Tesis
TI 2011
M367

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA INFORMÁTICA

Diseño y Desarrollo Formal de un Sistema Novedoso de Mensajería Instantánea para Usuarios Móviles

TRABAJO INSTRUMENTAL DE GRADO

Presentado ante la

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

Como parte de los requisitos para optar al título de

INGENIERO EN INFORMÁTICA

REALIZADO POR	MARIANI HIDALGO, Danielle J.
TUTOR EMPRESARIAL	ALONZO SANCHEZ, Carlos.
TUTOR ACADÉMICO	TARRÍO VILLAVERDE, Gloria.
FECHA	Octubre de 2011

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA INFORMÁTICA

Diseño y Desarrollo Formal de un Sistema Novedoso de Mensajería Instantánea para Usuarios Móviles

Este Jurado; una vez realizado el examen del presente trabajo ha evaluado su contenido con el resultado: Veinte puntos (20)

J U R A D O E X A M I N A D O R

Firma:		
Nombre:	GLORIA TARRÍO V.	Wilmer Pereira	Luis Ignacio Núñez

REALIZADO POR	MARIANI HIDALGO, Danielle J.
TUTOR EMPRESARIAL	ALONZO SANCHEZ, Carlos.
TUTOR ACADÉMICO	TARRÍO VILLAVERDE, Gloria.
FECHA	Octubre de 2011

Para Luis Raúl, Silberia y Danny

"Cuando deseas alguna cosa, todo el Universo conspira para que puedas realizarla"

Paulo Coelho

AGRADECIMIENTOS

Primero que nada, agradezco a Dios por todo lo que me ha dado en la vida, por guiar mis pasos, por darme la fuerza y la voluntad para seguir adelante y superar todos mis obstáculos.

Agradezco a mi madre Danny Mariani y a mi abuela Silberia H. de Mariani, por todo el apoyo que me brindaron a lo largo de mi carrera universitaria, por la formación y principios que recibí de su parte, por siempre estar allí y enseñarme a ser una persona humilde. También agradezco al resto de mi familia, a mis tíos y mis primos por sus consejos y por alentarme a seguir adelante cada día.

A mis tutores, Gloria Tarrío por siempre estar atenta, por apoyarme, darme consejos e indicaciones, y a Carlos Alonzo por confiar en mí y motivarme durante todo el transcurso del proyecto, agradezco sinceramente el tiempo que invirtieron en mí. Agradezco a Divux Systems, por abrirme las puertas al campo laboral y darme la oportunidad de realizar este Trabajo Instrumental de Grado, también agradezco a todo el equipo de trabajo que allí opera, que día a día estuvieron ayudándome en lo posible.

Agradezco al amor de mi vida María del Mar Linarez por estar siempre allí en el beso y la distancia, por sus palabras de apoyo, por su paciencia y comprensión, por alentarme a seguir adelante para realizar mis metas.

A mis hermanos de corazón Rubén Hernández, Joel Hernández y Gustavo González, y a Adriana Márquez, por apoyarme, por mantenernos siempre unidos y demostrarme lo valioso que es la amistad.

Por último agradezco a todas aquellas personas que me apoyaron, a mis amigos, a mis profesores, a mis compañeros de estudio y de proyectos, mis más sinceros agradecimientos a todos.

Danielle Mariani

ÍNDICE DE CONTENIDO

ÍNDICE DE CONTENIDO iv
APENDICES vi
ÍNDICE DE FIGURAS vii
SINOPSIS viii
INTRODUCCIÓN 9
Necesidades de la Empresa 10
Solución Propuesta 11
Objetivos 12
Objetivo General 12
Objetivos Específicos 12
Alcance 13
Limitaciones 14
Justificación 15
CAPITULO I. MARCO TEÓRICO 17
I.1 Telefonía Móvil 17
I.2 Teléfono Celular 18
I.3 Teléfono Inteligente 18
I.4 Mensajería Instantánea 19
I.5 Cliente de Mensajería Instantánea 20
I.6 XMPP 21
I.7 Openfire 24
I.8 Android 25
I.9 BlackBerry 27
I.10 Modelo Vista Controlador 28
I.11 DAO 30
I.12 IDE 30
I.13 Java 30
I.14 XML 31
CAPITULO II. MARCO METODOLÓGICO 32
II.1 Exploración 32
II.2 Inicialización 32
II.3 Desarrollo 32
II.4 Estabilización 33
II.5 Pruebas 33
CAPITULO III. RECURSOS 34
III.1 Android SDK 10 34
III.2 Blackberry SDK 5.0 34
III.3 Smack API 3.1.0 34
III.4 SQLite 35

CAPITULO IV. DESARROLLO 36
IV.1 Etapa de Exploración 36
IV.1.1 Identificación y Análisis de Requerimientos 36
IV.2 Etapa de Inicialización 37
IV.2.1 Investigación 37
IV.2.2 Diseño 37
IV.2.3 Capacitación 38
IV.3 Etapa de Desarrollo 39
IV.3.1 Primera Iteración: Aplicación Android/BlackBerry – Inicialización y Conexión 39
a. Planificación 40
b. Trabajo 40
c. Liberación 44
IV.3.2 Segunda Iteración: Aplicación Android/BlackBerry – Lista de Contactos 46
a. Planificación 47
b. Trabajo 47
c. Liberación 51
IV.3.3 Tercera Iteración: Aplicación Android/BlackBerry – Mensajería Instantánea 52
a. Planificación 53
b. Trabajo 53
c. Liberación 59
IV.3.4 Cuarta Iteración: Aplicación Android/BlackBerry – Gestión del Perfil de Usuario 61
a. Planificación 62
b. Trabajo 62
c. Liberación 64
IV.4 Etapa de Estabilización 65
IV.4.1 Integración del Sistema 65
IV.4.2 Documentación 65
IV.5 Etapa de Pruebas de Sistema 65
CAPITULO V. RESULTADOS 66
CONCLUSIONES Y RECOMENDACIONES 73
REFERENCIAS BIBLIOGRÁFICAS 76

APENDICES

Apéndice A. Estructura del Código Fuente 78

Apéndice B. Pantallas de Inicio de Sesión 79

Apéndice C. Campo de contacto (ContactField) 79

Apéndice D. Pantallas de Lista de Contactos 80

Apéndice E. Pantallas de Conversación 81

Apéndice F. Notificaciones de Mensajes 82

Apéndice G. Pantallas del Perfil de Usuario 83

Apéndice H. Modelo de la Base de Datos Nativa de las Aplicaciones 84

Apéndice I. Manual de Usuario - Parte I 85

Apéndice J. Manual de Usuario - Parte II 86

Apéndice K. Manual de Usuario - Parte III 87

Apéndice L. XEP – Notificación de Mensajes Leídos – Parte I 88

Apéndice M. XEP – Notificación de Mensajes Leídos – Parte II 89

Apéndice N. Caso de Uso - Iniciar Sesión 90

Apéndice O. Caso de Uso - Cerrar Sesión 90

Apéndice P. Caso de Uso - Recordar Nombre de Usuario y Contraseña 91

Apéndice Q. Caso de Uso - Recordar Nombre de Usuario 91

Apéndice R. Caso de Uso - Enviar Mensaje 92

Apéndice S. Caso de Uso - Editar Perfil 92

ÍNDICE DE FIGURAS

Figura 1. Modelo de Red Celular. (Kioskea, 2008) 17

Figura 2. Teléfonos Inteligentes. (WebAccion, 2011) 19

Figura 3. Arquitectura del Sistema Operativo Android. (Android SO, 2011) 27

Figura 4. Patrón de Arquitectura de Software Modelo Vista Controlador 29

Figura 5. Ciclo de Vida del Desarrollo de Mobile-D 33

SINOPSIS

El presente Trabajo Instrumental de Grado (T.I.G.) titulado "Diseño y Desarrollo formal de un Sistema Novedoso de Mensajería Instantánea para Usuarios Móviles", desarrolla un sistema de Mensajería Instantánea (MI) para teléfonos inteligentes *Android* y *BlackBerry*, en el cuál le es permitido al usuario iniciar sesión, visualizar su lista de contactos, intercambiar mensajes con un contacto y personalizar su perfil de usuario.

La finalidad de este proyecto es brindar un nuevo producto de comunicación a toda la sociedad venezolana patrocinado por una importante compañía telefónica del país, bajo la limitante que solo está disponible para dispositivos móviles con sistema operativo Android 2.3 y BlackBerry 5.0.

Mobile-D fue la metodología empleada para la elaboración del proyecto, propuesta por la gerencia de proyectos de DIVUX SYSTEMS. Se basa en un desarrollo por etapas e iteraciones en donde la etapa desarrollo llevó a cabo en cinco (5) iteraciones que permitieron realizar los procesos de inicio y cierre de sesión mediante la autenticación del usuario, el listado de contactos y su información personal, el intercambio de mensajes, notificaciones de estado de los mensajes y estados de conversaciones, y por último, la gestión del perfil de usuario.

La parte técnica del sistema se basó en la implementación del protocolo de comunicación *XMPP* y consumir de los servicios del servidor *Openfire*, el cual es el referente principal de la plataforma de MI.

El proyecto utilizó las tecnologías de desarrollo en el lenguaje de programación Java, la implementación de *SQLite* para la gestión de la base de datos nativa de las aplicaciones, como también, fueron aprovechadas las ventajas del desarrollo bajo una arquitectura Modelo Vista Controlador permitiendo así una estructura de código organizada.

INTRODUCCIÓN

La Mensajería Instantánea (MI) se ha convertido hoy en día en un sistema de comunicación esencial, donde cada vez es mayor el número de usuarios que hacen uso de un cliente de MI para comunicarse con sus familiares, amistades y conocidos, todo esto con la finalidad de mantener un vinculo que los mantenga informados.

DIVUX SYSTEMS S.R.L (en adelante, DIVUX SYSTEMS) es una empresa especializada en el área de desarrollo y gestión de proyectos orientados a aplicaciones Web y en los últimos años incursionó en el desarrollo de aplicaciones para dispositivos móviles inteligentes. Tras continuos estudios del mercado de las aplicaciones móviles se determinó la elaboración de una aplicación multiplataforma que permita la comunicación de los usuarios de una operadora telefónica del país. El objetivo fundamental es desarrollar un novedoso cliente de mensajería instantánea disponible en teléfonos inteligentes *Android* y *BlackBerry*, el cual sea de fácil uso y le resulte agradable al usuario, para ello, se definieron un conjunto de objetivos específicos que determinan la trayectoria a seguir para el correcto desarrollo del proyecto.

Para llevar a cabo el desarrollo de las aplicaciones se debe tomar en cuenta diversos factores como lo son el lenguaje de programación de las aplicaciones, el protocolo de comunicación a emplear, el servidor que mantenga el sistema de mensajería instantánea activo y por último la conexión con este servidor. Tomando en cuenta los puntos mencionados se procedió a la elaboración las aplicaciones de MI Android y BlackBerry cumpliendo los requerimientos exigidos.

Necesidades de la Empresa

Las compañías de telefonía móvil en Venezuela hacen inversión de tiempo y dinero en la innovación de nuevos productos y servicios que son ofrecidos a sus clientes, esto se debe a la demanda que ha ejercido la sociedad en los últimos años en el entorno de las telecomunicaciones del país, produciendo una fuerte competencia entre las compañías telefónicas que operan en Venezuela.

Ante esta situación, DIVUX SYSTEMS ha venido trabajando en el desarrollo de soluciones innovadoras, con la finalidad de lanzar una gama de productos basados en aplicaciones para dispositivos móviles inteligentes que puedan operar en distintas plataformas.

La propuesta actual implica la construcción de una plataforma social amplia, compleja y de vanguardia contemplando varios aspectos: creación de una red social para Web y móviles personalizable a las exigencias del cliente; favorecer que los usuarios de la red compartan contenidos, gustos, preferencias e información; facilitar la integración con otras redes sociales existentes; proveer de un servicio novedoso de Mensajería Instantánea (en inglés IM, *Instant Messaging*), que a su vez pueda integrarse con otros IMs ya existentes en el mercado global.

Solución Propuesta

Se planteó para Trabajo Instrumental de Grado, el desarrollo formal de una aplicación para dispositivos móviles con sistemas operativos *Android* y *BlackBerry*, en donde el cliente pueda acceder a los servicios de mensajería instantánea.

La Aplicación

Debe dar soporte para los sistemas operativos *Android* y *BlackBerry*, bajo el lenguaje de desarrollo Java.

Protocolo de Comunicación

Para llevar a cabo la transmisión de datos entre clientes de IM, se estudiará el protocolo XMPP (Protocolo Extensible de Mensajería y Comunicación de Presencia / *eXtensible Messaging and Presence Protocol*), el cual es un protocolo abierto y extensible basado en XML, ideado para mensajería instantánea. Se deberá proponer mejoras al protocolo para garantizar la seguridad y eficiencia en la transmisión de datos sobre la red celular.

El Servidor

Se empleará *Openfire*, el cual es un sistema de mensajería instantánea de código abierto hecho en Java y emplea el protocolo XMPP, con él se pueden administrar usuarios, compartir archivos, auditar mensajes, mensajes *offline*, mensajes *broadcast*, grupos, etc.

Objetivos

Objetivo General

Desarrollar una solución de Mensajería Instantánea novedosa y de valor agregado para los usuarios de telefonía móvil en Venezuela.

Objetivos Específicos

- Estudiar el protocolo para mensajería instantánea XMPP.
- Investigar sobre el servidor *Openfire* para la gestión del sistema de mensajería.
- Proponer formalmente los cambios al protocolo XMPP para poder garantizar seguridad y eficiencia en la transmisión de datos.
- Aprender sobre el desarrollo de aplicaciones en el sistema operativo *Android* y *BlackBerry*.
- Construir el cliente de Mensajería instantánea para *Android*.
- Construir el cliente de Mensajería instantánea para *BlackBerry*.
- Verificar formalmente el código de los clientes de IM para *Android* y *BlackBerry*.
- Verificar formalmente los cambios realizados al protocolo XMPP.
- Documentar con los artefactos exigidos por DIVUX SYSTEMS durante todo el proceso de desarrollo de la solución.

Alcance

- **Autenticación**

  El sistema de mensajería instantánea verificará los datos del usuario cuando esté inicie sesión en la aplicación.

- **Mensajería**

  El usuario podrá establecer conversaciones individuales con sus contactos. No se contemplan salas grupales.

- **Perfil**

  El usuario podrá personalizar su perfil, a saber: podrá modificar su nombre, avatar, mensaje personal, su estatus de disponibilidad, su mensaje personal y su ubicación geográfica.

- **Documentación**

  Los documentos que se deben entregar al finalizar el Trabajo Instrumental de Grado son:

  - Especificación de Requisitos Software (ERS).
  - Documento de Especificación formal del Protocolo XMPP extendido con los cambios propuestos.
  - Documento con los Casos de Uso implementados durante el trabajo.
  - Código fuente debidamente documentado.
  - Generar automáticamente los JavaDocs de toda la solución desarrolladas, y colocarlos en el *GITORIUS* (sistema de repositorios) de DIVUX SYSTEMS.
  - DIVUX SYSTEMS podrá requerir cualquier otro artefacto o documento que considere necesario, sin que se afecten los tiempos de finalización del Trabajo Instrumental de Grado.

