de varios otros componentes que permiten que la tarea pueda ser completada. Algunos componentes son encargados de tareas precisas como: configuraciones, almacenamiento, obtención de recursos. Estos componentes son descritos en este documento.

1.4.2.3.1.2. Device

Este objeto o componentes, permite ser un componente de manejo de características de los dispositivos, la idea principal es manejar los dispositivos con sus características, de forma genérica, es decir, manejarlos todos como un grupo de dispositivos y es aquí donde Device entra en acción, permitiendo que un dispositivo que es encontrado pueda ser almacenado temporalmente como un objeto del tipo Device, nos provee una mejor forma para luego entender el dispositivo que se encontró. La idea principal es que los dispositivos puedan tener sus cualidades almacenadas como tales en un objeto que agrupe las

cualidades de cierto dispositivo y que luego cuando le sean preguntadas por ciertas cualidades, para ser usadas en otras tareas, el objeto pueda entregarlas intactas, sin previa manipulación y sin haber sido adulteradas por procesos alternos. Esto permite que un mejor acceso, mejor autenticidad de los datos y una mejor concepción de las características de los dispositivos, mediante formas de obtención únicas para cada cualidad o característica de los mismos.

1.4.2.3.2. Sender

Componente de Software que se encarga de la manipulación de los envíos de los contenidos, este componente es quien, luego de conocer mediante el scanner los dispositivos que tiene disponibles a su alrededor y poseer los contenidos a ser publicitados, se encarga de enviar los contenidos de publicidad a los dispositivos de bluetooth.

Inicia las comunicaciones para negociar y es éste componente quien le permite al usuario ejercer su opinión de si desea recibir el contenido de publicidad, si su respuesta es aceptar, se prepara (este componente) para el envío de los paquetes de los contenidos y luego de que el mismo finalice, es éste componente quien también finaliza la comunicación y regresa a sus tareas de envío de contenidos con otros dispositivos. Este componente macro posee mas subcomponentes que le ayudan a gestionar su tarea principal, estos componentes son presentados a continuación.

1.4.2.3.2.1. Negotiator

El Negotiator o negociador, por su traducción al español, es un componente que se encarga de negociar las condiciones de la comunicación entre el servidor y el cliente, para esto este componente toma las direccionas MAC que previamente han sido recogidas por el Scanner e intenta comunicarse con cada una de ellas, con cada dispositivo al que corresponde cada dirección MAC

almacenada previamente en la base de datos, el Negotiator envía paquetes de descubrimiento para obtener información de cada dispositivo e intenta realizar un convenio entre la forma de envío de la información. Este componente define toda la paquetería Obex usada en la comunicación.

1.4.2.3.2.2. Simple Sender

Este componente de software es el encargado de enviar los contenidos hacia el dispositivo con el rol del servidor, en este caso el dispositivo que posee una persona. Es éste componente quien, luego de que el componente Negotiator le notifica que tiene permiso para enviar los contenidos, inicia el envío de contenidos tomando en cuenta los parámetros que el Negotiator le ha concedido en su tarea.

Este componente es quien toma cada paquete, en los que los contenidos se han distribuido, y los intenta enviar, este componente también se encarga de ver si un paquete no ha sido entregado correctamente a su destinatario y en su defecto, se prepara para reenviarlo a fin de hacer que las entregas sean completadas con éxito.

1.4.2.3.2.3. Packet

Este componente es uno de los encargados de agrupar a los distintos campos y cabeceras de un paquete común de IrObex. Conceptualmente se sabe que un paquete en todo protocolo de comunicaciones es equivalente a un arreglo de *Bytes*. Es decir es una cadena o agrupación de *bytes* que llevan un determinado orden y una estructura (esto es determinado por el protocolo). Adicionalmente estos *bytes* llevan en su interior los valores que el usuario desea establecer. Estos valores son los que luego son extraídos y leídos para saber las condiciones o sentencias que un paquete desea entregar. Los mensajes en sus cabeceras son bytes de ciertos valores en números

hexadecimales que representan ciertas operaciones o palabras claves. De igual forma el cuerpo de un mensaje es una secuencia de bytes que dan la información que se está enviando a través del paquete.

Este componente es quien agrupa a todos los campos de tipo byte. Estos campos son estandarizados como un *Header* o cabecera. A su vez, la cabecera o *header* es definida a continuación como una clase o componente que puede almacenar datos para luego conformar, mediante un grupo de cabeceras o bytes, un paquete completo del tipo que el usuario desee.

1.4.2.3.3. Dbmanager

Para el manejo de los registros que se deben llevar, acerca de dispositivos que son encontrados, dispositivos que no aceptan la publicidad, dispositivos que no tienen servicio de OPP, entre otros, se necesita que exista un manejador de estos registros, es por esto que debe existir un componente que se encargue de manejar estos registros, para este caso, el manejador de las distintas tareas de la administración de la base de datos, es el componente denominado Dbmanager, el cual posee todas estas tareas. Su objeto principal Dbaction, es quien se encarga de monitorear que los procesos de consulta e inserción a la base de datos deban ser como se especifica en sus reglas. (Ver 1.4.2.4.1 Scanner) para esto hace uso de varios procesos que aseguran la efectividad en el cumplimiento de sus tareas.

1.4.2.3.3.1. Dbaction

Este componente, es el principal del modulo de manejo de la base de datos, dentro de sus principales funcione están:

- Ingresar los datos de un nuevo dispositivo encontrado.

- Actualizar los datos de un dispositivo que ha sido previamente encontrado
- Consultar datos de un dispositivo en específico.
- Asegurar la correcta manipulación de los datos de los dispositivos.

A su vez, este componente, es quien dirige todos los procesos de persistencia de datos en la base de datos. En los siguientes capítulos se muestra como configurar las conexiones desde el ambiente de desarrollo.

1.4.2.4. Justificación de los módulos

Una característica que es esencial en cada solución de software, que se propone, es la Abstracción. Esta característica es la que permite que un objeto o componente de software pueda desarrollar sus tareas y que los demás componentes con los que interactúa, no sepan todo el proceso que el desempeño de las mismas conlleva.

Es por esto que a continuación se propone una razón simple pero eficaz, del porque se plantean los componentes que se han planteado anteriormente, esto obedeciendo a la transparencia que propone la abstracción. Fomentando la segmentación y delimitación de las tareas.

A continuación, se argumenta sobre la creación de los componentes y se responde a las preguntas de la creación de los subcomponentes propios de cada componente macro de la solución. Esto se hace con el fin de ver con justificación conceptual lo que cada subcomponente hace a grandes rasgos.

1.4.2.4.1. Scanner

Se necesita un componente que pueda coordinar las tareas que se le encargan, esto tomando en cuenta que, para que las mismas puedan ser desarrolladas, se

usará otros componentes o subsistemas que permitan que las tareas puedan ser terminadas y luego los resultados que las mismas produzcan puedan ser entregados al coordinador.

1.4.2.4.1.1. ¿Por qué Simple Scanner?

Porque se necesita un objeto o componente que tenga una responsabilidad única: devolver el conjunto de dispositivos que se encuentran alrededor, cerca para poder recibir (en caso acepten) los contenidos de publicidad que desean ser enviados.

1.4.2.4.1.2. ¿Por qué Device?

Porque se necesita que exista un componente de software que estandarice las características que poseen en general todos los dispositivos, además se necesita que este componente pueda devolver cualidades únicas de cada dispositivo sin tener que saber el proceso por medio del cual se está obteniendo los datos.

Este componente funciona como un formato para los datos que cada dispositivo posee. Obedeciendo a la abstracción, este componente posee métodos que permiten la devolución de las cualidades o datos de cada dispositivo, a los demás componentes, de forma transparente con el fin de que se entreguen en el formato correcto.

1.4.2.4.2. Sender

Se necesita este componente macro porque debe existir un coordinador que permita hacer las tareas de envío de componentes, pero, se puede ver que para un envío no se necesita de solo enviar la información sino también negociar las formas o convenciones por las cuales se van a realizar las tareas de envío de información.

1.4.2.4.2.1. ¿Por qué Negotiator?

Porque se necesita un componente que permita definir el estándar de paquetería IrObex que se estará usando en la comunicación. Este componente es quien descubre, mediante paquetes *connect-request*, los dispositivos cercanos disponibles que desean recibir contenidos de publicidad.

1.4.2.4.2.2. ¿Por qué Sender?

Porque debe existir un componente que se encargue de desarrollar únicamente la tarea de envío de información, preguntando previamente por los parámetros de conveniencia negociados con el destinatario y luego ajustarse a ellos y enviar la información, así también este componente se necesita, para asegurar que la tarea de envío de información sea desarrollada con éxito.

1.4.2.4.2.3. ¿Por qué Packet?

Porque se necesita de un componente que permita agrupar muchas cabeceras de manera que se vea un paquete como una secuencia de bytes o cabeceras, las cuales tienen una estructura dada por Obex.

1.4.2.4.3. Dbmanager

Este subcomponente nace de la necesidad de tener un objeto de software que permita el manejo de los registros a la base de datos y sus parámetros de configuración a fin de transparentar los procesos particulares de cada componente.

1.4.2.4.3.1. ¿Por qué Dbaction?

Porque se necesita un objeto o componente de software que se encargue de manejar los procesos de la base de datos, esto para que cada componente sea encargado de tareas específicas y poder separar de esta forma, las tareas de cada componente.

1.4.2.4.4. Interface entre los módulos

En esta sección se conceptualiza la relaciones entre los dos grandes módulos de la solución, para este caso se procede a describir la relación entre el Scanner y el Sender, los dos componentes mayoritarios de la solución de software que se plantea.

1.4.2.4.4.1. Interface: Scanner-Sender

Esta relación se ve intrínsecamente establecida entre ambos, dado que se necesita de dos operaciones macro en el algoritmo de la solución, estas operaciones son las siguientes:

- Primera fase: se debe, de alguna forma, saber o conocer de los dispositivos que se encuentran a nuestro alrededor y que están disponibles para ser contactados para envío de publicidad.

- Segunda fase: una vez se tengan los dispositivos que en la primera fase fueron descubiertos, se debe iniciar el contacto con cada uno de los seleccionados a fin de poder enviar los contenidos de publicidad, así que en esta fase es donde se hacen estas operaciones de envío.

Es importante hacer notar que uno de los componentes, podría imaginar el lector, podrá ser una aplicación que se mantiene en el servidor de contenidos para poder manipular las actividades de publicidad; es decir relacionadas al manejo de los contenidos y control de envíos, entre otros. Sin embargo este componente esta fuera de las fronteras del alcance de esta solución, esta podría ser una implementación que cualquiera de los lectores de este trabajo podría hacer.

La relación es, entonces, evidente. Tomando en cuenta que a fin de que la segunda fase se pueda efectuarse, debe existir la primera fase como requisito obligatorio. Esto obedece a que simplemente las salidas que proporciona el proceso de la primera fase son las entradas que necesita el proceso de la segunda fase. Es importante notar que esta interface esta representada por el manejador de la base de datos que proporciona la función de una cola de mensajes. En esta solución de software se usará la base de datos **Postgres** usando las librerías de conectividad entre C++ y Postgres conocida como *pqxx*.

Se puede ver en este punto que la perspectiva sistémica se cumple –planteando que un sistema es mucho más como un sistema completo y no como la suma de todas sus partes individuales-. Esto hace que los dos componentes macros puedan encajar en una solución de software y que ambos puedan trabajar en conjunto para cumplir el objetivo de la solución.