Limitaciones

- El desarrollo del proyecto está acotado a 4 meses, cumpliendo labores de 32 horas semanales.
- El desarrollo se realizará única y exclusivamente para sistemas operativos *Android* 2.3.3 *Gingerbread* (disponible en la versión 2.1 como mínima versión) y *BlackBerry* 5.0.
- Solo se transmitirá texto en las ventanas de conversación de la aplicación de mensajería instantánea.
- No se contemplan la creación de salas grupales de chat.
- No se contemplan pruebas de usabilidad de las aplicaciones móviles desarrolladas.

Justificación

En los últimos años, el incremento del uso de la telefonía celular en la sociedad venezolana como en las demás sociedades del mundo ha sido de forma exponencial, donde la innovación de nuevas tecnologías abarcan las más altas demandas del mercado mundial, bien sea en el diseño de equipos más sofisticados y el desarrollo de software de alta calidad enfocados en satisfacer las más exigentes necesidades de los usuarios.

En la actualidad las aplicaciones de Mensajería Instantánea (MI) para dispositivos móviles están divididas en 3 tipos, una son las que solo operan en una sola plataforma, es decir, que solo funcionan en un tipo de sistema operativo (Ej. ***BlackBerry Messenger***, que hoy en día solo está disponible en los equipos *BlackBerry*).

En segundo lugar tenemos las aplicaciones de integración de otras mensajerías instantáneas, estas están disponibles para distintos sistemas operativos móviles en algunos casos y se vale de otras plataformas de MI (Ej. ***Trillian*** es una aplicación que permite conectarse a *Windows Live Messenger, Google Talk, ICQ, AOL IM*, entre otros. En algunos casos es gratuita y en otros tiene un costo dependiendo del Sistema Operativo del dispositivo).

Finalmente, tenemos el tercer tipo, que son aquellas aplicaciones multiplataforma, es decir, operan en distintos sistemas operativos y cuentan con su propio sistema de mensajería (Ej. ***WhatsApp Messenger*** es una aplicación MI disponible en varios SO como *Android*, *BlackBerry*, *iPhone*, etc.).

Está aplicación se realizó para sistemas operativos *Android* y *BlackBerry*, lo que la convierte en una aplicación multiplataforma y de gran potencial, ya que la estructura y sistema en la que se desarrolló permite su extensión a otros dispositivos móviles como *iPhone*, *Windows Mobile, Java*

ME y *Symbian*, el cual es un objetivo a futuro de la empresa, está funcionalidad le permite al usuario interactuar con sus contactos sin depender del equipo que estos usen, siempre y cuando la aplicación esté disponible para dicho dispositivo y cuenten con una conexión de datos.

Además de permitir el intercambio de mensajes de texto, entre otros beneficios, ofrece al usuario el manejo de notificaciones en los mensajes de texto, de manera que un usuario puede saber si un mensaje fue enviado, entregado y leído, como también, le avisa si un contacto con el cual chatea está escribiendo un mensaje.

CAPITULO I. MARCO TEÓRICO

En esté capitulo se tratarán los fundamentos teóricos requeridos en el proyecto con el fin de orientar al lector en los aspectos conceptuales de los términos empleados.

I.1 Telefonía Móvil

Telefonía móvil se define como un "sistema de radiocomunicación que funciona en una zona dividida en células adyacentes, cada una de las cuales contiene una estación de transmisión-recepción radioeléctrica". (Definiciones, 2011)

Referente a la historia de la telefonía móvil tenemos que "Martin Cooper fue el pionero en esta tecnología, a él se le considera como *el padre de la telefonía celular* al introducir el primer radioteléfono en 1973 en los Estados Unidos mientras trabajaba para Motorola; pero no fue hasta 1979 en que aparece el primer sistema comercial en Tokio Japón por la compañía *Nippon Telegraph & Telephone Corp*". (Martinez, 2004)

Las redes de telefonía móvil se basan en el concepto de celdas (ver Figura 1), es decir zonas circulares que se superponen para cubrir un área geográfica.

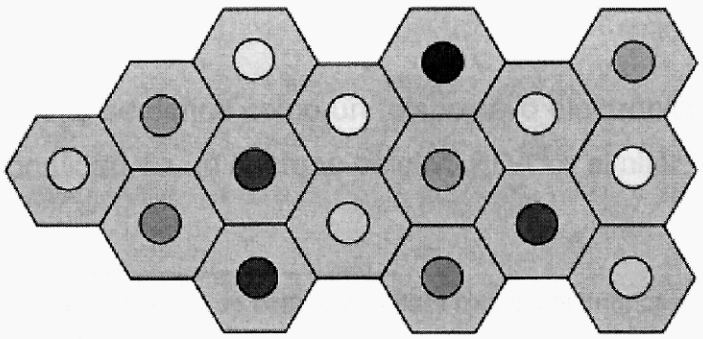


Figura 1. Modelo de Red Celular. (Kioskea, 2008)

Las redes celulares se basan en el uso de un transmisor-receptor central en cada celda, denominado "estación base" (o Estación base transceptora).

En una red celular, cada celda está rodeada por 6 celdas contiguas (por esto las celdas generalmente se dibujan como un hexágono). Para evitar interferencia, las celdas adyacentes no pueden usar la misma frecuencia. En la práctica, dos celdas que usan el mismo rango de frecuencia deben estar separadas por una distancia equivalente a dos o tres veces el diámetro de la celda. (Kioskea, 2008)

I.2 Teléfono Celular

Es un dispositivo inalámbrico electrónico que permite tener acceso a la red de telefonía celular o móvil. Se denomina celular debido a que el servicio funciona mediante una red de celdas, donde cada antena repetidora de señal es una célula, si bien también existen redes telefónicas móviles satelitales. Su principal característica es su portabilidad, que permite comunicarse desde casi cualquier lugar. La principal función es la comunicación de voz. (Wikipedia, 2011)

I.3 Teléfono Inteligente

También llamado *Smartphone*, es un teléfono que puede comunicarse a través de Wi-Fi[1], Bluetooth[2], conexión a Internet, envío de mensajería, e-mails (ver Figura 2).

Generalmente se define como un "dispositivo electrónico de mano que integra la funcionalidad de un teléfono celular, PDA[3] o similar. Generalmente

---

[1] Wi-Fi (Wireless Fidelity) Conjunto de estándares para redes inalámbricas basado en las especificaciones IEEE 802.11

[2] Bluetooth es una especificación tecnológica para redes inalámbricas que permite la transmisión datos mediante una radiofrecuencia segura (2,4 GHz)

se realiza añadiendo funciones de teléfono a un PDA existente o añadiendo funcionalidades "inteligentes", como las funciones del PDA, en un teléfono celular. Una característica clave de un *Smartphone* es que las aplicaciones adicionales pueden ser instaladas en el dispositivo. Las aplicaciones pueden ser desarrolladas por el fabricante del dispositivo, por el operador o por cualquier empresa desarrolladora de software". (GSMSpain, 2011)

Figura 2. Teléfonos Inteligentes. (WebAccion, 2011)

I.4 Mensajería Instantánea

La Mensajería Instantánea (MI) "permite conversar con otra gente mediante mensajes instantáneos y unirse a foros de debate (grupos MI) con temas específicos. Hay varios proveedores de servicios que mantienen servidores MI compatibles en los que puede iniciar sesión tras registrarse a un servicio MI. Puede que la compatibilidad con las funciones varíe según el proveedor de servicios". (Nokia, 2008)

Los Chat, ya sean privados o en grupo, son el origen de la MI. La gran innovación de la MI es el agrupamiento de todos estos sistemas en uno solo, cuyas principales características son la administración de contactos y conversaciones. Además, poco a poco van introduciendo mejoras como son

[3] PDA, acrónimo de Personal Digital Assitant (Asistente Digital Personal), es una computadora de mano. (PergaminoVirtual, 2011)

la presencia, transferencia de archivos e incluso conversaciones con voz y video.

La MI permite intercambiar mensajes de texto entre usuarios que previamente han aceptado comunicarse entre sí de esta manera (JabberEs, 2003). El procedimiento varía de un sistema a otro pero, en general, funciona así:

a. Un usuario se conecta a un servidor, en el cual está almacenada su lista de contactos, y se establece su estado: disponible, ocupado, ausente, etc.
b. El servidor le envía su lista de contactos y el estado de cada uno de ellos.
c. El servidor, automáticamente, informa de la presencia de este usuario a todos los usuarios de su lista de contactos que estén conectados en ese momento.
d. A partir de este momento, si un usuario quiere comunicarse con alguno de sus contactos, no tiene más que seleccionar el usuario deseado.
e. Para dar de alta un contacto en la lista de contactos, hace falta saber su dirección o alias y que el contacto autorice la inclusión.
f. Cuando el usuario cierra su programa de MI, el programa informa al servidor de la desconexión y éste, a su turno, notifica a todos los contactos.

I.5 Cliente de Mensajería Instantánea

Es una aplicación que opera en una plataforma de mensajería instantánea, estás aplicaciones se rigen bajo un protocolo que emplean para conectarse y comunicarse con el servidor de la plataforma.

Los clientes de mensajería instantáneas le permiten al usuario hacer uso de la sistema de chat, donde pueden ver su lista de contactos y conversar con ellos.

I.6 XMPP

Protocolo Extensible de Mensajería y comunicación de Presencia (*Extensible Messaging and Presence Protocol* por sus siglas en inglés), también conocido como *Jabber*, es un protocolo abierto basado en el estándar XML para el intercambio en tiempo real de mensajes y presencia entre dos puntos en Internet. (JabberEs, 2003)

El protocolo XMPP sigue la tan conocida arquitectura cliente/servidor. Los clientes XMPP únicamente se comunican con el servidor XMPP de su dominio, que es quien les ofrece el servicio. En los sistemas cliente/servidor, el cliente muestra información al usuario final y maneja sus peticiones, las peticiones del cliente son enviadas al servidor que ofrecerá unos determinados servicios. (Lage Serrano, 2005)

Las principales características del protocolo XMPP son:

- Es abierto: XMPP es un protocolo gratuito y de código abierto.
- Es extensible: como XMPP utiliza el lenguaje XML, lograr extender el protocolo para crear nuevas funcionalidades resulta ser sencillo.
- Fácil de Implementar: Debido a que el protocolo XMPP es abierto, cualquier persona es libre de instalar un servidor que implemente el protocolo sin ninguna dependencia por parte de un tercero.
- Es seguro: XMPP es un protocolo seguro, el cual soporta la implementación de SSL/TLS en las comunicaciones cliente/servidor. Además, puede implementar SASL en los procesos de autenticación de usuarios.

JID

Como todo sistema que gestiona usuarios, cada usuario en la plataforma XMPP debe tener un identificador único, para ello está el JID (Jabber ID). El JID está compuesto por el nombre de usuario (username) seguido de símbolo "@" y, finalmente le sigue el nombre del dominio del servidor, esta estructura lo hace similar a una dirección de correo electrónico. Por ejemplo: usuario@dominio.

Además de toda la arquitectura de dominios, Jabber/XMPP también tiene otras particularidades como puede ser la distinción de un usuario y una conexión de éste, llamada **resource**. Al contrario de otros sistemas de MI, en XMPP un usuario puede estar conectado simultáneamente a su servidor de dominio desde diferentes clientes de MI (PC, Smartphone, entre otros) pero con su único identificador XMPP (Jabber ID). Cuando se implementa el *resource* se debe agregar al JID el nombre del recurso separado por el símbolo "/". Por ejemplo: usuario@dominio/recurso.

Protocolos internos de XMPP

- **Protocolo Info/Query**

También llamado *IQ*, es un protocolo sencillo y ampliable de petición-respuesta que permite a los usuarios hacer peticiones, y almacenar o cambiar datos. *IQ* es simplemente el conductor de esas peticiones y respuestas, y gestiona qué datos son necesarios de acuerdo con las necesidades particulares de cada servidor.

Extensiones dentro del protocolo IQ:

- **Registro - IQ/Register:** Permite crear cuentas de usuario, actualizar la información de las cuentas de usuario y eliminar cuentas de usuario.

- **Autenticación - IQ/Auth:** Es uno de los protocolos únicamente dedicados a la seguridad en XMPP, permite al usuario demostrar a su servidor que son quien dice ser.

- **Contactos - IQ/Roster:** Este protocolo permite la suscripción entre contactos lo que permite establecer las relaciones de amistad.

- **Información Personal - IQ/vCard:** Las *vCards* son fichas que poseen la información personal del usuario.

A través de esta extensión es permitido crear, modificar y eliminar la *vCard* del usuario.

- **Protocolo Presence**

XMPP tiene la posibilidad de que los usuarios puedan detectarse los unos a los otros y saber cuando un usuario se encuentra conectado. Además hoy en día se va más allá y podemos saber si aunque esté conectado al sistema de MI en ese instante está ocupado o ausente. Además XMPP nos ofrece la posibilidad de poner tantos estados y mensajes personales diferentes como desee el usuario, así siempre se sentirá más identificado con el estado en el que pone su sistema de MI.

- **Protocolo Message**

Los mensajes son la parte más importante de la MI. El envío de mensajes es la mayor responsabilidad de cualquier sistema XMPP.

El protocolo de mensajes es extremadamente sencillo, los paquetes son enviados por un usuario a un receptor. Por defecto, no se reciben confirmaciones de que el mensaje ha sido recibido por el destinatario para reducir el tráfico en el servidor, además si

el receptor no se encuentra disponible, el servidor guardará el mensaje hasta que se conecte, a este protocolo se le llama "*store and forward*". (Lage Serrano, 2005)

Notificaciones Oficiales de estados de mensajes en XMPP

- **Mensaje Entregado**

El protocolo para las notificaciones de mensajes entregados se encuentra definido en la extensión ***XEP-0184: Message Delivery Receipts***, en donde se define que un usuario que quiera que sus mensajes sean notificados como entregados deberá enviar la solicitud de entrega dentro del cuerpo del paquete *Message*, de manera que el contacto que recibe el mensaje al recibir la solitud deberá enviar un paquete *Message* sin mensaje, en donde notifica el *ID* del mensaje que recibió.

Notificaciones Oficiales de estados de chat en XMPP

- **Composing y Paused**

Entiéndase por *Composing* como el evento que notifica que el contacto con el cual el usuario conversa se encuentra escribiendo un mensaje y *Paused* al evento que notifica que el contacto ha dejado de escribir el mensaje.

I.7 Openfire

Es un servidor bajo licencia de GPL[4] que trabaja en tiempo real e implementa el protocolo de comunicación XMPP, por lo que el servidor es empleado para establecer un sistema de MI.

---

[4] GPL (General Public License) es una licencia orientada principalmente a proteger la libre distribución, modificación y uso de software.

Openfire está desarrollado en Java y cumple tareas como administrar usuarios, mensajería instantánea, compartir archivos, auditar mensajes, mensajes offline, mensajes broadcast, salas de conversación grupal, entre muchas otras opciones.

I.8 Android

Android se define como "un sistema operativo orientado a dispositivos móviles, basado en una versión modificada del núcleo Linux. Inicialmente fue desarrollo por Android Inc., una pequeña empresa, que posteriormente fue comprada por Google; en la actualidad lo desarrollan los miembros de *Open Handset Alliance* (liderada por *Google*)". (Concepto Definición, 2011)

Android es un sistema abierto, multitarea, que permite a los desarrolladores acceder a las funcionalidades principales del dispositivo a través de aplicaciones, estas aplicaciones pueden ser desarrolladas libremente por terceros mediante herramientas de desarrollo que brinda *Google* empleando Java como lenguaje de programación.

Arquitectura del Sistema Operativo Android

Android se encuentra compuesto por 5 elementos, donde en pocas palabras, en base a un kernel *Linux* y un colección de librerías C/C++ están expuestas a través de una estructura de aplicaciones que proporciona servicios y gestión del tiempo de ejecución y de las aplicaciones (Ver Figura 3).

a. Linux kernel

Servicios básicos (incluidos los controladores del hardware, procesamiento y administración de memoria, seguridad, redes y la administración de gestión). El kernel también proporciona una capa de abstracción entre el dispositivo y el resto de la estructura del sistema operativo.

b. Librerías

Ejecutadas sobre el kernel, *Android* incluye un conjunto de librerías de C/C++ que permiten realizar tareas como reproducción de audio y vídeo, gestión de pantalla, procesamiento de gráficos 3D y 2D a través de SGL y OpenGL, gestión de bases de datos nativas mediante SQLite, SSL y WebKit para el navegador Web integrado y la seguridad en Internet, entre otras.

c. Runtime de Android

Incluyendo las librerías del núcleo y la máquina virtual de *Dalvik*, el tiempo de ejecución *Android* es el motor que impulsa la aplicación y, junto con las librerías, es la base para el Framework de Aplicaciones.

- **Librerías Básicas:** Las librerías básicas de *Android* proporcionan la mayor parte de las funcionalidades disponibles en las librerías básicas de Java, así como las librerías específicas de Android.

- **Dalvik:** Es una máquina virtual basada en el kernel de Linux para la gestión de hilos y memoria a bajo nivel.

d. Framework de Aplicaciones

Provee las clases utilizadas para crear aplicaciones *Android*. Además, proporciona una abstracción genérica para el acceso al hardware y gestionar la interfaz de usuario y recursos de las aplicaciones.

e. Aplicaciones

Todas las aplicaciones, tanto nativas como hechas por terceros, se encuentran sobre la capa de aplicaciones por medio del mismo API

de librerías. La capa de aplicación se ejecuta dentro del Runtime de *Android*, usando las clases y servicios disponibles en el Framework de Aplicaciones.

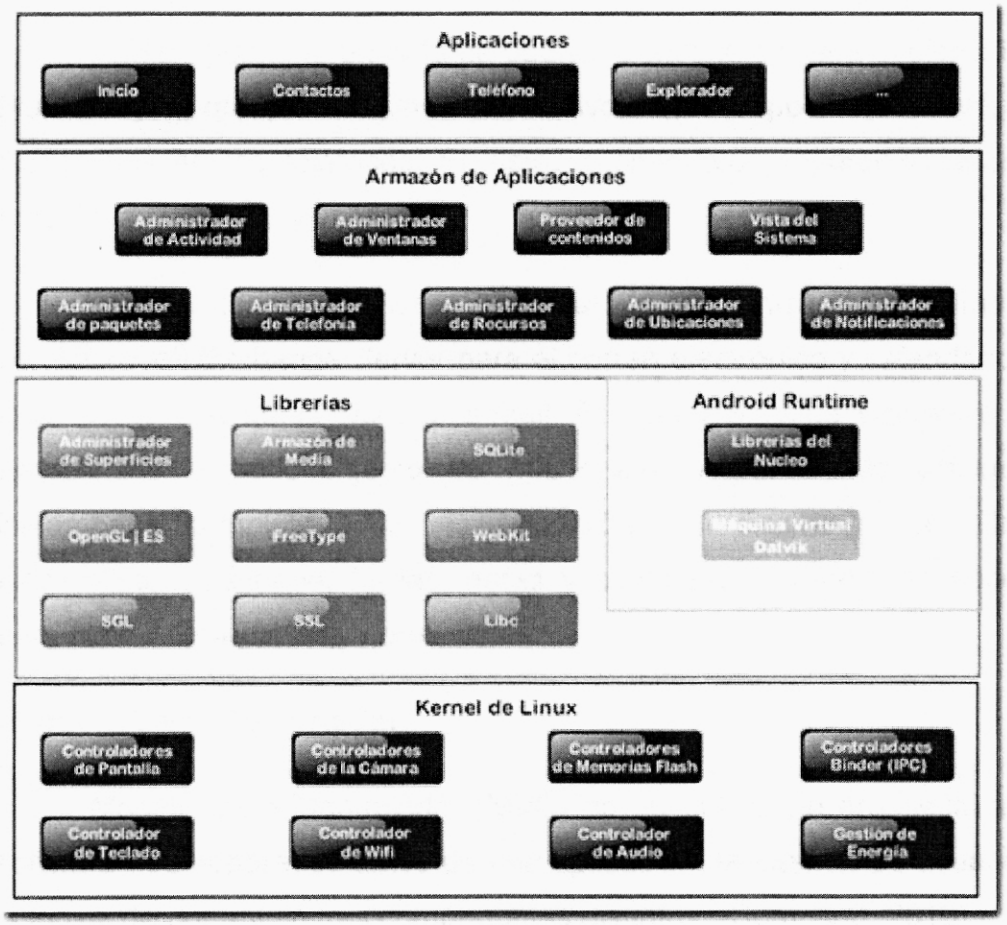


Figura 3. Arquitectura del Sistema Operativo *Android*. (Android SO, 2011)

I.9 BlackBerry

Es una línea de teléfonos inteligentes (*Smartphones*) desarrollado por la compañía canadiense *Research In Motion* (RIM). *BlackBerry* funciona como un asistente personal con libretas de direcciones, calendario, notas, emails, reproductor de video y audio, cámara, etc.

BlackBerry es fundamentalmente conocido por su capacidad para enviar y recibir mensajes instantáneos y correos electrónicos de Internet

accediendo a las redes móviles de compañías de teléfono celular que brindan este servicio.

- **Sistema Operativo**

RIM proporciona un sistema operativo multitarea para el dispositivo *BlackBerry*, lo que permite un uso intensivo de los dispositivos de entrada disponibles en los teléfonos. El sistema operativo proporciona soporte para Java MIDP 1.0 y WAP 1.2.

Las versiones anteriores permitían la sincronización inalámbrica con Microsoft Exchange Server para el correo electrónico y calendario, al igual como con Lotus Domino e-mail. El actual Sistema Operativo 5.0 proporciona un subconjunto de MIDP 2.0, y permite la activación inalámbrica completa y la sincronización con Exchange de correo electrónico, calendario, tareas, notas y contactos, y añade un soporte para *Novell GroupWise* y *Lotus Notes*.

I.10 Modelo Vista Controlador

Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. (Comusoft, 2010)

- **Modelo:** El modelo es la representación específica de la información con la cual el sistema opera, en donde se accede a la capa de persistencia (Almacenamiento de Datos) y se definen las reglas de negocio que rigen la aplicación.

  En el desarrollo de las aplicaciones el componente modelo se encuentra en el paquete *model*.

- **Vista:** Está conformada por el conjunto de pantallas (vistas) que son empleadas para la interfaz gráfica de la aplicación con la finalidad de

interactuar con el usuario en los procesos de entrada y salida de datos.

En el desarrollo de las aplicaciones el componente vista se encuentra en el paquete *ui*.

- **Controlador:** El controlador posee las reglas que gestionan los eventos en la aplicación para tomar decisiones en base a los datos recibidos a través de la vista.

  En el desarrollo de las aplicaciones el componente controlador se encuentra en el paquete *controller*.

En la arquitectura MVC el modelo debe estar aislado, por lo que no debe interactuar con el controlador ni con la vista, caso contrario, la vista implementa el modelo para representarlo de manera gráfica y el controlador coordina los eventos respecto al modelo y la vista (ver Figura 4).

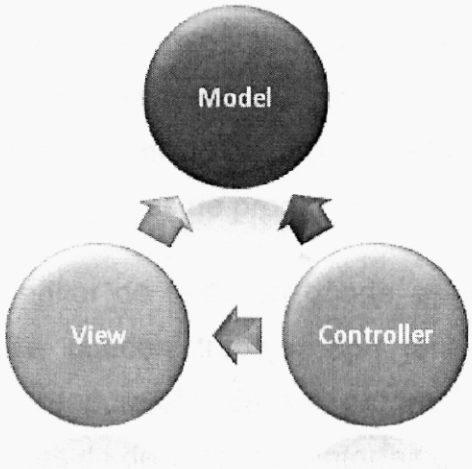


Figura 4. Patrón de Arquitectura de Software Modelo Vista Controlador

I.11 DAO

Objeto de Acceso a Datos (*Data Acces Object* por sus siglas en inglés) es un patrón de diseño que proporciona una técnica para la separación de la persistencia de objetos y la lógica de acceso a datos de cualquier mecanismo de persistencia en particular o API. (CodeFutures, 2011)

Existen claros beneficios de este enfoque desde una perspectiva arquitectónica. El enfoque de DAO proporciona flexibilidad para cambiar de un mecanismo de la aplicación de la persistencia en el tiempo sin la necesidad hacer re-ingeniería en la lógica de la aplicación que interactúa con el objeto de acceso a nivel de datos.

En el desarrollo de las aplicaciones el objeto DAO se encuentra en el paquete *dao* bajo el nombre de *DataBase* y se encarga de interactuar con la base de datos de la aplicación para la persistencia de los usuarios mediante *SQLite* (ver Recursos).

I.12 IDE

Un Ambiente de Entorno Integrado, también conocido como IDE (*Integrated Development Environment* por sus siglas en inglés) es un programa en el cual le es permitido al programador desarrollar diversas aplicaciones en diferentes lenguajes de programación.

Un IDE tiene un editor de texto en donde es introducido el código fuente del programa, se pueden trabajar con lenguajes compilados o interpretados, permite la depuración de los programas y por lo general cuentan con funcionalidades de desarrollo de interfaz gráfica.

I.13 Java

Es un lenguaje de programación desarrollado por *Sun Microsystems, Inc.* en 1991. Elaborado bajo una estructura C/C++ Java

cuenta con cinco (5) pilares fundamentales: la programación orientada a objetos, la posibilidad de ejecutar un mismo programa en diversos sistemas operativos, la inclusión por defecto de soporte para trabajo en red, la opción de ejecutar del código en sistemas remotos de manera segura y la facilidad de uso.

Lo habitual es que las aplicaciones Java se encuentren compiladas en un bytecode[5], aunque también pueden estar compiladas en código máquina nativo.

I.14 XML

Lenguaje de Marcas Extensible (*eXtensible Markup Language* por sus siglas en inglés) es un metalenguaje que permite definir lenguajes de marcado adecuados a usos determinados.

En la práctica corresponde a un estándar que permite a diferentes aplicaciones interactuar con facilidad a través de la red. XML fue creado por *World Wide Web Consortium* (W3C), partiendo de las amplias especificaciones de SGML[6]. Su desarrollo se comenzó en 1996 y la primera versión salió a la luz el 10 de febrero de 1998. La primera definición que apareció fue: Sistema para definir validar y compartir formatos de documentos en la Web. (Master Magazine, 2004)

---

[5] Bytecode es un fichero binario que tiene un programa ejecutable.

[6] SGML (Standard Generalized Markup Language) es un lenguaje que permite la organización y el etiquetado de documentos

La metodología a aplicar durante el ciclo de vida del proyecto se basa en *Mobile-D*, la cual es una metodología ágil, derivada de las metodologías ***eXtreme Programming (XP),*** metodología ***Crystal*** *y* ***Rational Unified Process (RUP)***. Por lo que *Mobile-D* hereda los principios de *XP* como lo son la simplicidad, comunicación, retroalimentación, confianza y respeto en las decisiones de desarrollo; de *Crystal* se determina el orden, la asignación de responsabilidades y las comunicaciones del grupo de trabajo; y finalmente, de la metodología *RUP* recibe los fundamentos del desarrollo de documentos de cada una de las etapas del ciclo de vida del proyecto.

Etapas de la metodología *Mobile-D*, también se puede presenciar el ciclo de vida de la metodología en la Figura 5:

II.1 Exploración

La fase de exploración, se dedica al establecimiento de un plan de proyecto y los conceptos básicos. Por lo tanto, se puede separar del ciclo principal de desarrollo. Los autores de la metodología ponen además especial atención a la participación de los clientes en esta fase.

II.2 Inicialización

Se preparan e identifican todos los recursos necesarios. Se elabora la planificación de las siguientes etapas y se establece el entorno técnico (incluyendo la capacitación del equipo de desarrollo). La contribución al desarrollo ágil se centra en esta etapa, ya que se investigan y verifican los puntos fundamentales del desarrollo, brindando suficiente soporte a las etapas subsiguientes.

II.3 Desarrollo

En esta etapa se lleva a cabo un proceso de desarrollo iterativo e incremental, hasta implementar todas las funcionalidades. Se organiza el

trabajo en Iteraciones: cada iteración se planifica en términos de requisitos y tareas a realizar.

Ahora bien, este Trabajo Instrumental de Grado, está enmarcado dentro de un proceso de desarrollo de software formal, esto quiere decir que se emplearán técnicas, métodos y mecanismos de verificación formal del sistema, en sus distintas fases.

II.4 Estabilización

Se llevan a cabo las últimas acciones de integración para asegurar que el sistema completo funciona correctamente. En esta etapa todo el esfuerzo se dirige a la integración del sistema.

II.5 Pruebas

La etapa de prueba y reparación del sistema tiene como meta la disponibilidad de una versión estable y plenamente funcional del sistema. El producto terminado e integrado se prueba con los requisitos de cliente y se eliminan todos los defectos encontrados.

Las pruebas ayudarán a validar el sistema, para lo cual se podrán construir los casos de pruebas unitarias y funcionales necesarias.

Figura 5. Ciclo de Vida del Desarrollo de Mobile-D

CAPITULO III. RECURSOS

III.1 Android SDK 10

Kit de Desarrollo de Software o SDK (*Software Development Kit* por sus siglas en inglés) es un software que permite a los desarrolladores crear aplicaciones para sistemas operativos *Android*. El SDK de *Android* incluye proyectos de muestra con código abierto, herramientas de desarrollo, emulador de dispositivos móviles y las librerías necesarias para elaborar aplicaciones *Android*. Las aplicaciones se elaboran usando el lenguaje de desarrollo Java y son ejecutadas sobre *Dalvik*, el cual es una maquina virtual diseñada para el uso integrado que se ejecuta sobre un kernel *Linux*.

Para el desarrollo de la aplicación de mensajería instantánea *Android* se empleó la versión 6 como la mínima disponible para la aplicación, la cual es empleada en sistemas operativos Android 2.1.

III.2 BlackBerry SDK 5.0

El SDK de *BlackBerry* provee el conjunto de herramientas y programas que permiten al desarrollar construir aplicaciones para dispositivos *BlackBerry*, además, incluye el simulador de dispositivos, códigos de muestra y documentaciones. Las aplicaciones desarrolladas en *BlackBerry* están basadas en el lenguaje de programación Java.

III.3 Smack API 3.1.0

Es una librería *Open Source* basada en el protocolo XMPP para cumplir funciones de cliente de mensajería instantánea y presencia. Elaborada en el lenguaje de programación *Java*, esta librería puede ser implementada en las aplicaciones para crear desde un cliente XMPP completo a una simple integración XMPP, tales como el envío de mensajes de notificación y la presencia de los dispositivos. (Ignite Realtime, 2011)

Smack debió ser modificada en ambas aplicaciones móviles para poder operar en la plataforma *Android* y *BlackBerry*.

III.4 SQLite

SQLite es una librería de software que implementa un sistema autónomo, sin servidor, sin necesidad de configuración, motor de base de datos transaccional SQL. *SQLite* lee y escribe directamente en los archivos de disco normal. Una base de datos SQL completa con varias tablas, índices, triggers y vistas, está contenida en un solo archivo. El formato de archivo de base de datos es multi-plataforma. (SQLite, 2011)

SQLite es implementada en las aplicaciones para gestionar los datos de los usuarios que emplean las aplicaciones móviles para *Android* y *BlackBerry*.