2. MARCO PRÁCTICO

2.2. Análisis, diseño y desarrollo de la solución

2.2.1. Diseño de la arquitectura

La arquitectura de esta solución está basada en las principales características de calidad que debe tener una arquitectura; esto, tomando en cuenta todo lo que se planteó en el marco teórico, respecto a lo que el sistema de software debe contener para brindar un alto rendimiento. En este caso los más importantes son:

- Comunicación síncrona: esta comunicación permite una línea directa entre componentes.
- Rendimiento: esta característica se basó en la forma en la que un servidor realiza su trabajo. Para este caso la solución planteada se dispone a realizar únicamente la tarea asignada, (escanear-enviar contenido), la cual se establece en el servidor.
- Disponibilidad: esto permite que el servicio siempre esté arriba. Esto sin importar si un componente está caído o tiene alguna falla.
- Tiempo de respuesta: este tiempo debe ser relativamente corto. Es por esto que es un aspecto importante en la arquitectura.

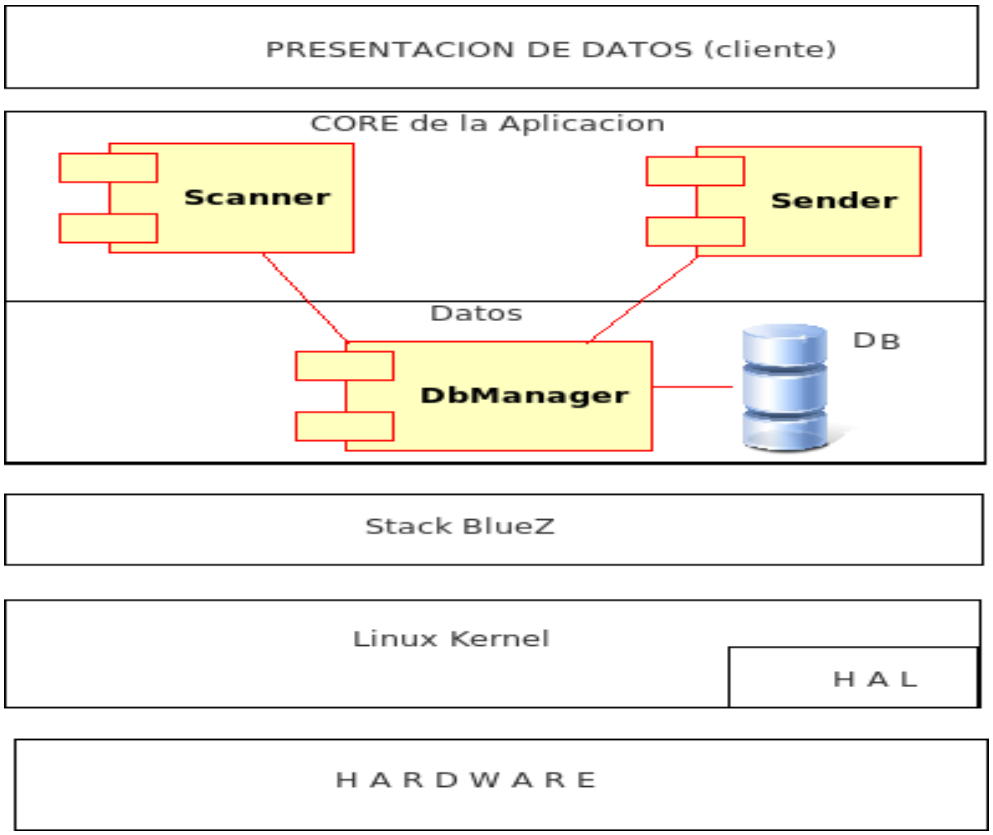
Luego de analizar todos estos factores de calidad, se puede ver que la arquitectura debe ser capaz de contener todos éstos. Para ello se propone la arquitectura de 2-Capas. Sin embargo, para este caso la comunicación será síncrona solo entre los componentes que tienen relación. En su defecto, los componentes usarán al componente manejador de la base de datos como conexión. La base de datos será alimentada por el Scanner y consumida por el Sender. Esto proporciona comunicación asíncrona usando una arquitectura de 2-Capas. Esta es una modificación al uso de las arquitecturas de 2-Capas que

generalmente son usadas en comunicaciones síncronas con el uso de interfaces. Ahora con esta modificación, se puede tener las funcionalidades deseadas.

2.2.1.1. Arquitectura

Con base en las condiciones expuestas anteriormente, se crea el diagrama del diseño de la arquitectura. Éste cumple con todas las especificaciones para la solución. Es necesario hacer saber que, en esta solución, se propone el modelo de datos, más conocido como **Núcleo de la aplicación**.

Figura 12. Arquitectura de la solución

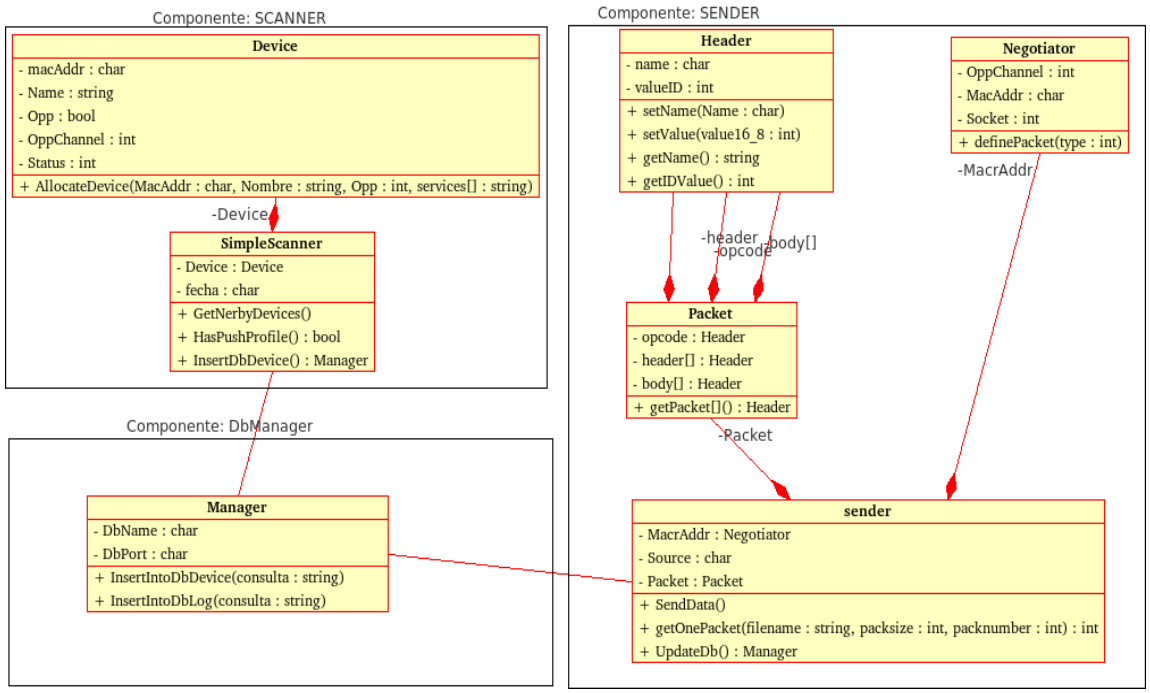


Fuente: Ronmell Fuentes.

2.2.1.2. Diagrama de clases

El diagrama de clases ayuda a comprender de una manera más detallada la forma en la que interactúan los componentes de la arquitectura.

Figura 13. Diagrama de clases de la solución



Fuente: Ronmell Fuentes

2.2.2. Diseño de componentes

2.2.2.1. Scanner

2.2.2.1.1. Simple Scanner

Este componente posee muchas de las características propias del Scanner, entre otras las funciones que posee son:

- GetNerbyDevices: reconocer cada dispositivo que se encuentra cerca de las antenas, en el rango aceptado y luego formar una lista de cada uno.

- HasPushProfile: luego de extraer la lista de los dispositivos que fueron encontrados, mediante el reconocimiento de dispositivos cercanos, se toma cada uno de los dispositivos y se pregunta por el servicio de Opp para verificar cuales dispositivos lo poseen y cuales no.
- InsertDbDevice: comunicarse con el componente manejador de la base de datos para que guarde todos los registros de los dispositivos encontrados con sus cualidades.

A continuación, se describen cada una de estas funciones. Para esto cada función hace uso de funciones y métodos propios de la librería de utilidades que brinda bluetooth para el desarrollo de aplicaciones sobre dispositivos de tecnología de Bluetooth.

2.2.2.1.1.1. Funciones

Todas las funciones descritas a continuación fueron codificadas en C/C++ usando los métodos y librerías que ofrece el controlador de Bluetooth y tomando como referencia la información provista por el libro "*Bluetooth Essentials for Programmers*" de Albert Huang y Larry Rudolph.

2.2.2.1.1.1.1. GetNerbyDevices

Con esta función se obtienen todos los dispositivos que se encuentran a nuestro alrededor. Esta función retorna un arreglo de espacios de memoria donde cada espacio es un registro de un dispositivo que fue encontrado. Para esto, el tipo de función es un tipo arreglo de los provistos por C++ a su vez usando el tipo Device el cual es el componente de alojamiento de esta información. A continuación, se describe el segmento de código que hace esto.

Previo a escribir el código de la función, se debe hacer las llamadas a las cabeceras respectivas para la clase en la que se esta implementando la función. Estas cabeceras son:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/socket.h>
#include <bluetooth/bluetooth.h>
#include <bluetooth/hci.h>
#include <bluetooth/hci_lib.h>
```

El código que permite obtener los registros de los dispositivos es el siguiente:

```
vector<Device> GetNerbyDevices()
{
   vector miVector; //vector de devolución.
   inquiry_info *dispositivos = NULL;  //apuntador de
                                       //memoria a dispositivos.
   int max_resp, num_resp; // numero: max de respuestas
                           //y respuestas obtenidas.
   int adapter_id, sock, flags, len; //socket, adaptador,
                                     //banderas, longitud.
   int i; //apuntador de espacios en arreglo.
   char direc[19] = {0}; //dirección de conexión
   char nombre[248] = {0}; //nombre del dispositivo.

adapter_id = hci_get_route(NULL);
sock = hci_open_dev( adpter_id );
//manejando el error en la aperture del socket
```

```
if( adapter_id <0 || sock < 0) {
        perror("apertura del socket");
        return NULL;
}
len = 8;
max_resp = 255;
flags = IREQ_CACHE_FLUSH;
dispositivos = (inquiry_info *)malloc(
                        max_resp * sizeof(inquiry_info));
num_resp = hci_inquiry (adapter_id,len,max_resp,
                        NULL,&devices,flags);
if(num_resp<0)perror("error escaneando");
miVector = new Vector(num_resp);
for (i=0;i<num_resp;i++) {
        ba2str(&(devices+i)->bdaddr,direc);
        memset(nombre,0,sizeof(nombre));
        if(0!=hci_read_remote_name(sock,
&( dispositivos + I )->bdaddr,
 sizeof(nombre), nombre, 0)) {
   strcpy(nombre,"[desconocido]");
}
miDispositivo = new Device(name, addr);
miVector.push_back(myDevice);
        cout<<"direccion: "<<direc<<" nombre:  "<<nombre<<endl;

}
free( devices );
close( sock );
return myVector;  }
```

Este código hace un escáner sobre los dispositivos que se encuentran a la vecindad y que responden a la llamada del escáner. El código se explica a continuación:

Primero se crea una estructura de datos del tipo Vector, así como la clase que almacenará la información del dispositivo, este es Device. Esto se hace en las líneas:

vector miVector;

Se necesita un apuntador de memoria que pueda alojar al arreglo que contiene la información de todos los dispositivos que respondieron, para eso se usa: inquiry_info *dispositivos = NULL;

Apuntando hacia NULL, inicialmente para que este libre la memoria. Las líneas:

int max_resp, num_resp;
int adapter_id, sock, flags, len;
int i;

Son números enteros que almacenarán los valores de: número máximo de respuestas permitidas para ser almacenadas, número de respuestas que efectivamente respondieron al llamado, el identificador único de la antena sobre la cual se ejercerá la comunicación, el identificador o número de socket, las banderas de comunicación que recibe el socket, la longitud de la dirección y el apuntador de dirección para el vector, respectivamente.

Luego se crean los arreglos de para guardar los datos obtenidos, estos son: el nombre y la dirección MAC de cada dispositivo. Están inicializadas a 0 para evitar que exista data errónea o valores de variables sin inicializar.

char direc[19] = {0};
char nombre[248] = {0};

En la línea: *adapter_id = hci_get_route(NULL); e*l metodo *hci_get_route();* devuelve un entero, el cuál es el identificador de la antena de bluetooth sobre la cual se iniciará la comunicación, en este caso el parámetro *NULL* hace referencia a que no es una antena en específico. Esto provoca que el sistema busque la antena que este disponible y sea ésta la usada en la comunicación. Se puede cambiar este parámetro por la dirección MAC de una antena en específico, en caso se desee.

Luego de generar un identificador de la antena de conexión, se debe abrir el socket de comunicación. Este es un entero. El método *hci_open_dev();* recibe como parámetro el identificador de la antena sobre la que se iniciará la comunicación. En este caso es el obtenido anteriormente:

```
sock = hci_open_dev( adpter_id );
```

En caso de que no se pueda abrir el socket, se maneja el error:

```
if( adapter_id <0 || sock < 0) {
        perror("apertura del socket");
        return NULL;
}
```

Si no existe problema alguno en el socket, se establecen algunos valores de almacenamiento y comunicación, estos son: la longitud de las direcciones MAC, el número máximo de respuestas aceptadas, que son 255. Las banderas de la comunicación. Esto se hace con estas líneas de código:

```
len = 8;
max_resp = 255;
flags = IREQ_CACHE_FLUSH;
```

Se aloja, usando el apuntador *dispositivos,* memoria suficiente para la información de todos los dispositivos; a éste se le asigna una dirección de

memoria con la función *malloc,* enviándole como parámetro el tamaño de memoria para el alojamiento; en este caso es: el número máximo de respuestas permitidas multiplicado por el tamaño de la información del dispositivo. Así se tendrán 255 posiciones de memoria con el tamaño requerida por la información de cada dispositivo de esos 255.

*dispositivos = (inquiry_info *)malloc(max_resp * sizeof(inquiry_info));*

Ahora que se tiene los parámetros establecidos, se procede a realizar el escáner de los dispositivos, esto es:

num_resp = hci_inquiry (adapter_id,len,max_resp,NULL,
&dispositivos,flags);

Se obtiene el número de dispositivos que respondieron, en num_resp. Luego el método *hci_inquiry();* recibe como parámetros:

adapter_id: el identificador de la antena.
len: la longitud de información.
num_resp el número máximo de respuestas permitidas.

&dispositivos: la dirección de memoria que se reservó para el alojamiento de la información de los dispositivos.

flags: las banderas sobre las cuales se hace la conexión.

En caso de que exista un error en el socket, o exista falla en la conexión de los dos dispositivos cuando se hace el escaneo, la función *hci_inquiry* devuelve -1 por eso se maneja de la forma siguiente:

if(num_resp<0)perror("error escaneando");

Se le asigna memoria al vector de dispositivos, con el número de respuestas obtenidas:

```
miVector = new Vector(num_resp);
```

Ahora se procede a recorrer las direcciones de memoria donde se alojó la información de los dispositivos que respondieron, esto es:

```
for (i=0;i<num_resp;i++) {
   ba2str(&(devices+i)->bdaddr,direc);
   memset(nombre,0,sizeof(nombre));
   if(0!=hci_read_remote_name(sock, &( dispositivos + I )->bdaddr,
  sizeof(nombre), nombre, 0)) {
strcpy(nombre,"[desconocido]");
}
miDispositivo = new Device(name, addr);
miVector.push_back(myDevice);
cout<<"direccion: "<<direc<<" nombre:  "<<nombre<<endl;
}
```

Se usa el método *ba2str();* para convertir a String una dirección de dispositivo. Luego se copia con *memset();* el nombre del dispositivo. Esto es en la posición de memoria actual que nos proporciona el apuntador *i*. Si no existe un nombre se le asigna por default el nombre “desconocido”. Se genera un nuevo objeto con cada información del dispositivo y luego se agregan al vector de información mediante el método *push_back();* de la clase Vector. Por ultimo se imprime en consola la información de cada dispositivo obtenido.

Se libera el apuntador de memoria con: *free(devices);* y luego se cierra el socket de conexión: *close(sock);* posteriormente se regresa el arreglo de información de dispositivos: *return myVector;* y se regresa a la función principal o tarea de la aplicación.

2.2.2.1.1.1.2. HasPushProfile

Para saber cuáles dispositivos poseen el servicio de OPP, se debe hacer la comunicación con cada uno en particular, para esto se posee un arreglo de dispositivos que ha retornado el Scanner en su función de obtención de los dispositivos cercanos. El código que realiza la función de verificar por el servicio de OPP es el siguiente:

```
bool HasPushProfile(String MacAddr) {
//declarando algunas variables
bool myreturn=false;
uint16_t service_uuid_int=0x1105;
int status=-1;
bdaddr_t target;
uuid_t service_uuid;
sdp_list_t *response_list=NULL, *search_list, *attrid_list;
sdp_session_t *mysession=0;
//rango del servicio
uint16_t range = 0x0000FFFF;
str2ba(MacAddr.c_str(),&target);
char mac_antena_u[18];
String antenna=" 00:A0:A1:FF:02:01";
strcpy(mac_source_unr, antenna.c_str());
bdaddr_t mac_antenna;
str2ba(mac_antenna_u,&mac_antenna);
//generando la session de conexion
mysession = sdp_connect(&mac_antenna,
                        &target,SDP_RETRY_IF_BUSY);
sdp_uuid16_create(&service_uuid, service_uuid_int);
//generando las listas
```

```
search_list = sdp_list_append(0,&service_uuid);
attrid_list = sdp_list_append(0,&range);
//preguntando por el servicio
status = sdp_service_search_attr_req( mysession,
search_list,SDP_ATTR_REQ_RANGE,
attrid_list,&response_list);
        if(status==0){
                myreturn=true;
                cout<<"posee servicio de Opp"<<endl;
}
sdp_list_free(attrid_list,0);
sdp_list_free(search_list,0);
sdp_list_free(response_list,0);
sdp_close(mysession);
return myreturn;
}
```

Antes de iniciar con la explicación del código, se debe agregar las líneas siguientes, a la clase en la que se vaya a implementar la función anterior.

```
#include <stdio.h>
#include <stdlib.h>
#include <bluetooth/bluetooth.h>
#include <bluetooth/sdp.h>
#include <bluetooth/sdplib.h>
```

Primero se declaran las variables a ser usadas, la primera es el valor a retornar en nuestra función, esta es:

bool myreturn=false;

Se inicializa en false, y luego se actualiza en caso de respuesta de OPP encontrada en el dispositivo.

Luego se procede a declarar un entero de 16 bits para el identificador del servicio sobre el cual se ejecutará la consulta. Esto es:

uint16_t service_uuid_int=0x1105;

El número 0x1105 es el identificador para el servicio de OPP. (Ver Apéndice 3 - Numeros Asignados de Obex (Obex Numbers), de esta tesis). Luego se declaran otras variables:

int status=-1;
bdaddr_t target;
uuid_t service_uuid;

La primera variable se refiere al estado de la conexión. *target* se refiere a la dirección Mac del dispositivo al que se quiere conectar, ésta es del tipo bdaddr_t designado como un arreglo de caracteres para manejar este tipo de direcciones. *service_uuid* es del tipo uuid_t ya que es un tipo de dato especifico para el identificador del servicio, en este caso es distinto al arreglo de 16 bits del entero declarado a inicios del código.

Luego se crean los apuntadores a las listas para almacenamiento de los datos:

*sdp_list_t *response_list=NULL, *search_list, *attrid_list;*

Ahora se procede a declarar el apuntador para la sesión de la conexión:

*sdp_session_t *mysession=0;*

Se asigna un rango de la consulta, para este caso, se pone el rango de las direcciones que pueden oscilar entre valores altos para cubrir todo el rango de direcciones MAC, esto es: *uint16_t range = 0x0000FFFF;* y se procede a copiar

la dirección Mac del dispositivo a conectar, a una estructura mas conocida por el *stack* de *bluez*, enviándole como parámetro la dirección de memoria de la estructura, en este caso *target*:

str2ba(MacAddr.c_str(),&target);

A continuación, se crea un arreglo de caracteres. Luego se define la dirección origen o de la antena de conexión, se copian los valores a las estructuras y por ultimo se le da formato cómodo para el stack de blueZ:

```
char mac_antena_u[18];
String antenna=" 00:A0:A1:FF:02:01";
strcpy(mac_source_unr, antenna.c_str());
bdaddr_t mac_antenna;
str2ba(mac_antenna_u,&mac_antenna);
```

Ahora se procede a la generación de una sesión para la conexión.

```
mysession = sdp_connect(&mac_antenna,
        &target,SDP_RETRY_IF_BUSY);
```

El método *sdp_connect();* recibe como parámetros: la dirección de memoria de la antena de conexión, la dirección de memoria donde esta guardada la dirección del dispositivo objetivo y las banderas de conexión. Estas últimas pueden establecerse a 0 en caso de no conocer los distintos parámetros, para este caso se usa *SDP_RETRY_IF_BUSY.*

Se convierte el entero de 16 bits a una estructura acorde para ser manejada por los métodos del *stack* de *bluez*, para esto se envía, como primer parámetro, la dirección de memoria de la estructura que recibe. Como segundo parámetro envía el valor almacenado en el entero de 16 bits:

```
sdp_uuid16_create(&service_uuid, service_uuid_int);
```

Para muchos servicios no especificados, se puede usar el método *sdp_uuid128_create();* el cual recibe los mismos parámetros.

Ahora se procede a agregar las listas a la consulta:

```
search_list = sdp_list_append(0,&service_uuid);
attrid_list = sdp_list_append(0,&range);
```

Una vez hayan sido establecidos estos parámetros, se desarrolla la consulta.