CAPITULO IV. DESARROLLO

El presente capítulo tiene por finalidad explicar en su totalidad el proceso de planificación, diseño y elaboración de las aplicaciones de mensajería instantánea para dispositivos móviles Android y BlackBerry en función a la metodología aplicada.

El desarrollo del proyecto se fundamenta en el uso de librerías que brinden soporte a la aplicación cliente en el manejo del protocolo XMPP, facilitando así el proceso de comunicación con el servidor *Openfire*. Como también, el desarrollo de funcionalidades que gestionen los distintos eventos y acciones dentro de la aplicación.

La etapa de desarrollo dentro de la metodología ***Mobile-D*** se compone de cinco (5) etapas (ver Marco Metodológico) las cuales son explicadas a continuación:

IV.1 Etapa de Exploración

IV.1.1 Identificación y Análisis de Requerimientos

Luego de realizar continuas reuniones con la directiva de DIVUX SYSTEMS se delimitó el proyecto, donde se identificaron los requerimientos funcionales de la aplicación tanto en su versión para *Android* como para *BlackBerry*. Finalmente, los requerimientos obtenidos fueron:

- Las aplicaciones deberán realizar los procesos de autenticación de los datos del usuario para iniciar sesión.
- El usuario podrá establecer conversaciones de carácter individual con sus contactos, por lo que se omite las conversaciones en conferencia.
- El usuario podrá personalizar su perfil.

Además, se elaboró un documento en el cual están plasmados los casos de uso de la aplicación.

IV.2 Etapa de Inicialización

IV.2.1 Investigación

Se adquirieron conocimientos referentes a las herramientas a utilizar y se realizó una preparación para llevar a cabo el proceso de desarrollo de las aplicaciones. El proceso de investigación contó con las siguientes tareas:

- **Introducción a XMPP**

Se debió realizar una investigación sobre las principales funcionalidades y procesos de comunicación del protocolo XMPP con la finalidad de adquirir conocimientos y poseer una visión clara de las herramientas que se están empleando en el proyecto.

Accediendo a sitios oficiales y no oficiales se recolecto información referente al protocolo, abarcando puntos de carácter teórico como también referente a los procesos prácticos, la sintaxis empleada en los paquetes durante el intercambio de información, entre otros.

- **Introducción a Openfire**

Mediante una investigación concreta de las funcionalidades que posee el servidor se adquirieron los conocimientos necesarios para comprender su operatividad y la manera en que gestiona la base de datos que implementa.

IV.2.2 Diseño

El proyecto fue divido en cuatro (4) módulos:

- **Módulo de Autenticación**

  Encargado de coordinar los procesos de Inicio/Cierre de sesión a través de la autenticación del usuario.

- **Módulo de Contactos**

  Este módulo gestiona los contactos del usuario, listándolos en dos grupos, los contactos que están Conectados y los que están No Conectados.

  Además maneja los eventos que cada contacto puede generar, como lo son los cambios de presencia o cambios de los datos de su cuenta (Nombre, Mensaje Personal y Avatar).

- **Módulo de Mensajería Instantánea**

  Tiene la finalidad de gestionar los procesos de mensajes instantáneos, coordinando las tareas de enviar mensajes como también de manejar los eventos de mensajes recibidos.

- **Módulo de Perfil de Usuario**

  Maneja los datos del usuario, permitiendo así modificar datos personales (Nombre Completo, Correo Electrónico, Teléfono, Fecha de Nacimiento y Ubicación).

IV.2.3 Capacitación

- **Ambientes de Desarrollo**

Consistió en la preparación de los ambientes de desarrollo para *Android* y *BlackBerry*, el **IDE** (ver Marco Teórico) utilizado fue Eclipse, Android SDK 10 y BlackBerry SDK 5.

- **Aprender sobre el desarrollo Android y BlackBerry**

Con la finalidad de adquirir conocimientos y práctica acerca del desarrollo en sistemas operativos *Android* y *BlackBerry*, se investigó en Internet información referente a guías para desarrolladores a través de sitios oficiales y no oficiales de *Android* y *BlackBerry*. También fue fundamental haber participado en las electivas de Desarrollo de Aplicaciones *BlackBerry* y Desarrollo de Aplicaciones *Android* para adquirir una práctica y docencia apropiada.

Se abarcaron puntos para el manejo de conexiones, intercambio de datos en un sistema *cliente-servidor*, gestión de componentes gráficos, coordinación de eventos en el dispositivo, entre otras funcionalidades que influyen el desarrollo de las aplicaciones.

IV.3 Etapa de Desarrollo

Basado en iteraciones, cada iteración se encuentra subdividida en tres puntos fundamentales como lo son la planificación de la iteración, el trabajo y la liberación o lanzamiento de la iteración. En este proyecto se empleó un número de cuatro (4) iteraciones, las cuales son explicadas a continuación.

IV.3.1 Primera Iteración: Aplicación Android/BlackBerry – Inicialización y Conexión

La iteración inicial del proyecto se encuentra enfocada en el proceso de establecer conexión con el servidor, el cual es el encargado de gestionar la plataforma de mensajería instantánea.

Esta iteración requiere el desarrollo de una aplicación para dispositivos móviles *Android* con sistema operativo 2.3.3 Gingerbread bajo la implementación del min. SDK 6 para que pueda operar en versiones 2.1 y 2.2, y una aplicación para dispositivos móviles *BlackBerry* con sistema

operativo 5.0. Ambas aplicaciones deberán establecer conexión con el servidor vía TCP o WIFI cumpliendo el protocolo de comunicación XMPP.

a. Planificación

Las tareas a realizar para esta iteración son:

- Iniciar desarrollo de las aplicaciones para *Android/BlackBerry*
- Establecer conexión con el servidor *Openfire*
- Procesos de Inicio y Cierre de Sesión (*Login y Logout*)
- Recordar Usuario y Contraseña
- Diseño y desarrollo del entorno gráfico (*Look & Feel*)

b. Trabajo

- **Iniciar desarrollo de las aplicaciones para *Android/BlackBerry***

Se preparó el ambiente de desarrollo óptimo para elaborar las aplicaciones móviles para *Android* y *BlackBerry*.

Para el manejo del protocolo XMPP dentro de las aplicaciones móviles se hizo uso de ***Smack***, una librería *Open Source* basada en Java la cual se encarga de gestionar en su totalidad las principales funciones del protocolo. Vale destacar que el código fuente de *Smack* fue sometido a diversos cambios a lo largo del proyecto, principalmente para permitir su compatibilidad en *Android* y *BlackBerry* y segundo, para añadir nuevas funcionalidades y realizar modificaciones en sus procesos de trabajo.

El proyecto se encuentra estructurado bajo una arquitectura **Modelo Vista Controlador** (ver Marco Teórico) lo cual permitió el desarrollo de código organizado que favorece a su optimización y escalabilidad. El núcleo de la aplicación se encuentra en los

paquetes ***controller*** y ***org,*** ya que éstos administran todo el cliente XMPP, gestionan la lógica y distribución adecuada de las funcionalidades. Dentro de *controller* se encuentran clases llamadas ***listeners***, cuyos métodos son invocados cuando se reciben paquetes por parte del servidor y que requieren un tipo de trato en específico. También se entra en contacto con aplicaciones y funcionalidades nativas del dispositivo móvil como puede ser el caso de notificaciones, gestión de bases de datos internas, editar valores de la configuración del equipo, administrar el uso de la red y las conexiones disponibles, entre otras.

Respecto al entorno gráfico, todas las clases se encuentran incorporadas dentro del paquete ***ui***, en este paquete se gestionan todas las vistas que son usadas dentro de la aplicación móvil. Se emplean pantallas para Iniciar Sesión, Lista de Contactos, Lista de Conversaciones abiertas, Pantalla de Conversación y Perfil de Usuario. Las pantallas se desarrollaron lo más idéntico posible tanto para *Android* como para *BlackBerry*.

El modelo esencial de la aplicación se base en 4 clases principales, ***User, Presence, IQ*** *y* ***Message*** (ver Apéndice A), las cuales definen la estructura de los objetos primordiales que son empleados en toda la aplicación.

- **Establecer conexión con el servidor Openfire**

Los teléfonos inteligentes *BlackBerry* permiten varios tipos de conexiones, y esto requiere el desarrollo de funcionalidades en para cada tipo de conexión, para el caso aplicación se podrá establecer conexión a través de **Direct TCP**, **WIFI**, **BES** y **BIS**. Los teléfonos *Android* también permiten varios tipos de conexiones a Internet, pero

el sistema operativo gestiona las conexiones de forma automática, se encarga de ver cuáles son los tipos de conexiones existentes (Red Celular, WIFI, etc.) y establecer la conexión por cualquiera de ellas. La conexión con el servidor se realiza a través del manejo de sockets, empleando la clase ***Socket*** de Java y utilizando la clase ***XMPPConnection*** de *Smack*, en donde es posible establecer la conexión con el servidor y permitir el intercambio de datos.

La clase ***Application*** administra los aspectos principales de la aplicación y el dispositivo móvil, esta clase posee constantes globales que poseen los datos del servidor y el puerto, en el caso del servidor se hace uso del ***host im.divux.com*** y por estándar del protocolo XMPP el **puerto** a usar es el ***5222***.

Luego de iniciar el socket, se procede a enviar un paquete que posee la etiqueta XML ***<stream:stream>***, ante esta acción se espera una respuesta por parte del servidor, si se recibe un paquete que posee la etiqueta *<stream:stream>* la aplicación ha establecido conexión de forma exitosa con el servidor, en caso contrario, si el contenido del paquete tiene la etiqueta *<stream:error>*, la conexión con el servidor no ha podido ser establecida.

- **Procesos de Inicio y Cierre de Sesión (*Login y Logout*)**

La clase *Application* también posee otra constante importante llamada ***resource*** (ver Marco Teórico). Cuando se procede a realizar el inicio de sesión, los datos del usuario son enviados al servidor junto con el *resource* para que este gestione la conexión.

Security es una clase empleada para el proceso de autenticación del usuario, a través de esta clase se obtiene la clave encriptada del usuario en **MD5**, esta clave es la que realmente es

enviada al servidor como medida de seguridad e integridad de la cuenta del usuario.

La conexión con el servidor se hace a través de *Smack*, que es la base del protocolo XMPP en nuestro proyecto, para iniciar sesión nos basamos en los paquetes del tipo *IQ/Auth*, en este paquete esta apuntado nuestro nombre de usuario (*username*), recurso y contraseña encriptada. La autenticación se lleva a cabo mediante un paquete que es generado en la clase *PacketWriter* en donde va contenida la información de *login* del usuario. En caso de iniciar sesión exitosamente se mostrará una pantalla (por ahora es una pantalla de prueba).

El proceso de cerrar sesión se compone en dos partes, donde inicialmente enviamos un paquete ***Presence*** donde notificamos que nuestro estado es "No Conectado" o "*Unavailable*" y finalmente enviamos un paquete *IQ* donde indicamos la finalización de nuestra sesión con una etiqueta ***</stream:stream>***.

- **Recordar Usuario y Contraseña**

Se implementó una pequeña base de datos basada en ***SQLite***, en donde se guarda la información del usuario como nombre de usuario y contraseña, en la cuarta iteración, el modelo de la base de datos es modificado para almacenar más datos del usuario. Esta base de datos es administrada a través de la clase ***DataBase*** del paquete ***dao***, en esta clase son ejecutados los métodos de persistencia como insertar, actualizar, buscar y eliminar.

A través de la interfaz gráfica, el usuario le podrá otorgar los permisos a la aplicación para que sus datos sean almacenados, bien

sea el nombre de usuario y contraseña o sólo el nombre de usuario. Esta funcionalidad le permite al usuario un acceso rápido a la plataforma de mensajería instantánea ya que evita que la persona que usa la aplicación ingrese sus datos cada vez que inicie sesión.

- **Diseño y desarrollo del entorno gráfico (*Look & Feel*)**

LoginScreen es una vista sencilla que posee los campos para que el usuario introduzca el nombre de usuario y la contraseña, seguido de un botón que invoca la acción de iniciar sesión. Además cuenta con dos *checkboxes*, que le permiten al usuario confirmar la opción de guardar nombre de usuario (*username*) y contraseña (*password*) respectivamente (ver Apéndice B).

ControlLogin, el controlador de *LoginScreen*, posee las funcionalidades requeridas para llevar a cabo los procesos de iniciar sesión e implementar la clase *DataBase* para guardar los datos del usuario a través de los datos que son obtenidos mediante la interfaz.

c. Liberación

- **Integración**

Las clases desarrolladas fueron distribuidas bajo la arquitectura empleada para el desarrollo del proyecto, la integración de los módulos coloca a los controladores como los puntos de administración y comunicación entre las vistas correspondientes y las clases de la librería *Smack* que gestionan el protocolo XMPP.

- **Pruebas de Aceptación**

- **Iniciar/Cerrar Sesión**

La prueba se basa en la evaluación de los procesos de inicio y cierre de sesión, verificando que la aplicación se conecte

con el servidor y que la autenticación del usuario se realice correctamente. Se ejecutaron diversas pruebas de inicio de sesión en las que se utilizaron cuentas ya existentes en el servidor y los resultados obtenidos fueron positivos. También se intentó iniciar sesión con cuentas no registradas y la aplicación hizo uso correcto de las excepciones y avisos.

Luego de obtener resultados positivos de las pruebas de Inicio de Sesión se prosiguió a verificar que la aplicación cierre sesión correctamente cumpliendo el proceso establecido por el protocolo XMPP, tomando en cuenta que el servidor cierre la sesión del usuario y que la aplicación móvil finalice la conexión y cierre la aplicación en el dispositivo. Finalmente, los resultados obtenidos fueron exitosos, logrando así, que la aplicación cumpla con uno de los requerimientos principales, poder cerrar sesión.

- **Recordar Usuario y Contraseña**

Esta prueba consta de verificar que la aplicación almacene los datos del usuario con respecto a su nombre de usuario y contraseña, para ello, se inició sesión y se seleccionaron las opciones que otorgan los permisos de almacenar los datos a la aplicación. Luego de realizar varias pruebas se obtuvieron resultados favorables.

- **Recordar Usuario**

Dividida en dos partes, esta prueba tiene la finalidad de que solo sea almacenado el *username* del usuario que inicia sesión. Está dividida en dos partes ya que si no existe un registro en la base de datos que esté asociado al usuario, se inserta un registro con un valor nulo en el campo de contraseña, pero también está la opción de que dicho usuario ya esté registrado

en la base de datos, si éste posteriormente desea que la aplicación no recuerde su contraseña, la aplicación deberá actualizar su registro y asignar un valor nulo a su contraseña.

Se realizaron pruebas para ambos casos, en donde se almacenó un nuevo usuario sin contraseña y también se actualizó la cuenta del usuario que tenía almacenado username y password, indicándole a la aplicación que ahora sólo debía recordar el nombre de usuario. En los dos casos evaluados se verificó que la funcionalidad opera correctamente.

- **Olvidar Usuario**

Si un usuario desea que la aplicación no recuerde su username y password, el programa deberá inspeccionar los usuarios almacenados en la base de datos y eliminar el registro correspondiente. De esta forma se garantiza que no habrá data referente al usuario que usa la aplicación. Los resultados obtenidos al probar esta funcionalidad fueron positivos.