```
status = sdp_service_search_attr_req( mysession,
search_list,SDP_ATTR_REQ_RANGE,
attrid_list,&response_list);
```

El método *sdp_service_search_attr_req();* recibe como parámetros: la sesión de conexión, la lista de búsqueda, el parámetro de búsqueda (*SDP_ATTR_REQ_RANGE*), la lista de atributos devuelta por la consulta y la dirección de memoria a la que apunta la lista en respuesta a la petición del servicio. Este método devuelve un valor entero, es 0 si la consulta fue realizada con éxito y -1 en caso contrario. Para esto se debe estar seguro que se han devuelto datos con la validación:

```
if(status==0){
   myreturn=true;
   cout<<"posee servicio de Opp"<<endl;
}
```

Esto actualiza el valor a retornar por el método a fin de notificar si el dispositivo posee o no, el servicio de OPP. Finalmente se liberan los apuntadores a las estructuras (listas) usadas para almacenar los datos:

```
sdp_list_free(attrid_list,0);
sdp_list_free(search_list,0);
```

sdp_list_free(response_list,0);

Luego se procede a terminar la sesión y se retorna el valor booleano:

sdp_close(mysession);
return myreturn;

En caso se desee imprimir el valor devuelto por la consulta de los servicios se puede usar método: *sdp_record_print(record);* el cual recibe una estructura del tipo *sdp_record_t *record;* y que puede ser obtenido, recorriendo la lista de respuesta y mediante la forma:

*record=(sdp_record_t *) response_list->data;*

Se puede ver que se debe castear los datos obtenidos para que sean del mismo tipo que la estructura del record, en este caso del tipo *sdp_record*.

2.2.2.1.1.1.3. InsertDbDevice

Esta función se encarga simplemente de hacer una llamada a la clase DbAction, la cual posee la función de inserción. Para hacer una llamada a esta clase se procede de la siguiente manera:

CDbAction *mng;
mng=new CDbAction();
mng->Insert(dev_info,opp,net);
delete mng;

Se puede ver que se crea un apuntador a un objeto de tipo DbAction. Luego se asigna memoria al objeto previamente creado, posteriormente se hace la llamada al método, esto es: *mng->Insert(dev_info);* donde dev_info, es un

objeto del tipo Device. Este posee las características de un dispositivo de bluetooth.

Como buena practica, se termina el objeto, para liberar la memoria. Esto se hace mediante la instrucción: *delete mng;* permitiendo que el objeto quede eliminado y la memoria limpia, lista para ser usada por otras aplicaciones o métodos de nuestra aplicación.

2.2.2.1.2. Device

Este componente simplemente maneja los dispositivos, es una clase de C++ que se encarga de dar alojamiento un dispositivo. Cada vez que se encuentra un dispositivo, cuando se hace escaneo de lo dispositivos en el área, se agrega a un arreglo de dispositivos, este arreglo recibe objetos del tipo Device. Este tipo de datos es manejado por este componente.

2.2.2.1.2.1. Funciones

A continuación, se presentan las funciones relacionadas al componente Device, estas funciones son llamadas por cada objeto, implementación de este componente, cada vez que se encuentra un dispositivo en el escaneo de rutinario.

2.2.2.1.2.1.1. AllocateDevice

Es la función principal del constructor de este objeto. Cuando se llama a la creación de un objeto de este tipo, simplemente se hace la llamada a esta función. Esta función se describe a continuación.

```
void AllocateDevice(string direccionMac,
string nombre, int opp, Vector servicios) {
str2ba(direccionMac.c_str() ,&macAddr);
this->name=nombre;
this->oppId=opp;
if(servicios.size>0) {
   this->services=servicios;
}
}
```

Los parámetros son los de un dispositivo, se piden: el nombre, la dirección MAC, el puerto del servicio OPP y un vector de servicios que posee este dispositivo.

Se convierten las direcciones a un formato que estandarizado que maneja el *stack* de *bluez*, esto es:

```
str2ba(direccionMac.c_str() ,&macAddr);
```

Donde str2ba es una función del *stack* de *bluez*, recibe los dos parámetros que son: el primero es el arreglo de caracteres que posee la dirección Mac del dispositivo y el segundo es la dirección de memoria a la que apunta la variable en la que hay que almacenar este valor transformado. En este caso **macAddr** es una variable del tipo de dato: **bdaddr_t.** El cual es un tipo de dato proporcionado por el *stack* de *bluez*.

Luego se inicializan los valores como el nombre, identificador del servicio de OPP y luego los servicios. Para el caso de los servicios se define un vector que posee el nombre y su identificación. Se verifica que el vector realmente tenga contenido para esto se hace la validación:

```
if (servicios.size>0) {
```

```
        this->services=servicios;
    }
```

Si se encuentra información, entonces se inicializa la variable *services* que es de tipo Vector. Sino se encuentra información entonces la variable esta vacía, es decir apunta hacia NULL.

Se puede ver que solo es un estándar de objetos. Es decir se realiza para evitar confusiones de datos y atributos de cada dispositivo.

2.2.2.2. Dbmanager

Este componente maneja la base de datos. En este caso se usan dos tipos de transacciones:

- La inserción de un dispositivo nuevo.
- La actualización de los datos de un dispositivo nuevo.

Se usa la inserción para registrar los dispositivos nuevos que se encuentran en cada escaneada que se realice. Se usa también la actualización para tener las condiciones actuales de los dispositivos. Es decir tener las características más actuales de los dispositivos.

2.2.2.2.1. Dbaction

Es la clase que permite el acceso a la base de datos. Es quien tiene las funciones de inserción de dispositivos.

2.2.2.2.1.1. Funciones

Las funciones relacionadas al objeto Dbaction, son las que se usan para el manejo de la base de datos, sin embargo estas funciones están especializadas

para las tablas o entidades existentes en la base de datos, tomando en cuenta los tipos de datos específicos para los dispositivos y su información.

2.2.2.2.1.1.1. InsertIntoDbDevice

Esta función es la encargada de insertar un dispositivo a la base de datos. En este caso se ejecuta el siguiente código:

```
bool InsertIntoDbDevices(string consulta) {
try {
        connection Conn("dbname=mibasededatos");
        work my_transaction(Conn,"T000001");
        my_transaction.exec(statement);
        my_transaction.commit();
        Conn.disconnect();
        return=true;
    }
    catch (const sql_error &e) {
        cerr << e.what() << endl;
        cout <<"unable to connect..."<<endl;
        return=false;
        }
}
```

Aquí se puede ver que existen los siguientes pasos:

- Se crea la conexión hacia la base de datos.
- Se crea una transacción.
- Se ejecuta la transacción.
- Se desconecta de la base de datos.

Antes de ejecutar este código se debe agregar las sentencias de agregación de archivos o librería esto es:

#include <iostream>

#include <pqxx/connection>

#include <pqxx/transaction>

#include <pqxx/result>

#include <stdexcept>

#include <libpq-fe.h>

La operación que se debe hacer es parecida a esta:

Insert into dispositivo (mac,nombre,opp,fecha,status)

values ('AD:DD:AA:AA:AA:AA' ,'JoseManuel' , 5 , now(), 1);

Tomando en cuenta que la tabla tiene 5 campos. Estos son:

- Mac: que es la dirección MAC del dispositivo.
- Nombre: nombre del dispositivo.
- OPP: canal del servicio de OPP.

- Fecha: fecha de escaneada, *now()* es la función que devuelve la fecha.
- Status: si esta activo o inactivo el dispositivo.

Una pregunta que viene a la mente es: ¿por qué no existe un *password* y un usuario? La respuesta es simple: porque Postgres es una base de datos que obtiene algunos de los parámetros de conexión desde variables de entorno del perfil del usuario en el que se está haciendo la conexión. En este caso existen parámetros básicos que se deben agregar al archivo del perfil del usuario del sistema operativo. Estos parámetros son:

- PGUSER.
- PGPASSWORD.
- PGDATABASE.
- PGHOST.

Estas variables son escritas en el perfil del usuario, generalmente se hace escribiendo en la consola de Linux lo siguiente:

$ gedit ~/.bash_profile

Esta instrucción debe mostrar un archivo que posee las variables del ambiente del usuario activo. Para agregar las variables se debe seguir este estándar:

PGUSER=usuario

export PGUSER

Esto se debe hacer con cada variable, el formato es: VARIABLE=valor export VARIABLE. Se debe escribir los valores reales de conexión de la base de datos para que se pueda hacer la conexión. Se guardan los cambios y se cierra el archivo. Luego se debe cerrar sesión y volverla a iniciar para que los cambios sean efectivos.

Para realizar la actualización de los datos, se debe llamar a la misma función, con la diferencia de que se debe cambiar la consulta, esta debe tener el estándar de instrucciones *update* que operan los manejadores de base de datos. En este caso refiérase al manual de Postgres.

2.2.2.2.1.1.2. InsertIntoDbLog

Se encarga de ingresar el *log* de los dispositivos que se han encontrado en cada escaneada. Sin embargo éste debe llevar un conteo de todos los previos. Para esto se hace necesaria una secuencia definida en la base de datos. Se crea la secuencia y luego se debe cambiar un parámetro en la consulta a realizar. La consulta debe parecer a la siguiente:

Insert into log_dispositivo (identificador, mac,nombre,opp,fecha,status)

values (' select nextval(sequencia) ', 'AD:DD:AA:AA:AA:AA' ,

'JoseManuel' , 5 , now(), 1);

Aquí se puede ver que el valor a enviar al identificador es una secuencia, para esto se usa la instrucción "*select*" que proporciona el manejador de base de datos, para obtener el nuevo valor al que esta apuntando la secuencia. Como parámetro la función *nextval()* recibe el nombre de una secuencia para realizar la consulta sobre ella y devolver el valor siguiente.

Luego de tener lista la consulta, simplemente se hace la llamada a la función **insertIntoDbDevices("consulta")** enviándole la consulta (*query*) de inserción de datos al *log*. Esto se hace para facilitar y reutilizar las funcionalidades provistas por otras funciones.

2.2.2.3. Sender

Este componente maneja la comunicación entre los dos dispositivos en el envío y recepción de los contenidos publicitarios e información de dispositivos.

2.2.2.3.1. Sender

Este componente es el encargado de manejar las siguientes funciones:

- Abrir la conexión con el servidor.
- Obtener la información a ser enviada.
- Dividir la información o archivos, a pequeños paquetes de acuerdo al tamaño especificado por el protocolo.
- Enviar los paquetes al servidor.
- Reenviar los paquetes, en caso de pérdida de alguno.
- Enviar un paquete en específico, en caso de perdida.
- Cerrar la conexión con el servidor.

Para esto, se ayuda de funciones que realizan tareas específicas. Estas tareas se describen a continuación.

2.2.2.3.1.1. Funciones

Cada una de las funciones descritas a continuación obtienen gran parte de los datos con los que trabajan a partir de la información que está contenida en la base de datos, es decir, dependen del trabajo que haya hecho antes el componente Scanner.

2.2.2.3.1.1.1. SendData

Esta función es la encargada de enviar la información. Esto lo hace tomando en cuenta los parámetros, antes, encontrados por el Scanner y establecidos por el

Negotiator. Para esto, se debe construir un socket de conexión (*Rfcomm*), enlazarlo a un servidor y luego enviar los datos, siempre verificando que los paquetes puedan llegar al destino correcto. El proceso es el siguiente:

- Se realiza la conexión del *Socket Rfcomm* propio (cliente) al dispositivo (servidor) que se desea.
    - Se envía un paquete connect-request.
    - Se recibe el paquete connect-response.
- Se envía el primer paquete del tipo put y se espera respuesta.
    - Se envía un paquete del tipo put (*put-request*); el paquete inicial posee todos los parámetros del archivo u objeto que se va a enviar.
    - Se espera por el paquete del tipo response (*put-response*).
        - Si no se obtiene. Se reenvía el primer paquete.
- Se siguen enviando los paquetes de tipo put y se espera confirmación. Todos los paquetes o pedazos del archivo siempre que no sea el último.
- Se envía el último paquete con las características vistas en los tipos de paquetes.
- Se cierra la conexión.

Cuando se abre la conexión, se debe enviar en un canal específico. Aquí es donde se vuelve importante el servicio de OPP. En la función, para saber si un dispositivo tiene OPP, se puede obtener el canal en el que se posee este servicio. Esto se puede hacer mediante la siguiente modificación a la función para obtener el canal.

```
//ciclo que recorre la lista de respuestas obtenidas en la búsqueda

//de servicios.

int canal=-1;
```

```
for(;response_list;response_list=response_list->next) {
sdp_record_t *fila_registro = (sdp_record_t *) response_list->data;
    if(sdp_get_access_protos(fila_registro,&proto_list)==0)    {
    canal=sdp_get_proto_port(proto_list,RFCOMM_UUID);
     }
     sdp_record_free(fila_registro);
}
```

Luego de esto, sobre este mismo canal, se debe abrir la conexión del socket para que se tener la implementación de OPP.

El código de la función que realiza el proceso anterior se describe a continuación:

```
/** parámetros:
* addr: dirección MAC del dispositivo con quien se va a conectar.
* buffer: el paquete que se va a enviar.
* buffsize: tamaño del buffer en bytes.
*/
int SenData(string addr, int channel, char * buffer, int buffsize) {
struct sockaddr_rc addr={0};
int status, mi_socket;
//creando un socket.
```

```
mi_socket=socket(AF_BLUETOOTH,
                        SOCK_STREAM, BTPROTO_RFCOMM);
//estableciendo los parametros de la comunicacion.
addr.rc_family=AF_BLUETOOTH;
addr.channel= channel; //canal de opp.
//convirtiendo el destino a un formato legible.
str2ba(addr.c_str(),&addr.bdaddr);
//conectando al servidor.
status=connect(socket,(struct sockaddr *)&addr,sizeof(addr));
//enviando la data.
if(0==status) {
        //las flags establecidas a 0.
        //buffsize: tamaño del buffer en bytes.
        status=send(socket, buffer,buffsize,0);         }
if(status<0) {
        cout <<"error en la conexion del socket!"<<endl;
        return -1; }
//envío exitoso, cerrando socket.
close(socket);
}
```

Antes de implementar esta función se debe agregar ciertas cabeceras para poder hacer uso de los archivos de la librería de bluetooth. Estas cabeceras son:

```
#include <stdio.h>

#include <unistd.h>

#include <sys/socket.h>

#include <bluetooth/bluetooth.h>

#include <bluetooth/rfcomm.h>
```

Una vez agregadas estas cabeceras, se puede hacer la implementación. En el código de la función se puede ver que primero realizan las declaraciones de las variables que van a ser usadas en la manipulación de los datos a ser enviados y de la comunicación, posteriormente se realiza la apertura de un socket con los parámetros establecidos, esto es mediante las líneas de código:

```
struct sockaddr_rc addr={0};

int status, mi_socket;

//creando un socket.

mi_socket=socket(AF_BLUETOOTH,

                    SOCK_STREAM, BTPROTO_RFCOMM);
```

Luego se establecen los parámetros del socket, aquí se puede ver que se establece la familia (AF_BLUETOOTH) y el canal en el que esta operando el servicio de OPP. Este canal es vital para la comunicación.

```
addr.rc_family=AF_BLUETOOTH;
```

```
addr.channel= channel;
```

Ahora se convierte la dirección MAC a un formato legible por la librería y se agrega como parámetro del socket.

```
str2ba(addr.c_str(),&addr.bdaddr);
```

Se procede a conectar al servidor.

```
status=connect(socket,(struct sockaddr *)&addr,sizeof(addr));
```

Los parámetros son: el socket creado, la dirección MAC con todos sus componentes (casteada al tipo *struct sockaddr*) y el tamaño de la estructura para alojar memoria.

```
if(0==status) {
        //las flags establecidas a 0.
        //buffsize: tamaño del buffer en bytes.
        status=send(socket, buffer,buffsize,0);
}
```

Si el *Status* es igual a 0, significa que la conexión fue exitosa, se procede a enviar la data. En este punto es necesario recordar que primero se debe enviar los paquetes de conexión y luego los paquetes del tipo put siguiendo el proceso que el protocolo de IrObex ha establecido. No se debe olvidar cerrar la comunicación mediante el mensaje del tipo *disconnect* establecido por el protocolo anteriormente mencionado. En caso no se haga la desconexión, el socket seguirá ocupado y el puerto también, haciendo imposible la conexión iniciada, tanto de otros dispositivos al propio como del éste a otros dispositivos.

Cada vez que se envíen los paquetes se debe cerrar el socket. Esto se hace mediante el siguiente código:

close(socket);

return 0;

Se retorna el valor de la función y se termina la misma. Como se pudo notar, en el código se maneja el error en el caso de que exista un fallo en la comunicación.

Para poder escuchar el mensaje del tipo *response*, se debe poner el socket en modo de escucha, a fin de poder escuchar cualquier mensaje que le pueda llegar, para esto es necesario modificar la función agregando las siguientes líneas de código:

```
//variables

//memoria de lectura.

struct sockaddr_rc rem_addr={ 0 };

unsigned int opt = sizeof(rem_addr);

int cliente, bytes_read;

char buffer_read[1024]={0}; //ajustar a tamaño de paquete.

//socket en modo escucha.

listen(socket, 1);

// se acepta la comunicacion.

cliente = accept (socket, (struct sockaddr *)&rem_addr, &opt ) ;

//se convierte lo leído a otro formato, en este caso a string.
```

```
ba2str( &rem_addr.rc_bdaddr, buffer_read ) ;

memset(buffer_read, 0, sizeof(buffer_read) ) ;

// leer los datos del cliente.

bytes_read = recv(client , buffer_read , sizeof(buffer_read) , 0);

if( bytes_read > 0 ) {

        cout<<"buffer recibido:  "<<endl<<buffer_read<<endl;

}
```

Esta modificación puede ser agregada a la función de envío de datos, o también se puede agregar como una función extra. Con estas dos funciones se debe hacer el envío de los datos, como se dijo anteriormente, siempre recordando el proceso y el formato establecido por el protocolo IrObex.

2.2.2.3.1.1.2. GetOnePacket

Para esta función, se selecciona un archivo, de cualquier tipo, y se establecen los parámetros de la separación en paquetes, estos paquetes se hacen dependiendo del tamaño especificado. Esta función recibe tres parámetros para poder realizar su desempeño.

Antes de implementar esta función se debe incluir las siguientes librerías en el archivo de implementación:

```
#include <ifstream>
#include <stdio.h>
#include <tclap/CmdLine.h>
using namespace std;
```

A continuación, se presenta el código de la función para, posteriormente, ser explicada en sus líneas de código.

```
char * getOnePacket (string filename, int packsize, int number) {
ifstream archivo;
int tamanio;
int apuntador=packsize*number;
char * buffer;
if(filename.length<0)return null;
buffer=new char[packsize*1024];
archive.open(filename ,   ios::binary);
archivo.seekg (0, ios::end);
tamanio = archivo.tellg();
archivo.seekg (0, ios::beg);
//validando la disponibilidad del paquete
If(apuntador>=tamanio) {
 cout<<"paquete fuera de rango"<<endl;
 return null;
 }
//extrayendo el paquete
bool loop=true;
int contador=0;
while(loop) {
        archivo.read(buffer, packsize*1024);
        if(number==contador) loop=false;
        contador++;
}
archivo.close();
return buffer;
}
```

Esta función toma el nombre del archivo, inicia la lectura dependiendo del tamaño del paquete y el número que se le está indicando. El número del paquete así como el tamaño del mismo le sirven para saber si se está leyendo en los límites del archivo. Si el apuntador de lectura de datos es mayor al tamaño del archivo, entonces esto representa que se está tratando de leer un paquete que no existe dentro de los paquetes del archivo en cuestión. Veamos en detalle.

Primero se muestran los parámetros pedidos por la función: *string filename, int packsize, int number*. Estos corresponden al nombre del archivo, el tamaño en *Kilobytes* del paquete y el número de paquete, que se está pidiendo, respectivamente.

Seguidamente se declara el objeto de tipo *ifstream*, el cual es un objeto que hereda de la clase *istream* que a su vez hereda de *ios*. Luego la variable que alojará el tamaño, en *bytes*, del archivo.

```
ifstream archivo;

int tamanio;
```

Se declara la variable que posee el valor hacia el cual apunta el inicio del iterador de lectura del archivo.

```
int apuntador=packsize*number;
```

Se declara el apuntador del buffer que se retornará como paquete de lectura.

```
char * buffer;
```

Sin embargo, es necesario asegurarse que el nombre de archivo sea un argumento válido, para esto se mide la longitud de la cadena. Esto es:

```
if(filename.length<0)return null;
```

Luego se crea el arreglo de caracteres que se retornará esto es:

```
buffer=new char[packsize*1024];
```

Ahora se procede a abrir el archivo, esto se hace de forma binaria. El primer parámetro es el nombre del archivo y el segundo la forma de apertura, para este caso en forma binaria.

```
archive.open(filename,  ios::binary);
```

Seguidamente se procede a obtener la longitud del archivo en *bytes*. Esto se hace con la función **seekg()** que proporciona el objeto *ifstream*. En realidad la función anterior proporciona la funcionalidad de establecer la posición del apuntador o iterador de lectura del archivo, es por esto que, cuando se le indica en sus argumentos el parámetro **ios::end**, se le está indicando que sitúe el apuntador de lector de datos, en el final y luego se obtiene este valor mediante la función **tellg().** Se debe regresar la posición del apuntador al inicio del archivo. Esto se hace dado que luego se va a recorrer el archivo y se desea que esté apuntando hacia el inicio de los datos. Para esto se usa de nuevo la función **seekg()** enviándole como parámetro **ios::beg** que hace referencia al inicio del archivo. Esto es el siguiente código:

```
archivo.seekg (0, ios::end);
tamanio = is.tellg();
archivo.seekg (0, ios::beg);
```

Luego se verifica que el paquete que se está pidiendo este contenido dentro del archivo. Esto se hace dado que puede estarse pidiendo un paquete que esté fuera de los límites del paquete. Es decir que exista un paquete que haga referencia a datos que están fuera de las fronteras del archivo en cuestión. Para

esto es necesario que el apuntador de paquetes esté por debajo del tamaño del archivo, esto se hace de la siguiente manera:

```
If(apuntador>=tamanio) {
        cout<<"paquete fuera de rango"<<endl;
        return null;
}
```

Ahora se procede a leer los datos, para esto se inician las variables que servirán de ayuda. **loop** es de tipo *booleano* que esta inicializada en valor verdadero, lo que indica que en la validación del bucle, éste si entrará a realizar las operaciones dentro del mismo. El contador se inicia en 0 **contador=0;** para contar el paquete que se esta pidiendo. Para leer los datos se usa la función **read()** enviándole los parámetros de: **buffer** el cual es la variable que alojará el segmento de datos leídos y **packsize*1024** que es el tamaño del bloque a ser leído. En este caso se usa esta operación para obtener el tamaño de datos que se esta leyendo, se debe recordar que el parámetro **packsize** es el tamaño del paquete en *Kilobytes*, esto indica que cuando se hace la instrucción **packsize*1024** en realidad se esta convirtiendo a *bytes*. Esta es la unidad de medida que recibe como segundo parámetro la función **read()** del objeto **ifstream**. Aquí la función mencionada, devuelve al buffer los datos, que corresponden al tamaño de bloque leído proporcionado por el segundo parámetro.