IV.3.2 Segunda Iteración: Aplicación *Android/BlackBerry* – Lista de Contactos

La segunda iteración tiene por finalidad listar los contactos del usuario, en este punto hay que destacar que la base de datos que posee el servidor cuenta con usuarios y relaciones de contacto ya establecidas a través de la aplicación Web. Para está iteración se requiere crear métodos que gestionen los paquetes *IQ* destinados al manejo de ***Roster***, ***vCards*** (datos personales de los usuarios) y ***Presence*** (disponibilidad y mensajes personales).

Mediante los paquetes ***IQ/Roster*** se obtiene el listado de los contactos del usuario, pero éste no es suficiente ya que no indica el estado

de disponibilidad de cada uno así como todos sus datos personales. Una vez que la aplicación ha iniciado sesión comenzará a recibir paquetes Presence por parte del servidor en donde se indican los estados y mensajes personales de los usuarios. Finalmente, se solicita al servidor los datos personales de cada usuario (Nombre Completo, Número Telefónico, Correo Electrónico, Ubicación y Avatar) a través de paquetes ***IQ/vCard***.

a. Planificación

Las tareas para esta iteración son:

- Obtener listado de contactos (*Roster*)
- Obtener presencias y estados de los contactos (*Presence*)
- Obtener datos personales de los contactos (*vCard*)
- Diseño y desarrollo del entorno gráfico (*Look & Feel*)

b. Trabajo

- **Obtener listado de contactos (*Roster*)**

En el modelo de la aplicación se desarrolló ***User***, una clase que cuenta con los atributos principales de un contacto, como lo son su nombre completo, número telefónico, correo electrónico, ubicación y avatar, también cuenta con un atributo de identificación, que en el caso de XMPP corresponde al **JID** (ver Marco Teórico).

La clase ***ContactAdministrator*** gestiona toda la información en relación a la lista de contactos, se encarga de obtener y modificar los contactos, los contactos son manejados mediante un arreglo de objetos *User*, *ContactAdministrator* también gestiona los eventos requeridos por la aplicación cuando un contacto ha cambiado sus datos o el estado.

El proceso de obtención de la lista de contactos se lleva a cabo una vez que el usuario haya sido autenticado, cuando esto ocurre se envía un paquete *IQ/Roster* al servidor, para ello se crea una instancia de la clase ***RosterPacket*** de *Smack* y obtenemos un objeto modelo del paquete indicándole que sea del tipo *get* (para obtener respuesta del servidor), al recibir un paquete *IQ/Roster* por parte del servidor se verifica el tipo, en caso de ser *result* el XML es analizado mediante ***XmlPullParser*** y se obtiene el listado de usuario los cuales son almacenados en arreglo de contactos que posee *ContactAdministrator*. En caso de que el tipo sea *error*, la solicitud no fue procesada correctamente.

- **Obtener presencias y estados de los contactos (*Presence*)**

A través de la clase *PacketReader* de *Smack* se detectan los paquetes que recibe la aplicación por parte del servidor, ya sean del tipo *IQ*, *Presence* o *Message*, estos paquetes son analizados mediante métodos de la clase *XmlPullParser* para inspeccionar su contenido y, dependiendo del tipo de paquete se ejecutará una acción determinada. Esta acción debe ser desarrollada ya que *Smack* brinda las bases para soportar el protocolo pero el desarrollador debe gestionar los procesos cuando se envía y reciben paquetes.

Cuando se recibe un paquete *Presence*, el método ***parser*** de la clase *PacketReader* lo examina según el tipo, determina de cuál usuario proviene, obtiene su estado y su mensaje personal (en caso de que esté referenciado en el paquete) y finalmente, retorna un objeto de la clase ***Packet*** con los datos obtenidos, en este caso retornará un objeto de la clase *Presence* que hereda de *Packet* para su posterior uso.

Los paquetes *Presence* son los más simples, por ello gestionarlos no tiene mayor complicación. Para el manejo de los objetos *Presence* se creó la clase ***PresenceManager***, esta clase se comporta como un listeners de los paquetes Presence, es decir, sus métodos son empleados al momento de recibir una instancia de la clase *Presence* que es generada cuando se analiza el paquete en *PacketReader. PresenceManager* se encarga de notificar a la clase *ContactAdministrator* que un contacto ha actualizado su estado, para ello, le es transferido el identificador del contacto y el objeto *Presence* para que *ContactAdministrator* actualice los datos del contacto correspondiente según el contenido obtenido del paquete *presence.*

- **Obtener datos personales de los contactos (*vCard*)**

Luego de obtener los contactos del usuario y haberlos almacenados en el arreglo de contactos de la clase *ContactAdministraror* se procede a solicitar al servidor las respectivas *vCards* de cada uno de los contactos a través del método ***downloadContactsVCard*** de la clase *ContactAdministrator*, este método crea un instancias de la clase ***VCardProvider*** y ***VCard,*** estas clases son especializadas en solicitar y recibir los paquetes *IQ/VCard* al servidor, se recorre cada contacto del arreglo y mediante su JID.

Se crea un paquete *IQ/vCard* del tipo *get* indicándole el JID solicitado, y haciendo uso del *PackageCollector* indicamos que capture inmediatamente el próximo paquete *IQ/vCard* bajo un tiempo de espera (*timeout*). Se hace uso del *timeout* porque puede haber casos en que un usuario no posea vCard, de manera que si

transcurre el tiempo especificado se debe a un error de transferencia o que el servidor no dio respuesta asociada a la vCard del contacto requerido.

Cuando se recibe un paquete *IQ/vCard* por parte del servidor, éste es analizado y se desglosan los atributos, un paquete de este tipo posee toda la información personal del contacto incluyendo el avatar (en caso de que posea un avatar). Finalmente se actualizan los datos del contacto correspondiente.

- **Diseño y desarrollo del entorno gráfico (*Look & Feel*)**

Un elemento esencial de la interfaz gráfica es el campo descriptivo de un usuario, al cual nos referimos como ***ContactField***, este componente se basa en un icono que indica el avatar del usuario, con un par de etiquetas que hacen referencia al nombre del usuario y su mensaje personal, y por último, un icono que señala el estado de disponibilidad del usuario (ver Apéndice C). Un factor importante de este campo es su reutilización ya que es empleado tanto para mostrar los datos del usuario conectado como también cada uno de sus contactos.

ContactScreen se basa en dos secciones, la primera consta con un banner que muestra los datos del usuario que ha iniciado sesión, por los momentos sólo indica el nombre de usuario ya que no ha sido desarrollado los métodos requeridos para obtener los datos personales del usuario que se ha conectado al sistema (este punto se realiza en la cuarta iteración), el diseño de este banner consta de un *ContactField*.

La lista de contactos se basa en dos grupos, **Conectados** y **No Conectados**, estos grupos cuentan con la propiedad de ser desplegados, por lo que se pueden observar los contactos al expandir el grupo como también se pueden ocultar al contraerlo. La carga de contactos se realiza mediante un ciclo que recorre el arreglo de contactos que se encuentra en la clase *ContactAdministrator*, por cada contacto se genera un *ContactField*, y cada *ContactField* es añadido inicialmente en el grupo de No Conectados, el grupo de Conectados empezará a ser llenado a medida que se reciban los paquetes Presence que indican si el usuario está o no disponible. De estar disponible, el *ContactField* es extraído del grupo de No Conectados y es insertado en el grupo de Conectados (ver Apéndice D).

c. Liberación

- **Integración**

  La arquitectura MVC permitió la fácil integración de los módulos desarrollados, vinculando el controlador con el modelo y a su vez con la vista. Las clases de control como lo son *Application*, *ContactAdministrator*, *PacketReader* y *VCardProvider* se vinculan con el modelo de contactos y de paquetes respectivamente. Estas clases de control son empleadas por el controlador de la vista de contactos que se encarga de gestionar los diferentes eventos dentro de la vista.

- **Pruebas de Aceptación**

- **Obtener contactos**

  Se ejecutaron pruebas basadas en iniciar sesión con un usuario, invocando los métodos desarrollados para obtener los contactos, almacenarlos en el arreglo de la clase

ContactAdministrator y finalmente imprimirlos por pantalla (sin implementar la interfaz gráfica). Esta prueba se realizó con distintos usuarios y se lograron los resultados esperados.

- **Obtener *vCards* y Listar Contactos en la Interfaz Gráfica**

Mediante esta prueba se verificó que los contactos sean cargados correctamente en la interfaz de usuario a través de los *ContactFields* y que la solicitud de las *vCards* al servidor sea satisfactoria de manera que se visualicen los datos del contacto.

- **Manejo de la presencia de los contactos**

Esta prueba se encarga de verificar que la aplicación administre de forma correcta los contactos, de manera que al recibir un paquete Presence que provenga por parte de un contacto, éste sea colocado en el grupo correspondiente (Conectados o No Conectados) según la disponibilidad que indique el paquete de presencia, y también, si un contacto actualiza su mensaje personal, se verificará que la aplicación actualice el *ContactField* correspondiente al contacto para que su mensaje personal sea visualizado. Los resultados obtenidos luego de realizar varias pruebas fueron positivos.

IV.3.3 Tercera Iteración: Aplicación *Android/BlackBerry* – Mensajería Instantánea.

Esta iteración tiene por objetivo desarrollar y verificar los procesos de gestión de la mensajería instantánea de la aplicación, para ello se requiere crear métodos que administren los paquetes de tipo *Message*, coordinar el listado de mensajes enviados y recibidos, agrupar el conjunto de conversaciones activas y diseñar los componentes gráficos necesarios.

Los mensajes de la aplicación manejan diversos estados, los cuales pueden ser:

- **Pendiente**: Mensaje en espera para ser enviado
- **Enviando**: Mensaje en proceso de envío.
- **Enviado**: Mensaje ha sido enviado al servidor
- **Entregado**: El destinario ha recibido el mensaje
- **Leído**: El destinatario ha leído el mensaje enviado

Los mensajes recibidos deben ser notificados al usuario mediante efectos sonoros, vibraciones o indicativos de luz, de manera que pueda percibir los mensajes recibidos.

a. Planificación

Las tareas para esta iteración son:

- Envío y Recepción de mensajes de texto por parte de un contacto (*Message*)
- Manejo de notificaciones de Mensaje Enviado, Mensaje Entregado y Mensaje Leído
- Manejo de estados de *Composing* y *Paused*
- Diseño y desarrollo del entorno gráfico (*Look & Feel*)

b. Trabajo

- **Envío y Recepción de mensajes de texto por parte de un contacto (*Message*)**

La clase ***Message*** representa el modelo de un paquete *Message* del protocolo XMPP, esta clase cuenta con atributos de origen y destinatario, tipo de mensaje y cuerpo del mensaje. Las instancias de esta clase serán empleadas para coordinar el manejo de mensajes en la aplicación. Otra clase de gran importancia para

es ***Conversation***, la cual se encarga de vincular las clases *User* y *Message*, además, almacena el identificador del contacto, indica si el chat está activo o no y posee un arreglo con los mensajes enviados y recibidos.

La clase ***ChatAdministrator*** tiene la tarea de coordinar los chats activos mediante instancias de la clase *Conversation*, cuando el usuario envía o recibe un mensaje, el mensaje es listado en la conversación, también actualizará un mensaje cuando este cambie de estado (caso que se explicará más adelante), y por último, si el usuario desea finalizar una conversación, *ChatAdministrator* limpiará la lista de mensajes y cambiará el estado del chat a inactivo.

En los controladores de la aplicación está la clase ***ControlMessage***, la cual es el punto de control por donde son enviados y recibidos los mensajes de la aplicación. Cuando un paquete *Message* es recibido, es analizado, se crea una instancia de la clase *Message* y se cargan sus atributos correspondientes, este objeto lo recibe *ControlMessage* y se verifica que el mensaje posea cuerpo para que sea enviado a *ChatAdministrator*, en caso de que no posea cuerpo es descartado el mensaje recibido.

Para el caso de mensaje enviados, se compone un objeto *Message* con el destinatario y el cuerpo ingresado por el usuario, el objeto es enviado a *ChatAdministrator* para su almacenamiento en la lista de mensajes de la conversación y también a *ControlMessage* para su envió final. Para el proceso de envío de mensajes, *ControlMessage* implementa la clase *PacketWriter*, que cumple la función de generar los paquetes XMPP en base a los datos del objeto y llevarlos a formato XML para así, finalmente enviarlos al servidor.

Como toda aplicación de mensajes, ésta debe hacer que el dispositivo le indique al usuario que ha recibido un mensaje nuevo bien sea a través del uso de Sonido, Vibración, Luz o combinaciones de los tres casos. Para el caso de *Android* se empleó ***Notification*** el cual es una clase del API de *Android* para realizar notificaciones de nuevos eventos, para el caso de *BlackBerry* se empleó ***ApplicationMessage***, que de igual forma se emplea para las notificaciones. En ambos casos se hace uso de **ringtone**, **vibración** intermitente y el **led** del dispositivo móvil para indicar al usuario que ha recibido un nuevo mensaje.

- **Manejo de notificaciones de Mensaje Enviado, Mensaje Entregado y Mensaje Leído**

La clase *Message* fue sometida a diversos cambios, en el cual se le añadieron las funcionalidades necesarias para que los mensajes implementen el manejo de notificaciones de estado. Vale acotar que el proceso de notificaciones es totalmente transparente para el usuario, por lo que él no debe configurar o realizar alguna acción para que la aplicación envíe y reciba las notificaciones.

El manejo de notificaciones de mensaje entregado, se basa en el manejo de dos etiquetas extras dentro de los mensajes, uno para la persona que envía el mensaje original y la otra para el contacto que responde. Basándonos en la extensión **XEP-0184** del protocolo XMPP, el mensaje original del usuario al contacto incluye la siguiente etiqueta ***<request xmlns='urn:xmpp:receipts'/>*** en donde es indicado que se solicita un mensaje de respuesta que afirme la recepción del mensaje original.

El contacto con el cual está conversando al recibir el mensaje determinará que el usuario origen le solicita la notificación de que el mensaje lo recibió exitosamente, a lo que el contacto le debe enviar un paquete *Message* al usuario, este paquete no posee cuerpo y que incluye la etiqueta ***<received xmlns='urn:xmpp:receipts' id='abc-1234'/>*** en donde es indicado que recibió el mensaje y hace referencia al ID del mensaje original.

Con respecto a las notificaciones de mensajes leídos, el protocolo XMPP se debió definir una nueva extensión del protocolo XMPP. Para este caso se creó una nueva implementación en el cual el contacto con que el usuario conversa le envía un paquete *Message* sin cuerpo y con una etiqueta extra que referencia al *ID* del mensaje que ha sido leído, se descartó la posibilidad de que el usuario origen realice solitud de esta notificación, por lo que la misma se hace de manera automática. La etiqueta incorporada al paquete *Message* es ***<read xmlns='urn:xmpp:read' id='abc-1234'/>***.

Mediante el arreglo de mensajes de *ChatAdministrator*, que posee un registro de los mensajes en cada conversación del usuario con sus contactos, cuando la aplicación recibe una notificación de un mensaje, bien sea Entregado o Leído, se detecta a cual *ID* hace referencia la notificación y se recorre el arreglo para actualizar el estado del mensaje cuyo *ID* coincida con el mensaje de notificación, de esta manera, a través de la interfaz gráfica el usuario observará el estado actual de sus mensajes enviados.

Para el envío de notificaciones de mensaje leído, los paquetes *Message* son enviados cuando el usuario ingresa caracteres en el campo de texto del chat, o cierra la conversación, al

ejecutar una de estas dos alternativas, la aplicación verifica los mensajes recibidos de la conversación cuyo estado sea Entregado, su estado es cambiado a Leído y se envían las notificaciones de los mensajes leídos al contacto.