```
bool loop=true;
int contador=0;
while(loop) {
        archivo.read(buffer, packsize*1024);
        if(number==contador) loop=false;
        contador++;
}
```

Como se puede ver, existe una validación de que el numero de paquete sea del mismo tamaño que el contador. Esta validación (number==contador) se hace para terminar el ciclo cuando el paquete deseado se haya alcanzado y retornar este paquete. Es por esto que luego de que el número de paquete sea igual al contador, el valor de la variable **loop** se establece en falso, para que pueda salir del bucle **while** y luego retornar el buffer.

Por último se cierra el archivo y se retorna el buffer de datos.

```
archivo.close();
return buffer;
```

Como se puede ver la variable *buffer* era un apuntador del tipo carácter. Como buena práctica, se retorna el *buffer*, dado que esto ocasionará que la memoria pueda ser liberada y que se mantenga en un buen uso de memoria para alojamiento de los valores usados por la aplicación. En caso de que buffer sea usado localmente y que no se retorne como valor de un método o que el método sea un método del tipo *void* y que no permita el retorno de una variable del tipo apuntador; ésta (variable) se debe liberar o eliminar, esto se hace mediante la instrucción: **delete variable;** esto para liberar la memoria.

Una buena práctica es llevar un perfecto control de los valores de variables de tipo apuntador de memoria, para luego, poder liberar estos apuntadores y dejar la memoria, que ocupaban, libre y lista para ser usada, ya sea por nuestra aplicación o por alguna otra aplicación.

2.2.2.3.2. Negotiator

Es una parte importante de la comunicación, dado que, como se mencionó anteriormente, este componente es quien establece los parámetros de comunicación entre él (cliente) y el servidor. Además es quien proporciona toda la utilería necesaria para el manejo de los paquetes que se establecen en el

protocolo IrObex (ver Tipos de paquetes de IrObex.); tomando en cuenta que no solo es necesario definir el formato de los paquetes, sino también, permitir establecer valores para los distintos campos o cabeceras que los paquetes llevan.

Este componente es el encargado de las siguientes funciones:

- Negociar los términos de comunicación con el dispositivo que juega el rol de servidor.
- Establecer los parámetros de los paquetes del protocolo IrObex para comunicarse con los dispositivos servidores.
- Define el formato para los paquetes del tipo:
    - Conexión.
    - Desconexión.
    - Put.
    - Get.
    - Responses.
- Permite definir los campos, banderas (*flags*) y demás características de los paquetes que serán enviados a los dispositivos servidores.

Este componente define la paquetería de IrObex que se usa en la comunicación. Para esto, se debe tener claro que un paquete es equivalente a una agrupación de cabeceras. Así que en este componentes se definen los paquetes que son llamadas al componente Packet.

2.2.2.3.2.1. Funciones

A continuación, se presentan la función principal relacionada al componente Negotiatior. Esta se ayuda de otras funciones que son menor relevancia para este trabajo.

2.2.2.3.2.1.1. MakePacket

Esta función se encarga de la creación de un paquete en específico. Se describe esta función para generar las cabeceras, obligatorias, de un paquete del tipo put. Como se mencionó anteriormente, un paquete (en la mayoría de protocolos) es un arreglo de *bytes* ordenados de una manera específica, en cada protocolo. Esto permite reconocer tanto cabeceras como campos.

Los campos indican atributos y valores de características de los paquetes. En esta función primero se especifica el tipo de paquete, dado por su código de operación (*Operation Code*) y luego se crea el paquete. Éste es el parámetro de tipo entero que define cuáles cabeceras contendrá el paquete que será creado. Para una mejor comprensión de los paquetes y sus cabeceras, puede ver la sección 1.3.3.4.3, en la que se muestran los ejemplos de los paquetes usados en IrObex.

El código de esta función es el siguiente:

```
vector  MakePacket(int caso) {
vector <BYTE> ret;
   switch(caso) {
         case 0x02: { //paquete inicial
                     BYTE * HI=new BYTE[8];
                     HI=int2byte(0x02);
                     ret.push_back(HI);
                     ret.push_back(this.GetPacketLength());
                     ret.push_back(this.GetObjectName());
                     ret.push_back(this.GetObjectLength());
                     ret.push_back(this.GetBody());
               }break;
         case 0x82: {
```

```
                    //paquete final
                    BYTE * HI=new BYTE[8];
                    HI=int2byte(0x82);
                    ret.push_back(HI);
                    ret.push_back(this.GetBody());
                }break;
            case 0x80: {
                    //paquete de conexion.
                    //codigo de creacion del paquete.
            }break;
            case 0x81: {
                    //paquete de desconexion.
                    //codigo de creacion del paquete.
            }break;
    }
    return ret;
}
```

Aquí se puede ver como se construyen los paquetes. Primero se toma, como base, el valor del Código de Operación (*OpCode*), el cuál especifica el tipo de paquete. Luego, con base en éste (*OpCode*), se van agregando las distintas cabeceras que son obligatorias para cada paquete. El ejemplo que se muestra es el de un paquete del tipo Put. Se debe seguir el mismo proceso para la construcción de los demás paquetes, tomando en cuenta la premisa: **un paquete, es una secuencia de bytes, en un orden determinado por el protocolo**. Luego de esto se procede al envío de los paquetes.

2.2.2.3.3. Packet

Este componente es una clase que agrupa distintas cabeceras para conformar un paquete IrObex. Todos los campos (*Headers*) o valores de un paquete pueden tomar los valores que son previamente definidos por IrObex. Sin embargo deben tener la estructura dada para cada tipo de paquete.

Las cabeceras son un componente por definición, es decir, solo son un modelo de atributo-valor. Debe permitir establecer los valores de:

- Nombre (atributo).
- Valor.

Donde el nombre es el campo de nombrado tales como: *opcode, packet_length, body_length*, entre otros. Y el valor asignado es un valor como una cadena de bytes que se desea establecer para este campo.

2.2.2.3.3.1. Funciones

Estas funcione están creadas para definir y extraer las cabeceras de un paquete del tipo put. Para crear paquetes de conexión, desconexión y get, se debe seguir la misma convención, tanto para los paquetes *request* como para los *response*. Generando funciones de establecimiento de valores y funciones de obtención de los mismos.

Todas las funciones, usan el tipo de dato definido como typedef *unsigned char*, es decir la siguiente declaración:

typedef unsigned char BYTE;

Esto ocasiona que las variables puedan ser tratadas como arreglos de bytes o caracteres sin ningún tipo de dato como enteros o cadenas de caracteres.

2.2.2.3.3.1.1. Funciones de mutación

Este tipo de funciones son nombradas de esta manera ya que permiten el cambio de valores para establecer las condiciones correctas en las que un componente debe trabajar. Generalmente son llamadas de esta manera, o su equivalente en inglés *Mutator Methods*.

2.2.2.3.3.1.1.1. SetPacketLength

Establece el valor del largo del paquete que se esta enviando. Para esto se debe recibir como parámetro el valor de tipo entero del largo del paquete. También se debe tener la variable de acceso global **PLength,** del tipo Vector, casteado a *BYTE*, para poder acceder a ella y establecer el nuevo valor.

```
//declarando la variable Global:
vector <BYTE> PLength;
void SetPacketLength(int length) {
  BYTE len= new BYTE[16];
  len= int2byte(length); //convertir el entero a byte
  PLength.push_back(len);
}
```

El parámetro es de tipo entero, este debe tener la cualidad de ser un número de dos *bytes*. En el formato: 0x0000 donde 0000 es el número en hexadecimal. Por ejemplo si el paquete mide, en total, 1058 bytes; su conversión a hexadecimal es: 0x0422. Este número debe ser enviado como parámetro a la función. La función *int2byte* es una función que debe convertir el entero a un arreglo de bytes para ser ingresado al tipo declarado *unsigned char*. Una vez se haya hecho esta declaración, se tendrá un vector con una sola posición ocupada.

Esta posición contiene el identificador de la cabecera (*Header Id* = HI) para el largo del paquete.

2.2.2.3.3.1.1.2. SetObjectName

Establece el nombre del objeto o archivo que se esta enviando, es decir, los datos del paquete. Este objeto es el que se divide en pequeños paquetes (*chunks*) para ser enviado mediante los paquetes PUT del protocolo. Recibe un parámetro de tipo *string* que representa el nombre del archivo más su extensión. Se debe tener la variable global **OName**.

```
//declarando la variable
vector <BYTE>  OName;
void SetObjectName(string file_name) {
   //creando e insertando el identificador de cabecera
   BYTE hi=new BYTE[8];
   hi=int2byte(0x01); //conviertiendo el entero a un arreglo byte.
   OName.push_back(hi);
   //creando e insertando el largo del nombre
   BYTE len= new BYTE[16];
   len= int2byte(file_name.length());
   OName.push_back(len);
   //creando e insertando el valor del nombre.
   BYTE value= new BYTE[file_name.length*2];
   value=(BYTE * )file_name.c_str();
   OName.push_back(value);
}
```

Se envía el parámetro del nombre del archivo, esto es sólo el nombre con su extensión, sin tener en cuenta la ruta completa del mismo. Por ejemplo: archivo.txt.

Cada segmento del código, proporciona la creación de un campo del *header*. Como se sabe, primero se crean los identificadores de cabeceras, luego los valores, en el caso de nombre del archivo que se envía; primero se crea el **HI**, posteriormente se establece el valor del largo del nombre y luego el valor del nombre.

2.2.2.3.3.1.1.3. SetObjectLength

Establece la cabecera del largo del objeto o archivo que se esta enviando. Para esto se debe tener la variable global de nombre **OLength** del tipo Vector. Para tener acceso a ella.

```
vector <BYTE>  OLength;
void SetObjectLength(int length) {
   //creando e insertando el HI
   BYTE hi = new BYTE[8];
   hi=int2byte(0xC3);
   OLength.push_back(hi);

   //creando e insertando el valor
   BYTE value= new BYTE[32]; //4 bytes.
   value=int2byte(length);
   OLength.push_back(value);
}
```

Se puede ver que primero se crea el **HI** y luego se crea el valor del largo del objeto. Luego se insertan estos campos al vector, siguiendo el orden definido por el tipo paquete.

2.2.2.3.3.1.1.4. SetBody

Establece el tamaño del pedazo que se esta enviando. Cuando se habla del pedazo (*chunk*) del archivo, se entiende que el archivo es demasiado grande y que ha sido dividido en pequeños paquetes para ser enviado en su totalidad. Para esta función se establece la variable **Chunk** con alcance global.

```
vector <BYTE> Chunk= new Vector;
void SetChunkLength(BYTE buffer, int length, bool end) {
   //creando y agregando la cabecera.
   BYTE hi= new BYTE[8]; // 1 byte.
   int cabecera=0x00;
   if(end)cabecera= 0x49;
   else cabecera=0x48;
   hi=int2byte(cabecera);
   Chunk.push_back(hi);
   //creando e insertando el largo.
   BYTE len=new BYTE[16]; //2 bytes.
   len=int2byte(length+3); //largo + 1 (HI) + 2 (length).
   Chunk.push_back(len);
   //creando el cuerpo e insertandolo.
   Chunk.push_back(buffer);

}
```

Al igual que en la función anterior, en este campo se tienen: 1 *byte* para el **HI** y 2 *bytes* para especificar el tamaño del paquete. Luego viene la secuencia de *bytes* del paquete en sí, es decir el pedazo del archivo que se desea enviar. Este debe ser una secuencia de caracteres, en este caso del tipo BYTE. En esta función se toma un parámetro booleano para verificar si el paquete que se esta creando, es un paquete inicial, siguiente o final y, se especifica el tamaño del largo del pedazo, para ser establecido como valor de la cabecera.

2.2.2.3.3.1.2. Funciones de acceso

A diferencia de las funciones de mutación, las funciones de acceso permite la obtención de los valores que previamente fueron establecidos mediante las funciones de mutación. Son generalmente conocidas como *Accessor Methods.* Se usan este tipo de funciones ya que al igual que en las funciones de mutación, como buena practica se prefiere definir ciertas variables como privadas y así poder tener acceso a ellas únicamente mediante funciones propias escritas por el desarrollador.

2.2.2.3.3.1.2.1. GetPacketLength

Permite obtener el valor de la longitud del paquete. Ésta devuelve un arreglo de dos bytes con la longitud del paquete que se esta enviando.

```
Vector GetPacketLength() {
return PLength;
}
```

2.2.2.3.3.1.2.2. GetObjectName

Retorna el arreglo de campos de la cabecera del nombre del objeto o archivo. Valor previamente establecido mediante las funciones de mutación.

Vector GetObjectName() {

return OName;

}

2.2.2.3.3.1.2.3. GetObjectLength

Retorna el vector con la cabecera del largo del objeto o archivo que se esta enviando.

Vector GetObjectLength() {

return OLength;

}

2.2.2.3.3.1.2.4. GetBody

Retorna el vector o arreglo de *bytes* que posee toda la cabecera del cuerpo del mensaje o paquete.

Vector GetBody() {

return Chunk;

}

Todas las funciones descritas anteriormente son llamadas por las funciones encargadas de la creación de los paquetes. Es de esta forma en que los componentes se comunican.

Los componentes deben ser escritos, de forma ideal, de la manera en la que se ha mostrado, sin embargo, puede haber modificaciones que convengan a la persona que este desarrollando la aplicación de software.

Una vez se hayan escrito los componentes y se haya comprobado que no existen errores en los mismos, tanto lógicos como de compilación, se procede a

la implementación y puesta en marcha de la solución completa. Es decir se ensambla el producto de software y se prueba con su respectivo hardware.

3. IMPLEMENTACIÓN

A continuación, se presenta la forma en la que se debe hacer la implementación de la solución que se propone en el marco práctico.

3.1. 10 pasos para una implementación exitosa de Proximity Marketing

Primero: se reúnen los componentes descritos dentro de los requisitos de hardware de la solución descritos en la sección 1.4.1.2 **Hardware,** de esta tesis.

Segundo: se elaboran los componentes de software descritos en el marco práctico de esta tesis.

Tercero: se compilan los distintos archivos de los componentes de software que se generaron en el marco práctico.

Aquí, con la ayuda del IDE, se deben agregar las librerías, que se estuvieron usando, a los parámetros del proyecto. Esto se hace en las propiedades del proyecto.

Se agregan todas las librerías que se usaron, como bluetooth, *lpqxx* (para la base de datos) y las demás que fueron incluidas como parte de la creación de la solución de software.

Se debe notar que una vez las librerías hayan sido agregadas al proyecto, la forma de compilación se hace dentro del IDE de desarrollo. Se puede ver que cuando todos los parámetros de la compilación estén completos, las librerías aparecerán agregadas al comando de compilación que se muestra en la consola (*output*) del IDE. Una vez el proyecto esté listo, sin errores léxicos,

sintácticos y semánticos, el IDE generará el objeto ejecutable este es un archivo que tendrá la convención de nombrado: <nombre_proyecto>.o (<nombre_proyecto> será sustituido por el nombre que se le asignó al proyecto) y este archivo puede ser ejecutado desde el IDE o desde una terminal del sistema.

Cuarto: se insertan las antenas al servidor que será el encargado de tener la ejecución de la solución de software. Se debe verificar que las antenas, al ser conectadas, sean reconocidas por el sistema operativo y puedan ser usadas perfectamente.

> ***Opcional:*** Como se mencionó en el marco teórico, de esta tesis, se puede tener un enrutador de antenas de Bluetooth, esto se hace para tener muchas más antenas a la disposición, para la ejecución del software de la solución. Se debe colocar, entonces, las antenas al enrutador, luego éste se debe conectar al servidor y verificar conexión y funcionalidad mediante los asistentes o programas propios del sistema operativo del servidor.

Quinto: se debe preparar el ambiente de la base de datos. Esto es, establecer las variables del sistema para el correcto funcionamiento de la base de datos. Para esto refiérase al marco práctico, en el componente DbManager, para la preparación del ambiente para el funcionamiento de la base de datos.

Sexto: somo convención de esta tesis, se propone que el Scanner y el Sender sean ejecutados como dos procesos independientes, es decir, el Scanner se encarga de llenar la base de datos con los dispositivos encontrados, esto, como proceso independiente; a su vez, el Sender es un proceso independiente

que se encarga de obtener los datos que la base de datos posee (datos proporcionados por el Scanner) y proceder al envío de contenidos. Así que como sexto paso, se generan los procesos respectivos para el Scanner y el Sender a fin de poder ejecutarlos de manera independiente. Una vez se tengan los dos procesos corriendo de manera independiente se procede al siguiente paso.

Séptimo: hasta aquí, ya se tiene la mayor parte de la implementación realizada. Lo que a continuación se presenta, es la forma del manejo de todo el proceso que encierra una solución de software. En este punto, se debe escoger los contenidos publicitarios que se estarán usando en la solución. En este caso los archivos escogidos son archivos de tipo imagen que serán enviados a los dispositivos de bluetooth para que luego puedan ser vistos por el usuario de cada dispositivo y cumplir con la labor de la publicidad.

Octavo: aquí se escoge el algoritmo que manejará los dos procesos independientes, es decir el Scanner y el Sender. Como parte de esta solución se ha usado un algoritmo que se describe a continuación:

mientras (condición_de_verdad) hacer
entero contador=0;
mientras (contador<numero_antenas) hacer
//ejecutar scanner sobre la antena que posee el contador
ejecutar_proceso(Scanner, contador);
//esperar a que se desocupe la antena. 1 minuto.
esperar(1);
ejecutar_proceso(Sender, contador);
//esperar a que se desocupe la antena para el siguiente ciclo.

esperar(1);
fin mientras
fin mientras

Como se puede ver se alternan los procesos. Por supuesto, este algoritmo puede ser modificado o cambiado por completo, esto, para acomodarse a las necesidades del cliente. Generalmente se rediseñan los algoritmos luego de las pruebas que se realicen.

Noveno: la ubicación de antenas es algo importante, dado que la señal de las mismas debe llegar a los dispositivos que las personas poseen para que puedan tener intercomunicación.

La mayoría de veces, la colocación física de los objetos es basado en la premisa de "ensayo y error", para este caso no se hará uso de esta premisa, dado que, por intuición, las comunicaciones, de forma ideal, no deben tener obstáculos para que sean perfectas. Sin embargo, como se sabe, la idea de tener Proximity Marketing es tener interacción con las personas, así que, una forma de evitar los obstáculos es la ubicación de las antenas en la parte superior de los lugares en los que se estén ubicando. Esto provocará que las antenas tengan un enlace casi directo con los usuarios de los dispositivos de bluetooth y así lograr una mejor comunicación.

Además para el alcance, Juan Pablo Hurtarte, en su tesis: "Condiciones necesarias para la conexión de dispositivos inalámbricos en Guatemala a través del protocolo bluetooth", estsablece que el patrón de pérdidas, representado en decibeles, esta dado por la ecuación:

$$\boldsymbol{PL = 20\log\left(\frac{4\pi}{\lambda}\right) + 10\eta\log(d)}$$

En donde que η es un coeficiente que representa las condiciones del lugar o ambiente en el que se esta emitiendo la señal de Bluetooth, d es la distancia que llegará. Y λ es la frecuencia de onda que, en el caso de Bluetooth, es 2.45 GHz (Giga Hertz).

La tabla III muestra los valores que puede tomar η y la pérdida en decibeles (dB) según las condiciones del medio ambiente.

Tabla III. Valores promedio de η

Ubicación	η	dB
Oficina abierta	2.2	8.7
Oficina con paredes móviles	2.4	9.6
Oficina con paredes fixed	3.0	7.0
Industrias metálicas con propagación LOS	1.6	5.8
Industrias metálicas con propagación non-LOS	3.3	6.8
En el mismo nivel (promedio)	2.8	12.9
A través de niveles continuos (promedio)	4.2	5.1
A través de niveles separados un nivel (promedio)	5.0	6.5
A través de niveles separados dos niveles (promedio)	5.2	6.7

Fuente: Condiciones necesarias para la conexión de dispositivos inalámbricos en Guatemala a través del protocolo bluetooth. Pág.:95. USAC, 2005.

También se sabe que la sensibilidad mínima que debe percibir un receptor, es decir el factor de perdida (PL en la ecuación), es de -70 dBm. Si se toma este valor como la igualdad en la ecuación y teniendo en cuenta el valor de las demás variables, se puede obtener el valor de la distancia máxima que se

puede obtener en las condiciones de un ambiente dado. De esta forma se puede obtener una ubicación que sea óptima para las antenas de Bluetooth.

Décimo: este paso es el encargado de hacer lo que en inglés se conoce como "*improvement*" o la mejora. La idea es tener una fase de pruebas que proporcionen resultados y a su vez los mismos proporcionarán causas de posibles errores que deben ser arreglados mediante las mejoras a todo el proceso. En este punto, se debe observar el comportamiento de los componentes y de todo el proceso durante al menos 45 días. Se debe hacer pruebas sobre horas pico, para saber si existe, o no, alguna saturación por parte de los componentes. A partir de los datos obtenidos por las pruebas, se debe hacer mejoras a fin de lograr, mediante integración continua, un proceso óptimo en el desarrollo de Proximity Marketing sobre tecnología de Bluetooth.

CONCLUSIONES

1. El uso de Bluetooth es una buena alternativa, para la publicación de contenidos, y prometedora en el ámbito de los medios interactivos de publicidad.
2. Proximity Marketing, basado en el uso de bluetooth, es una forma que ofrece control tanto para el publicista como para el cliente que recibe la publicidad.
3. El alcance de las antenas de bluetooth, tomando en cuenta una interferencia media, es óptimo para el anuncio de publicidad de las casas comerciales, que desean anunciarse a sus clientes, los cuales pasan en cercanías a la ubicación física de éstas (casas comerciales).
4. Dependiendo del lenguaje de programación, en el que se desee programar los componentes, existirá variación en la forma de la manipulación de las antenas y las funciones de las mismas.
5. Mediante el servicio de OPP se puede saber, cuál dispositivo es ideal para el intercambio de datos, sin necesidad de palabras de autenticación o *pairing.*
6. El protocolo IrObex proporciona una forma explicita para el intercambio de información entre dos dispositivos de bluetooth.
7. El protocolo IrObex es una modificación de Obex (para infrarrojo), el cual puede ser fácilmente mapeado al protocolo DHCP; por lo que IrObex puede ser usado como modelo en otro tipo de comunicaciones.
8. El tipo de tecnología usada en la publicidad puede tener, en el cliente, un impacto importante. Este impacto a veces puede impulsar a un cliente a comprar o no, un bien o servicio que se anuncia.

9. A pesar de la versatilidad de la tecnología de Bluetooth, no debe ser usada para cubrir grandes distancias, incluso sin tener obstáculos en la trayectoria de la onda de la frecuencia.

RECOMENDACIONES

Luego del desarrollo de ciertas pruebas, se recomiendan las siguientes mejoras:

MEJORES PRÁCTICAS

1. Algoritmo de Coordinación

Como se pudo notar, en el algoritmo de conexión se poseen antenas desocupadas, es decir, cuando un proceso se está ejecutando, el otro proceso se mantiene ocioso. Esto no es conveniente si se desea cubrir la mayor parte de los clientes que ingresan a un centro comercial. Para esto se recomienda modificar el algoritmo de los procesos del inicial al siguiente:

```
mientras(condicion_de_verdad) hacer
        //arreglo con las direcciones mac de las antenas.
        cadena [] antenas= nueva cadena[num_antenas];
        entero scPtr=1; //apuntador del Scanner.
        entero snPtr=scPtr-1; //apuntador del Sender.
        booleano cond_de_verdad=verdad;
        mientras(cond_de_verdad) hacer

                current_antena=antenas[scPtr];
                si (desocupada(current_antena)) entonces
                        ejecutar_proceso(Scanner, current_antena;)
                fin si
```

```
                //esperar 30 segundos para evitar saturación de antenas.
                esperarSegundos(10);
                current_antena=antenas[snPtr];
                si (desocupada(current_antena)) entonces
                        ejecutar_proceso(Sender,current_antena);
                fin si
                contador++;
                scPtr=scPtr+1;
                snPtr=scPtr-1;
                si(scPtr>num_antenas) entonces
                        scPtr=1;
                fin si
        fin mientras
fin mientras
```

Esto provoca que se vaya haciendo la creación de procesos simultáneos, luego, cuando un proceso se termine, deja desocupada la antena y un nuevo proceso puede usar una antena; de esta forma se reduce el tiempo ocioso de cada antena.

2. Ubicación de las antenas de bluetooth

En el caso de la ubicación de las antenas, luego de las pruebas se determinó que la mejor ubicación, para un centro comercial tiene que ver con dos variables:

- La altura apropiada para la ubicación.
- La localización horizontal de la antena.

En varios casos probados se pudo determinar que en un centro comercial de tipo estándar se debe colocar las antenas o el servidor de antenas en la esquina superior del mismo. El punto clave es buscar la mayor cobertura de las antenas, para esto se debe dejar de forma tal que las antenas estén con cierto ángulo de inclinación (generalmente -30 grados sobre el eje horizontal) que permita que el alcance sea máximo y que la forma de la cobertura de las mismas permita abarcar mucha más área y por consiguiente muchos más dispositivos.

3. Red de servidores de Proximity Marketing

Se puede tener una red de servidores de Proximity Marketing, es decir, un conglomerado de enrutadores de antenas de bluetooth que puedan ponerse de acuerdo para poder interactuar entre ellos y poder cubrir varias extensiones de áreas de dispositivos de bluetooth. Esto proporciona la ventaja de coordinar diferentes tipos de publicidad para cada región, por ejemplo; publicidad para el área de restaurantes diferente con respecto de la publicidad para el área de deportes. Como recomendación para la construcción de una red de servidores, se propone el uso de las redes inalámbricas *Wi-fi* para poder cubrir distancias más grandes. Se puede hacer uso de las mismas antenas de bluetooth para formar una red, pero es preferible el uso de *Wi-fi* por el alcance de las antenas de bluetooth, en comparación con el alcance de las antenas inalámbricas de *Wi-fi*.

BIBLIOGRAFÍA

1 **Assigned Numbers – Service Discovery Protocol (SDP).** The Bluetooth SIG, Inc. 2003.

2 Burgos, Enrique y otros. **Del 1.0 al 2.0: claves para entender el nuevo marketing.** Versión 1.0. España: Bubok Publishing. Marzo 2009.

3 Huang, Albert y Larry Rudolph. **BLUETOOTH essentials for programmers.** Estados Unidos: Cambridge University Press. 2007.

4 Hurtarte Cáceres, Juan Pablo. CONDICIONES NECESARIAS PARA LA CONEXIÓN DE DISPOSITIVOS INALAMBRICOS EN GUATEMALA A TRAVÉS DEL PROTOCOLO BLUETOOTH. Tesis de ingeniería electrónica. Guatemala. USAC. Facultad de ingeniería. Agosto, 2005.

5 IEEE. **Bluetooth Specification. Generic Object Exchange Profile.** Part K: 10. Version 1.1. Estados Unidos: IEEE. Febrero, 2001.

6 IEEE. **Bluetooth Specification. IrDA Interoperability.** Part F: 2. Versión 1.1. Estados Unidos: IEEE. Febrero, 2001.

7 IEEE. **Bluetooth Specification. Object Push Profile.** Part K: 11. Versión 1.1. Estados Unidos: IEEE. Febrero, 2001.

8 Megowan, Pat y otros. **IrDA ® OBEX ™ Test Specification.** Versión 1.0.1. Estados Unidos: Extended Systems, Inc. 2002.

9 Megowan, Pat y otros. **IrDA ® OBEX ™ Obex Errata. Compiled for 1.3.** Estados Unidos: Extended Systems, Inc. 2003.

10 Megowan, Pat y otros. **IrDA ® Object Exchange Protocol OBEX ™.** Versión 1.3. Estados Unidos: Extended Systems, Inc. 2002.

11 Schildt, Herbert. **The Art of C++.** Estados Unidos: McGraw-Hill/Osborne. 2004.

12 Weber, Larry. **Marketing to the social web. How digital customer communities build your business.** Hoboken, New Jersey: Wiley & Sons, Inc. 2007.

APÉNDICES

1. Identificador de Cabeceras (Header Identifiers)

HI - identifier	Header name	Description
0xC0	Count	Number of objects (used by Connect)
0x01	Name	name of the object (often a file name)
0x42	Type	type of object - e.g. text, html, binary, manufacturer specific
0xC3	Length	the length of the object in bytes
0x44 0xC4	Time	date/time stamp – ISO 8601 version - preferred date/time stamp – 4 byte version (for compatibility only)
0x05	Description	text description of the object
0x46	Target	name of service that operation is targeted to
0x47	HTTP	an HTTP 1.x header
0x48 0x49	Body End of Body	a chunk of the object body. the final chunk of the object body
0x4A	Who	identifies the OBEX application, used to tell if talking to a peer
0xCB	Connection Id	an identifier used for OBEX connection multiplexing
0x4C	App. Parameters	extended application request & response information
0x4D	Auth. Challenge	authentication digest-challenge
0x4E	Auth. Response	authentication digest-response
0xCF	Creator ID	indicates the creator of an object
0x50	WAN UUID	uniquely identifies the network client (OBEX server)
0x51	Object Class	OBEX Object class of object
0x52	Session-Parameters	Parameters used in session commands/responses
0x93	Session-Sequence-Number	Sequence number used in each OBEX packet for reliability
0x14 to 0x2F	Reserved for future use	this range includes all combinations of the upper 2 bits
0x30 to 0x3F	User defined	this range includes all combinations of the upper 2 bits

Fuente: Object Exchange Protocol, OBEX ™ IrDA, 2003, versión 1.3

2. Códigos de Operación (OpCode)

Opcode (w/high bit set)	Definition	Meaning
0x80 *high bit always set	Connect	choose your partner, negotiate capabilities
0x81 *high bit always set	Disconnect	signal the end of the session
0x02 (0x82)	Put	send an object
0x03 (0x83)	Get	get an object
0x04 (0x84)	Reserved	not to be used w/out extension to this specification
0x85 *high bit always set	SetPath	modifies the current path on the receiving side
0x06 (0x86)	Reserved	not to be used w/out extension to this specification
0x87 *high bit always set	Session	used for reliable session support
0xFF *high bit always set	Abort	abort the current operation
0x08 to 0x0F	Reserved	not to be used w/out extension to this specification
0x10 to 0x1F	User definable	use as you please with peer application
Bits 5 and 6 are reserved and should be set to zero.		
Bit 7 of the opcode means Final packet of request.		

Fuente: Object Exchange Protocol, OBEX ™ IrDA, 2003, versión 1.3.

3. Números asignados de Obex (Obex Numbers)

Mnemonic	UUID size	Short UUID	Name	Ref
SDP	uuid16	0x0001	bt-sdp	See Bluetooth Service Discovery Protocol (SDP), Bluetooth SIG.
UDP	uuid16	0x0002		
RFCOMM	uuid16	0x0003	bt-rfcomm	See RFCOMM with TS 07.10, Bluetooth SIG.
TCP	uuid16	0x0004		
TCS-BIN	uuid16	0x0005	bt-tcs	See Bluetooth Telephony Control Specification / TCS Binary, Bluetooth SIG.
TCS-AT	uuid16	0x0006	modem	
OBEX	uuid16	0x0008	obex	
IP	uuid16	0x0009		
FTP	uuid16	0x000A	ftp	
HTTP	uuid16	0x000C	http	
WSP	uuid16	0x000E	wsp	
BNEP	uuid16	0x000F		BNEP
UPNP	uuid16	0x0010		ESDP
HIDP	uuid16	0x0011		See Human Interface Device Profile (HID), Bluetooth SIG
HardcopyControlChannel	uuid16	0x0012		See Hardcopy Cable Replacement Profile (HCRP), Bluetooth SIG
HardcopyDataChannel	uuid16	0x0014		See Hardcopy Cable Replacement Profile (HCRP), Bluetooth SIG
HardcopyNotification	uuid16	0x0016		See Hardcopy Cable Replacement Profile (HCRP), Bluetooth SIG
AVCTP	uuid16	0x0017		Audio/Video Control Transport Protocol, Bluetooth SIG
AVDTP	uuid16	0x0019		Audio/Video Distribution Transport Protocol, Bluetooth SIG

SerialPort	uuid16 0x1101	See Generic Access Profile, Bluetooth SIG.
LANAccessUsingPPP	uuid16 0x1102	
DialupNetworking	uuid16 0x1103	See Dial-up Networking Profile, Bluetooth SIG.
IrMCSync	uuid16 0x1104	See Synchronization Profile, Bluetooth SIG.
OBEXObjectPush	uuid16 0x1105	See Object Push Profile, Bluetooth SIG.
OBEXFileTransfer	uuid16 0x1106	See File Transfer Profile, Bluetooth SIG.
IrMCSyncCommand	uuid16 0x1107	See Synchronization Profile, Bluetooth SIG.
Headset	uuid16 0x1108	See Generic Access Profile, Bluetooth SIG.
CordlessTelephony	uuid16 0x1109	See Cordless Telephony Profile, Bluetooth SIG.

Fuente: Assigned Numbers – Service Discovery Protocol (SDP).