- **Manejo de estados de *Composing* y *Paused***

Las conversaciones en XMPP además de manejar notificaciones de estado de los mensajes, soportan notificaciones de estado del Chat, en el desarrollo del proyecto se implementaron dos, ***Composing***, la cual se emplea notificar al contacto que el usuario está escribiendo un mensaje y ***Paused*** para notificar que el usuario ha dejado de escribir.

Para el manejo de notificaciones de *Composing* se cumplió el estándar **XEP-0085** del protocolo XMPP, en el cual se compone un objeto *Message* sin cuerpo, y se le añade la siguiente etiqueta ***<composing xmlns='http://jabber.org/protocol/chatstates'/>.*** El objeto *Message* es enviado a la clase *PacketWriter* la cual se encarga de generar el paquete con el formato XML. Este proceso se lleva a cabo cada vez que el usuario escribe en el campo de texto empleado para generar el mensaje.

Cuando el usuario hace una pausa mientras escribe el mensaje o simplemente deja de escribir, el estándar dice que se debe enviar una notificación *Paused*, para ello se emplea un hilo encargado de detectar cuando el usuario deja de escribir por un período de tiempo de dos (2) segundos, si en ese transcurso de tiempo el campo de texto no recibe un carácter, el controlador de la vista de la conversación generará un objeto *Message*, sin cuerpo como el mensaje de *composing*, y es añadida la siguiente etiqueta

<paused xmlns='http://jabber.org/protocol/chatstates'/> que de igual forma, este objeto es enviado a *PacketWriter* que genera el paquete en formato XML y lo envía al contacto de la conversación.

- **Diseño y desarrollo del entorno gráfico (*Look & Feel*)**

Se generaron dos vistas, ***ConversationScreen***, que se activa cuando el usuario hace click sobre un contacto de su lista de amistades, la cual invoca la pantalla de conversación. Esta vista posee un banner principal compuesto por un *ContactField* que posee los datos del contacto con el cual conversa, un amplio campo vertical en el cual se listan los mensajes enviados y recibidos y, en la parte inferior de la pantalla, un campo de texto que le permite al usuario generar los mensajes a enviar (ver Apéndice E).

La otra vista desarrollada es ***ConversationPanel***, encargada de listar el grupo de conversaciones abiertas que posee el usuario, indicando el contacto con el cual está conversando y el último mensaje enviado/recibido como vista previa. En el caso de *BlackBerry* esta lista de chats es añadida en la parte superior de la lista de contactos y en el caso de Android, se genero una pantalla en la cual se accede mediante pestañas que permite el desplazamiento a contactos y hacia las conversaciones.

Cada vista posee su respectivo controlador para gestionar las funcionalidades de cada una, ***ControlConversation***, controlador de *ConversationScreen*, cumple la tarea de ser el puente entre *ControlMessage* que gestiona el envío y recepción de mensajes, *ChatAdministrator* y *ContactAdministrator*, encargados gestionar los mensajes y los contactos respectivamente, conectado así a la vista de la conversación. De igual manera, ***ControlConversationPantel***,

controlador de *ConversationtPanel*, coordina el conjunto de funcionalidades para administrar la parte lógica de las conversaciones activas como también la vista.

Los mensajes poseen un icono antes del texto, este icono se encarga de identificar el estado actual del mensaje, el cual puede ser Pendiente, Enviando, Enviado, Entregado o Leído. Cuando un mensaje actualiza su estado, el mensaje es actualizado en la vista para indicar el nuevo estado. Con respecto a las notificaciones de *Composing* y *Paused*, cuando es recibida una notificación de *Composing* y la vista de *ConversationScreen* está activa, el mensaje personal del usuario ubicado en el *ContactField* es sustituido por el mensaje "***Esta Escribiendo...***", cuando es recibida la notificación de *Paused* el mensaje de *composing* es removido, colocando en su lugar el mensaje personal del contacto (ver Apéndice F).

c. Liberación

- **Integración**

  Los controladores generados, *ControlConversation* y *ControlCoversationPanel*, invocan los métodos de sus respectivas vistas, *ConversationScreen* y *ConversationPanel*, en el cual transmiten los datos que van a ser cargados, también se comunican a su vez con las clases de control de mensajes y contactos como lo son *ControlMessage*, *ContactAdministrator* y *ConversationAdministrator*, de esta forma fluye la información que debe ser procesada de un módulo a otro de manera que se respete la arquitectura de la aplicación.

- **Pruebas de Aceptación**

- **Intercambio de mensajes**

La prueba consta en iniciar sesión con un usuario A, una vez que los procesos de autenticación y listar contactos hayan sido ejecutados se seleccionará un contacto de la lista haciendo click sobre el *ContactField* que lo compone, lo que genera un evento donde la aplicación invoca la pantalla de conversación, se escribirán mensajes y éstos deberán ser enviados y mostrados en la lista de mensajes, de igual forma, desde un usuario B conectado en un dispositivo distinto intercambiará mensajes con el usuario A.

A través de esta prueba se verificó que los usuarios puedan conversar, intercambiado todo tipo de mensajes, se realizaron pruebas con mensajes de longitudes distintas. Los resultados obtenidos fueron exitosos.

- **Notificaciones de mensajes**

Con dos clientes conectados en dispositivos distintos se estableció una conversación y se procedió a intercambiar mensajes. Mediante esta prueba se verificó que las funcionalidades de envío y recepción de notificaciones de mensajes Enviados, Entregados, Leídos y Recibidos se ejecuten de forma correcta. Para todos los casos los resultados fueron positivos.

- **Notificaciones de estado de las conversaciones**

De igual forma que las pruebas anteriores, mediante dos clientes conectados en diferentes móviles, identificados como usuario A y usuario B, se probó la funcionalidad de notificar

cuando un contacto está escribiendo un mensaje y cuando deja de escribir. Se procedió a escribir un mensaje con el usuario A, mientras se escribía la aplicación envía la notificación de *Composing* al usuario B, que a través de su dispositivo se observó que indicaba que el usuario con que conversaba está escribiendo un mensaje, por lo que la primera parte de la prueba fue exitosa.

Seguido de esto, se procedió a dejar de escribir con el usuario A, transcurridos dos segundos (*timeout* implementado en la aplicación) se envió correctamente la notificación de *Paused*, tras esto, en el dispositivo del usuario B, el mensaje que indicaba que el usuario A estaba escribiendo fue sustituido por el mensaje personal correctamente.

IV.3.4 Cuarta Iteración: Aplicación *Android/BlackBerry* – Gestión del Perfil de Usuario

La iteración final del proyecto tiene por objetivo la gestión de los datos del usuario de la aplicación, para ello se modificó el modelo que emplea la base de datos, en el cual se le añaden más atributos de carácter personal a la única tabla que posee, Usuarios (ver Apéndice H).

La aplicación posee un objeto global llamado ***USER*** dentro de la clase *Application*, el cual es un objeto de la clase *User* y es empleado para almacenar los datos del usuario que ha iniciado sesión mientras que la sesión este abierta, de igual forma que sus contactos, posee Nombre Completo, Mensaje Personalizado, Avatar, Número Telefónico, Correo Electrónico, Fecha de Nacimiento y Ubicación.

Vale acotar que el código empleado en la iteración 2 (Listado de Contactos) fue reutilizado para obtener la *vCard* del usuario, de esta forma se cumple con el principio de no repetir lo que está hecho, se obtienen los

datos personales del usuario, los cuales serán almacenados en la base de datos (siempre y cuando el usuario haya habilitado los permisos de almacenamiento en la aplicación).

a. Planificación

Las tareas para esta iteración son:

- Modificar el Modelo de la Base de Datos
- Obtener información personal del usuario (*vCard*)
- Editar los atributos principales del usuario
- Diseño y desarrollo del entorno gráfico (*Look & Feel*)

b. Trabajo

- **Modificar el Modelo de la Base de Datos**

El modelo de la base de datos de la aplicación hecho en *SQLite* fue modificado, en el cual se añadieron los nuevos campos. Esta modificación en el modelo conllevó a realizar cambios en la clase *DataBase*, mediante la modificación de sus métodos de persistencia y el manejo de las nuevas columnas. Sólo serán almacenados los datos personales de los usuarios que posean la opción de recordar usuario y contraseña o recordar usuario en la aplicación.

- **Obtener información personal del usuario (*vCard*)**

Reutilizando el código desarrollado en la segunda iteración del proyecto, en donde la fase de trabajo obtiene las respectivas *vCards* de los contactos, se emplearon los métodos establecidos para que de igual forma se obtuviese la *vCard* del usuario que ha iniciado sesión.

Cuando el usuario inicia sesión, se ejecuta en un hilo aparte para la carga de información de la *vCard* del usuario invocando el método ***downloadContactVCard*** de la clase *ContactAdministrator*, una vez es recibido el paquete de respuesta del servidor, en caso positivo, los datos que posee son cargados en el objeto *USER* y serán almacenados en la base de datos si el usuario otorgó los permisos a la aplicación.

- **Editar los atributos principales del usuario**

Todos los datos personales del usuario pueden ser personalizados como él lo desee, para ello se desarrolló la clase ***ControlIQ***, está clase se encarga de gestionar las solicitudes de actualización de uno o más datos personales del usuario.

Para el caso de actualizar los datos, la clase *ControlIQ* posee métodos para la actualización de cada atributo, para el caso de modificar el avatar, se debe codificar a ***Base64*** y generar un *String*. Finalmente en cada método de *ControlIQ* el valor modificado es sustituido en la respectiva etiqueta del paquete *IQ/vCard* y si el usuario no posee *vCard*, esta es creada mediante una instancia de la clase *VCard*. Los datos se cargan el paquete y se le asigna el tipo *set* para que el servidor actualice su base de datos.

- **Diseño y desarrollo del entorno gráfico (*Look & Feel*)**

ProfileScreen fue la vista desarrollada para está iteración, está cuenta con distintos ítems que permiten al usuario ver y editar sus datos personales (ver Apéndice G). ***ControlProfile***, el controlador de esta vista, es el encargado de recibir y verificar los

datos obtenidos a través de la interfaz y traspasarlos a *ControlIQ* para el respectivo procesamiento y envío al servidor.

Se emplean las clases ***Intent*** de *Android* y ***FilePicker*** de *BlackBerry* para seleccionar la imagen ubicada en la memoria del dispositivo del usuario, lo que le permite cambiar su avatar.

c. Liberación

- **Integración**

  La integración de las clases desarrolladas en los módulos de vista y control fue muy simple ya que se reutilizó el código desarrollado anteriormente en la segunda iteración, esto ayudó a la rápida integración de los componentes elaborados.

- **Pruebas de Aceptación**

- **Obtener vCard del usuario**

  La prueba se basa en realizar la solicitud al servidor de la vCard del usuario que ha iniciado sesión y que sus datos sean correctamente almacenados en la base de datos (si el usuario otorgo los permisos a la aplicación). Se verificó que las clases y métodos desarrollados cumplan correctamente sus funcionalidades obteniendo así resultados positivos.

- **Modificar vCard del usuario**

  Mediante esta prueba se verificó que los procesos de actualización de los datos del usuario se realizarán correctamente. Probando así editar cada elemento de su perfil.

Tras haber realizado diversas pruebas, los resultados obtenidos fueron exitosos.

IV.4 Etapa de Estabilización

IV.4.1 Integración del Sistema

A medida que se fueron realizando las iteraciones de la etapa de desarrollo la metodología *Mobile-D* exige que las funcionalidades desarrolladas deben ser integradas en la tarea de *liberación* de la iteración, por lo que para esta etapa se facilito el proceso de integración del sistema en general, verificando el correcto acoplamiento de los módulos desarrollados y la correcta distribución del código fuente respetando así la arquitectura Modelo Vista Controlador.

IV.4.2 Documentación

Se procedió a realizar un análisis del código fuente de las aplicaciones, para verificar, corregir y añadir (en caso de no haber) comentarios descriptivos a las principales clases, métodos y atributos del proyecto.

También se desarrolló el Manual de Usuario para ambas versiones de la aplicación, como entregable que brinda soporte en actividades de uso de las aplicaciones.

IV.5 Etapa de Pruebas de Sistema

Se realizaron pruebas en toda la aplicación en sí, tanto en la versión *Android* como en *BlackBerry*, corroborando así las pruebas anteriormente realizadas en la etapa de desarrollo.

CAPITULO V. RESULTADOS

Luego del trabajo realizado durante el ciclo de vida del proyecto se obtuvo como resultado final una aplicación de mensajería instantánea de gran calidad y alcance, disponible en teléfonos inteligentes con sistema operativo *Android* y *BlackBerry*, con funciones de autenticación de usuario, listado de usuarios, mensajería instantánea, implementación de notificaciones de estado de mensajes y personalización del perfil de usuario.

Para lograr el producto final, el proceso de elaboración de las aplicaciones de mensajería instantánea para *Android* y *BlackBerry* se desarrolló basándose en iteraciones, las iteraciones se encuentran divididas en tres tareas (*Planificación*, *Trabajo* y *Liberación*) y por cada iteración se elaboró un módulo de la aplicación. Esto permitió un desarrollo organizado y robusto de las aplicaciones. Finalmente, para verificar el correcto cumplimiento de los objetivos planteados se analizará cada uno tomando en cuenta el resultado esperado con respecto al resultado obtenido.

- **Estudiar el protocolo para mensajería instantánea XMPP**

Este objetivo tiene por finalidad la introducción y dominio a las principales funcionalidades y procesos de comunicación del protocolo XMPP, logrando así obtener una visión clara y concisa de los procedimientos de gestión que fueron aplicados para llevar a cabo las funcionalidades de inicio de sesión, listado de contactos, intercambio de mensajes, edición del perfil de usuario y el cierre de la sesión dentro de los clientes XMPP desarrollados (*Android* y *BlackBerry*).

Para cumplir este objetivo se debió investigar gran cantidad de la documentación existente del protocolo XMPP, tanto en sitios oficiales como no oficiales en Internet, vale destacar que la documentación del protocolo es muy amplia, todo esto con el propósito de tener una introducción al aspecto teórico de XMPP, como también a los procesos prácticos y la sintaxis

aplicada en formato XML empleada para la comunicación e intercambio de datos entre los dispositivos clientes y el servidor.

- **Investigar sobre el servidor *Openfire* para la gestión del sistema de mensajería**

Inicialmente, la empresa DIVUX SYSTEMS desarrolló una aplicación Web la cual incorpora la funcionalidad de chat empleando el servidor *Openfire*, luego de este proyecto surgió la idea de desarrollar un conjunto de aplicaciones móviles que se conecten al servidor con la finalidad de crear una amplia red de comunicación, en esa idea está centrado este Trabajo Instrumental de Grado.

Antes de desarrollar las aplicaciones para *Android* y *BlackBerry* se debe tener una inducción teórica tanto del protocolo de comunicación a implementar como también del ambiente en el cual la plataforma estará operativa, por ello, la finalidad de este objetivo es obtener una noción de las funcionalidades que posee el servidor *Openfire* y cuáles son sus características.

Mediante una investigación concreta de las funcionalidades que posee el servidor se adquirieron los conocimientos necesarios para comprender su operatividad y la manera en que gestiona la base de datos que implementa.

- **Proponer formalmente los cambios al protocolo XMPP para poder garantizar seguridad y eficiencia en la transmisión de datos**

Se deberá realizar una propuesta formal sobre los mecanismos de seguridad y transmisión de los paquetes de manera que favorezcan el intercambio de data integra.

Luego de haber investigado las propiedades del protocolo XMPP y las características que posee el servidor *Openfire*, se realizaron tres propuestas en función de la seguridad y eficiencia de la transmisión de datos:

- **Autenticación SASL:** El protocolo XMPP permite la autenticación mediante *SASL (Simple Authentication and Security Layer / Capa de Seguridad y Autenticación Simple)*, si bien el protocolo XMPP cuenta con su propio estándar de autenticación a través de los paquetes *IQ/Auth* también incluye *SASL* como mecanismo de seguridad e integridad de la data en el proceso de autenticación del usuario, lo que conlleva a realizar un proceso de autenticación más robusto y seguro.

- **Implementar SSL/TLS:** La implementación de *SSL (Secure Sockets Layer / Protocolo de Capa de Conexión Segura) TLS (Transport Layer Security / Seguridad de la Capa de Transporte)* permite el envió de paquetes encriptados de manera que se garantice la integridad de la información que viaja en la red.

  Hacer uso de una conexión SSL/TLS conlleva a que las aplicaciones y el servidor realicen un mayor trabajo en el proceso intercambio de paquetes, ya que se deben realizar procedimientos de *handshake* para la autenticación de la conexión a través de los certificados y la negociación de una clave secreta. Además, el manejo de paquetes encriptados tiene como consecuencia el incremento del tiempo para llevar a cabo el intercambio de data entre el servidor y el cliente debido a que se deben encriptar y desencriptar cada paquete que se envía o recibe.

  Ante lo mencionado anteriormente, se recomienda que la implementación de SSL/TLS sea de carácter opcional, en donde el usuario decida si desea establecer una conexión SSL/TLS o continuar conexión vulnerable con el servidor.

- **Compresión:** Utilizar *Zlib* en los streams de XMPP. *Zlib* es una librería *Open Source* que permite la compresión de datos. La implementación de esta librería permitirá enviar paquetes reduciendo así su tamaño y ahorrando el uso del ancho de banda de la red.

  Si bien la compresión resulta ahorrar el tamaño de los paquetes, también aumentaría el tiempo de respuesta entre el cliente y el servidor para llevar a cabo la compresión y la descompresión del paquete. Este procedimiento es en cierta parte es innecesario para casos de intercambio de solo texto. Por lo que una recomendación es aplicar esta funcionalidad en el intercambio de archivos digitales bien sea de imagen, audio o video, en el cual los tamaños son muy superiores en comparación al texto y el uso de compresión de paquetes puede ser provechoso para garantizar una comunicación eficiente y eficaz.

- **Aprender sobre el desarrollo de aplicaciones en el sistema operativo *Android* y *BlackBerry***

La finalidad de este objetivo es adquirir conocimientos y practica acerca del desarrollo en sistemas operativos *Android* y *BlackBerry*, ya que las aplicaciones requieren un manejo del alto potencial de sus funcionalidades como lo son el manejo de conexiones, el intercambio de datos bajo una arquitectura cliente-servidor, la incorporación de gran variedad de componentes gráficos, la gestión de eventos con respecto al dispositivo, entre otros.

Para cumplir este objetivo se investigó información referente a guías para desarrolladores en Internet, a través de sitios oficiales y no oficiales de *Android* y *BlackBerry*, vale acotar que también fue de gran apoyo haber inscrito las electivas de *Desarrollo de Aplicaciones BlackBerry* y *Desarrollo de Aplicaciones Android* para recibir una docencia y práctica apropiada.

- **Construir el cliente de Mensajería instantánea para *Android***

Con el propósito de presentar un producto de calidad, se tiene por finalidad el desarrollo de una aplicación que cumpla con el alcance propuesto inicialmente (ver Alcance y Limitaciones).

Este objetivo se llevó a cabo en todo el transcurso de la etapa de desarrollo del ciclo de vida del proyecto, centrado en la elaboración del cliente XMPP para dispositivos móviles con sistema operativo *Android*.

Para el desarrollo se utilizó la librería *Smack*, la cual es una librería *Open Source* muy extensa pero fácilmente configurable, esta librería fue sometida a diversos cambios para que pudiese operar en sistemas operativos *Android*. La aplicación se desarrolló en cuatro iteraciones encargadas de elaborar los procesos de Inicio y Cierre de Sesión, Listado de Contactos, Envió y Recepción de mensajes y Editar el perfil de usuario (para mayor información ver Desarrollo).

Como resultado final, se obtuvo una aplicación para dispositivos móviles con sistema operativo *Android* 2.1, 2.2 y 2.3 que cumple con los requerimientos iníciales, está aplicación requiere un espacio de 800 Kb en la memoria del dispositivo móvil.

- **Construir el cliente de Mensajería instantánea para *BlackBerry***

En paralelo al objetivo anterior, se desarrollo de igual forma el cliente XMPP para dispositivos móviles *BlackBerry*. Esta aplicación difiere en términos internos del código fuente con respecto a la aplicación realizada en *Android*, pero es transparente para el usuario ya que de igual forma puede realizar los procesos de Inicio y Cierre de Sesión, Listado de Contactos, Envió y Recepción de mensajes y Editar el perfil de usuario.

Como resultado final, se obtuvo una aplicación para dispositivos móviles con sistema operativo *BlackBerry* para la versión 5.0 que cumple

con los requerimientos iníciales, está aplicación requiere un espacio de 400 Kb en la memoria del dispositivo móvil.

- **Verificar formalmente el código de los clientes de IM para *Android* y *BlackBerry***

Este objetivo tiene el propósito de hacer entrega de un código fuente bien elaborado y correctamente documentado para ambas aplicaciones.

Para ello se presentó una estructura de código organizada bajo una arquitectura Modelo Vista Controlador, lo que permitió una correcta distribución de las clases y archivos, separando la capa lógica de la vista y colocando al controlador como el administrador respecto al modelo y las librerías empleadas. Los métodos y atributos de las principales clases de ambas aplicaciones fueron correctamente documentados, como también fueron sometidos a un proceso de optimización y limpieza del código fuente.

- **Verificar formalmente los cambios realizados al protocolo XMPP**

Deberán ser documentados los casos en que se hayan propuesto e implementado nuevas funcionalidades o modificaciones al protocolo XMPP como material de apoyo.

XMPP es un protocolo muy completo pero durante el desarrollo del proyecto se evidenció, a través de las investigaciones realizadas, que el protocolo no posee una publicación oficial de alguna extensión que defina el manejo de las funcionalidades de Notificaciones para Mensajes Leídos, por lo que se debió definir una nueva extensión para XMPP. Para este objetivo se realizó un documento en donde están especificadas las funcionalidades, requerimientos, formatos y consideraciones de la extensión de Notificaciones de Mensajes Leídos para el protocolo XMPP (ver Apéndices L y M).

- **Documentar los artefactos exigidos por DIVUX SYSTEMS en todo el proceso de desarrollo de la solución**

Con el propósito de generar toda la documentación y material de apoyo solicitado por la empresa DIVUX SYSTEMS, para este objetivo se elaboraron dos documentos:

- **Especificación de Requerimientos de Software:** En este documento se determinan los requerimientos funcionales para la aplicación de mensajería instantánea *Android/BlackBerry*, sus principales características y descripción de los Casos de Uso (ver Apéndices N, O, P, Q, R y S), especificando las actividades para llevar a cabo los principales procesos dentro de la aplicación.

- **Manual de Usuario:** Se elaboró un documento que sirve de guía para la asistencia de los usuarios en el uso de las funcionalidades de la aplicación y respuestas a preguntas frecuentes (ver Apéndices I, J y K).

CONCLUSIONES Y RECOMENDACIONES

Luego de haber finalizado el desarrollo de las aplicaciones de MI para dispositivos móviles *Android* y *BlackBerry* y en base a los resultados obtenidos en función de los objetivos planteados en este Trabajo Instrumental de Grado, podemos concluir que:

- Durante el ciclo de vida del proyecto se adquirieron gran cantidad de conocimientos que abarcan puntos en el área de desarrollo de software, la implementación de nuevas tecnologías hacia la era de los teléfonos inteligentes y la gestión de conexiones inalámbricas y la transferencia de datos, por lo que todo esto le da un valor agregado al trabajo realizado.

- La utilización de *Mobile-D* como la metodología aplicada permitió un desarrollo organizado y robusto de las aplicaciones *Android* y *BlackBerry*, donde cada etapa dentro de la metodología complementa el trabajo ordenado, como a su vez, el trabajo iterativo en la etapa de desarrollo de la metodología permitió el avance incremental del proyecto y la pronta resolución de inconvenientes.

- El desarrollo de las aplicaciones bajo una arquitectura Modelo Vista Controlador aportó un valor importante en la estructura y organización del código fuente lo que favorece a la ubicación y optimización de sus archivos.

- La implementación de un protocolo de comunicación potente como lo es XMPP brindó gran soporte a la reglamentación y gestión de la plataforma de mensajería instantánea y las aplicaciones móviles desarrolladas.

  Además, es factible la escalabilidad de XMPP, gracias a esto fue posible el desarrollo de una nueva extensión del protocolo que permite el manejo de notificaciones de Mensajes Leídos.

- Ejecutar pruebas sobre las funcionalidades desarrolladas en las iteraciones de la etapa de desarrollo permitió validar el buen cumplimiento de las tareas de la iteración y así avanzar a la siguiente con el respaldo de que el trabajo desarrollado funciona correctamente.

- La incorporación de funcionalidades para el manejo de notificaciones de estados de los mensajes como Pendiente o en Espera, Enviando, Enviado, Entregado y Leído añade gran valor a las aplicaciones de MI desarrolladas, permitiéndole al usuario conocer el estado actual de los mensajes que ha enviado.

Una vez concluido el trabajo realizado para el desarrollo de las aplicaciones y haber realizado las pruebas que garanticen su operatividad, se pueden realizar las siguientes recomendaciones:

- El protocolo XMPP cuenta con muchas funcionalidades que no fueron incluidas en las aplicaciones, por lo que se podrían desarrollar nuevos módulos que implementen funciones como lo son las conversaciones grupales y la transferencia de archivos digitales (imagen, audio y video) lo que colocaría a las aplicaciones en un alto nivel de competitividad.

- En el ámbito de la seguridad en la red, implementar una conexión basada en *SSL/TLS* es un aspecto importante que debe ser tomado en cuenta. Añadir está características a las aplicaciones clientes y al servidor de la plataforma de Mensajería Instantánea garantiza la integridad de la información en la red. Esta funcionalidad es de carácter opcional de manera que el usuario pueda decidir si emplear o no una conexión segura, teniendo en cuenta las consideraciones con respecto al uso de una conexión segura y los recursos que está puede consumir por parte de las aplicaciones y el servidor.

- Implementar la compresión de los *streams* de XMPP como medida de hacer un uso más eficiente de las conexiones durante el envío y recepción de archivos digitales bien sea imagen, audio o video. El uso de compresión para las transmisiones de solo texto queda descartado.

- Desarrollar una funcionalidad que permita el envío de mensajes de tipo *broadcast* sería un aspecto resaltante de la aplicación, en el cual le sea permitido al usuario la difusión de un mensaje a múltiples contactos, este mensaje podrá ser enviado a todos sus contactos o a los que el seleccione. Vale acotar que el protocolo XMPP soporta este tipo de mensajes.

REFERENCIAS BIBLIOGRÁFICAS

CodeFutures. (2011). Data Access Object. Recuperado el 2011, de CodeFutures: http://www.codefutures.com/data-access-object/

Comusoft. (13 de Noviembre de 2010). Modelo Vista Controlador. Obtenido de Comusoft: http://www.comusoft.com/modelo-vista-controlador-definicion-y-caracteristicas

Concepto Definición. (2011). Definición de *Android*. Obtenido de Concepto Definición: http://conceptodefinicion.de/android/

Definición. (2011). Definición de Java. Obtenido de Definición: http://definicion.de/java/

Definición. (2011). Definición de XML. Obtenido de Definición: http://definicion.de/xml/

Definiciones. (2011). Definición de Telefonía Celular. Obtenido de Definiciones: http://www.definiciones.com.mx/definicion/T/telefonia-celular/

Definiciones de. (Abril de 2011). Definición de BlackBerry. Obtenido de Definiciones de: http://definicionesde.com/e/blackberry/

Developers. (s.f.). Modelo Vista Controlador. Obtenido de Developers: http://developers.do/post/6326262572/primeros-pasos-en-asp-net-mvc

GSMSpain. (2011). Definición de *Smartphone*. Obtenido de GSMSpain: http://www.gsmspain.com/glosario/?palabra=SMARTPHONE

Ignite Realtime. (2010). Servidor *Openfire*. Obtenido de Ignite Realtime: http://www.igniterealtime.org/projects/openfire/index.jsp

Ignite Realtime. (4 de Julio de 2011). Librería XMPP - Smack. Obtenido de Ignite Realtime: http://www.igniterealtime.org/projects/smack/

JabberEs. (23 de Septiembre de 2003). Mensajería Instantánea. Obtenido de JabberEs: http://www.jabberes.org/node/72

JabberEs. (25 de Septiembre de 2003). Protocolo XMPP. Obtenido de JabberEs: http://www.jabberes.org/introduccion

Kioskea. (16 de Octubre de 2008). Red Celular. Obtenido de Kioskea: http://es.kioskea.net/contents/telephonie-mobile/gsm.php3

Lage Serrano, O. (15 de Junio de 2005). *Jabber/XMPP*. Obtenido de Jabber/XMPP: http://www.jabberes.org/files/download/Jabber-XMPP.pdf

Maier, R. (2010). *Professional Android 2 Application Development*. Newcastle: Wrox.

Martinez, E. (30 de Septiembre de 2004). *Eveliux*. Telefonía Celular. Obtenido de Eveliux: http://www.eveliux.com/mx/el-abc-de-la-telefonia-celular-parte-1.php

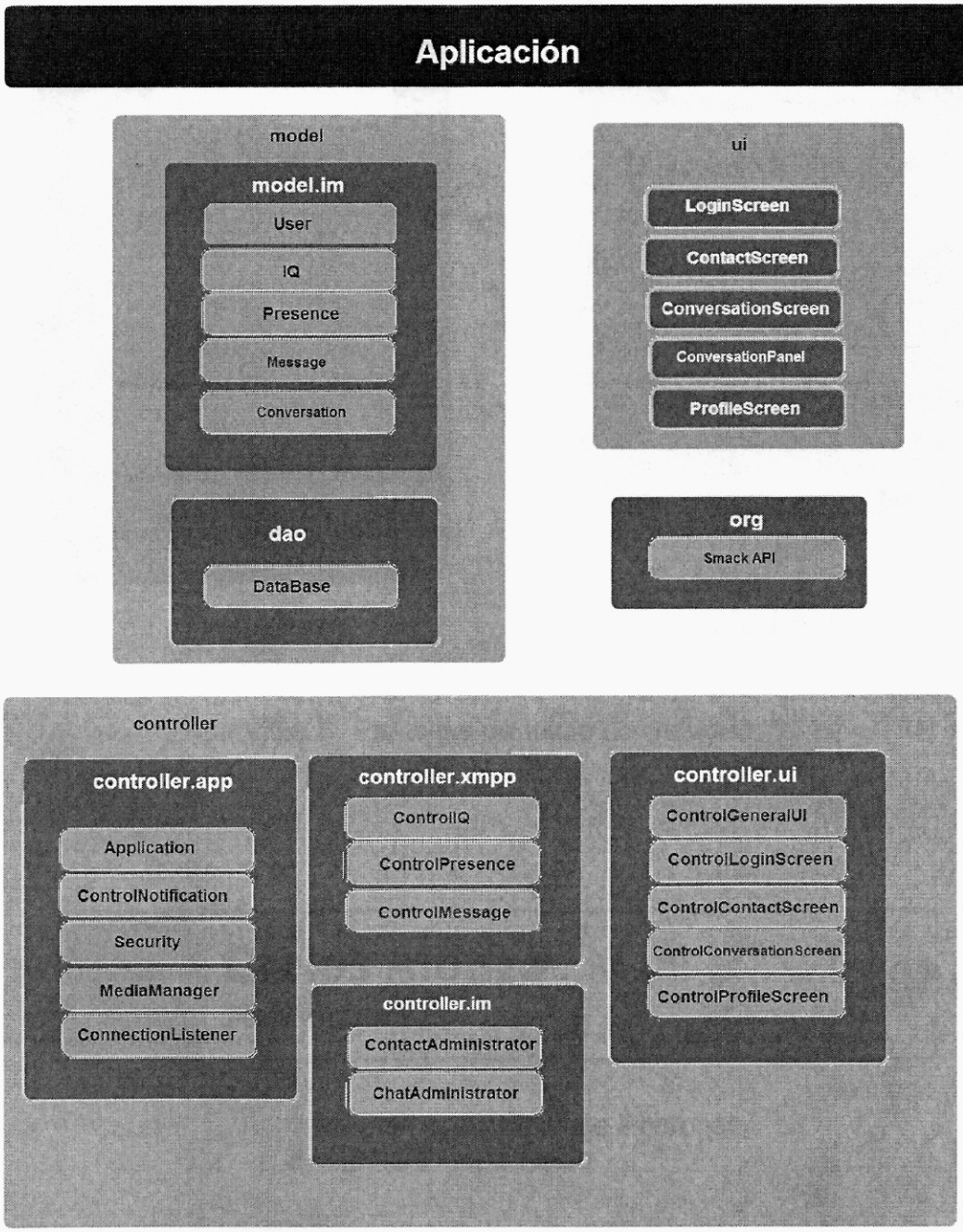
Master Magazine. (Febrero de 2005). Definición de XML. Obtenido de Master Magazine: http://www.mastermagazine.info/termino/7292.php

Nokia. (2008). Mensajería Instantánea. Obtenido de Nokia: http://nds1.nokia.com/phones/files/guides/Nokia_N79-1_Instant_Messaging_AAL_es.pdf

Rizk, A. (2009). *Beginning BlackBerry Development*. New York: Apress.

SQLite. (2011). About SQLite. Obtenido de SQLite: http://www.sqlite.org/about.html

Wikipedia. (2011). Telefonía Móvil. Obtenido de Wikipedia: http://es.wikipedia.org/wiki/Telefon%C3%ADa_m%C3%B3vil

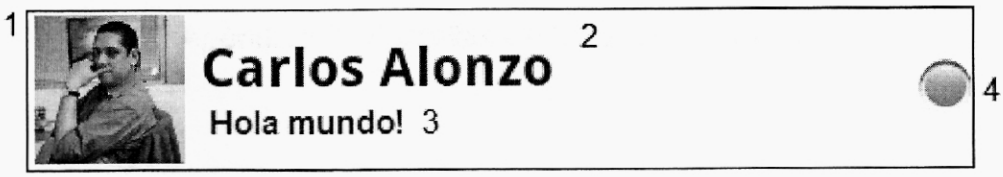


Apéndice A. Estructura del Código Fuente.

android

BlackBerry

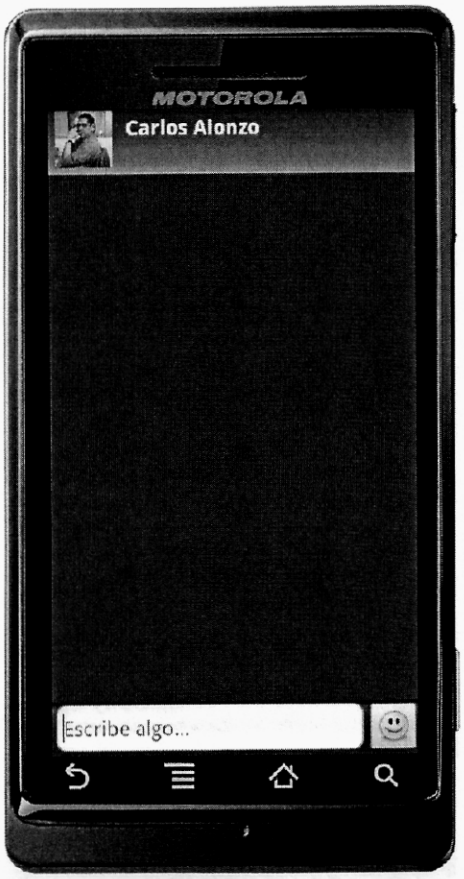
Apéndice B. Pantallas de Inicio de Sesión

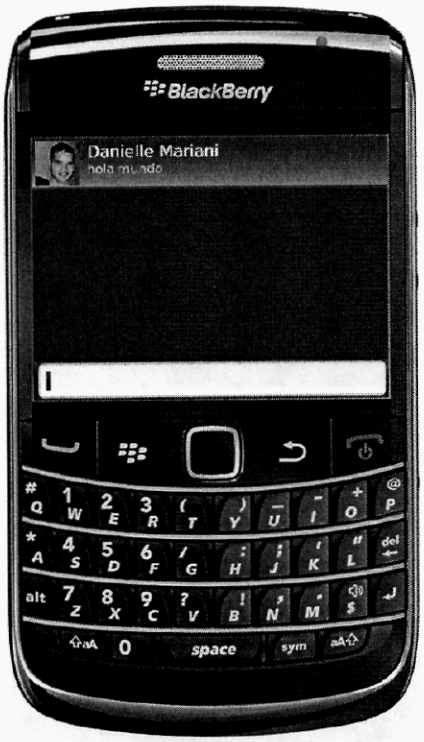


1.- Avatar 3.- Mensaje Personal
2.- Nombre 4.- Estado

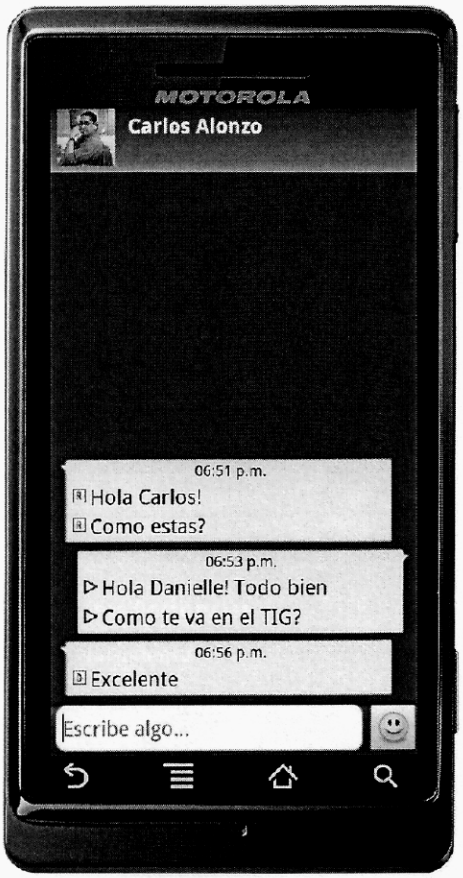
Apéndice C. Campo de contacto (ContactField)

Apéndice D. Pantallas de Lista de Contactos





Apéndice E. Pantallas de Conversación



android

BlackBerry

Apéndice F. Notificaciones de Mensajes

android

BlackBerry

Apéndice G. Pantallas del Perfil de Usuario

Apéndice H. Modelo de la Base de Datos Nativa de las Aplicaciones

USER
• id
• username
• fullname
o password
o status
o birthday
o email
o pone
o location

Manual de Usuario

El presente documento tiene la finalidad de asistir al usuario que utiliza la aplicación de Mensajería Instantánea tanto en su versión para teléfonos inteligentes Android y BlackBerry.

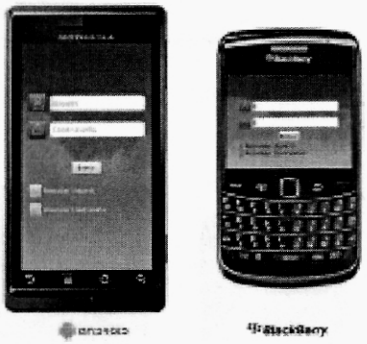
Requisitos para usar la aplicación

Necesita de un dispositivo móvil con sistema operativo Android o BlackBerry el cual disponga de una conexión a Internet bien sea a través de la línea telefónica o a través de una red Wi-Fi.

La Aplicación

1. **Inicio de Sesión**

Para iniciar sesión basta con ejecutar la aplicación en su dispositivo, para acceder a ella se debe hacer click en el icono de la aplicación. Esta acción levantará la ventana de Inicio de Sesión.



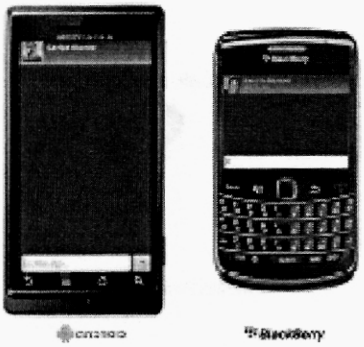
La pantalla de inicio de sesión cuenta con 2 campos de textos en el cual el usuario ingresa su nombre usuario y su contraseña, también cuenta con las opciones de recordar usuario y contraseña en la parte inferior de la pantalla. Una vez que los datos han sido ingresados se debe hacer click en el botón "Entrar".

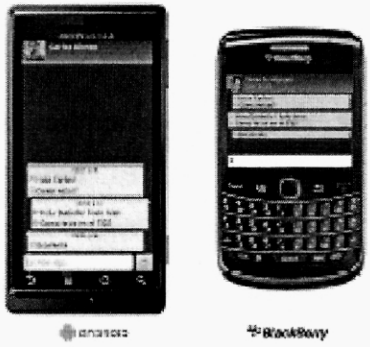
2. **Conversar con un contacto**

Luego de haber iniciado sesión, aparecerá la pantalla de Contactos, en ella se listan los contactos que posea el usuario, estos contactos se encuentran divididos en dos grupos, los contactos que están "conectados" y los que están "no conectados".

Para iniciar una conversación con un usuario se debe hacer click sobre el campo del contacto con que se desea hablar. Esta acción abrirá la pantalla de Conversación, dentro de esta pantalla se cuenta con un campo de texto en el cual se puede elaborar un mensaje de

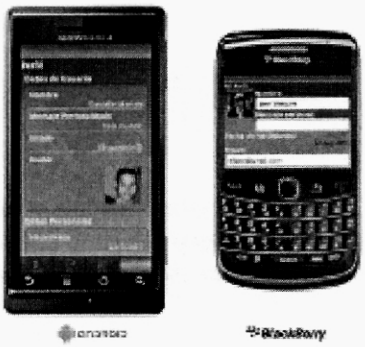
texto y para enviarlo se debe presionar la tecla de envió del dispositivo, o también se puede acceder al menú de la pantalla mediante la tecla de menú del dispositivo y entre las opciones desplegadas en la pantalla se encuentra la de enviar mensaje.





3. **Editar el Perfil de usuario**

Estando conectado es posible modificar la información personal de la cuenta, para ello debe ir a la pantalla de Perfil, a la cual se puede acceder haciendo click en la barra superior que posee nuestra información detallada en la pantalla de Contactos.



En la pantalla de Perfil es posible editar datos como: Nombre de usuario, Mensaje Personal, Estado, Avatar, Fecha de Nacimiento, Correo Electrónico, Teléfono y Ubicación. Cada atributo cuenta con su respectivo campo en donde se ingresa el valor deseado. Finalmente se confirman los cambios y de esta manera se actualiza la información personal del usuario.

4. **Cerrar Sesión**

Estando conectado es posible cerrar la sesión si ya no desea utilizar la aplicación en un momento dado, para ello se debe presionar la tecla menú en la pantalla de Contactos y seleccionar la opción de cerrar sesión.

Iconografía

1.- Icono de la Aplicación

2.- Icono de Mensaje Recibido

3.- Iconografía de los Estados de los Mensajes

Preguntas Frecuentes

1. **Mi aplicación no puede iniciar sesión**

   Asegúrese de que este ingresando los datos correctos y que el dispositivo móvil posea conexión a Internet.

2. **Mi aplicación no envia ni recibe mensajes**

   Verifique que el dispositivo móvil posea conexión a Internet.

3. **¿Cómo guardar mis datos de inicio de sesión?**

   Antes de iniciar sesión en la pantalla de Inicio de Sesión, seleccione las opciones de Recordar Usuario y Recordar Contraseña si desea que sean guardados ambos o sólo seleccione Recordar Usuario si no desea que su contraseña quede almacenada.

4. **¿Cómo eliminar mis datos de inicio de sesión?**

   Antes de iniciar sesión en la pantalla de Inicio de Sesión, deseleccione las opciones de Recordar Usuario y Recordar Contraseña si desea que sean eliminadas ambas o solo deseleccione Recordar Contraseña si solo desea que su contraseña no quede almacenada.

5. **¿Cómo finalizar una conversación?**

   Dentro de la pantalla de conversación presione la tecla menú del dispositivo y seleccione la opción de Finalizar Chat. De esta manera se elimina la conversación y la lista de mensaje con el respectivo contacto.

Apéndice K. Manual de Usuario - Parte III

XEP – Notificación de Estado de Mensajes Leídos

Resumen: Esta especificación define una extensión del protocolo XMPP para el manejo de mensajes leídos. Por lo que la persona que envía un mensaje puede recibir una notificación que indique que el mensaje ha sido leído por el receptor.
Autor: Danielle Mariani
Derechos de Autor: DIVUX SYSTEMS S.R.L.
Estatus: Borrador
Tipo: Normativa
Versión: 1.0
Última Actualización: 04-07-2011

- **Introducción**
  Ante la posibilidad de manejar notificaciones de mensajes entregados, se elaboró está extensión del protocolo XMPP destinada al manejo de estados de mensajes leídos.

  Este documento define el protocolo que habilita el manejo de notificaciones de mensajes leídos, a través de un mensaje de reconocimiento que es enviando por el cliente receptor luego de determinar que el mensaje ha sido leído, este mensaje es enviado al cliente emisor y se referencia el ID del mensaje original.

- **Terminología**
  - "Mensaje Fuente" hace referencia al mensaje original del cliente que emisor.
  - "Mensaje de Notificación" hace referencia al mensaje enviado por el receptor que notifica el estado de leído del mensaje fuente.

- **Requerimientos**
  Este documento aborda los siguientes requisitos:
  - El cliente debe estar habilitado para enviar la notificación de que un mensaje ha sido leído.
  - El cliente debe estar habilitado para recibir una notificación de que un mensaje enviado previamente ha sido leído.

- **Cuando emplear el Protocolo**
  El cliente XMPP receptor recibe un mensaje fuente por parte de otro cliente, en este caso el emisor. El receptor debe determinar mediante validaciones que el usuario que utiliza la aplicación ha leído el mensaje, posteriormente se procede a verificar toda la lista de mensajes recibidos y a

enviar un mensaje de notificación por cada mensaje de la lista cuyo estado sea diferente a leído.

- **Formato del Protocolo**
  - Cliente emisor envía el mensaje original

```
<message
  from='pedro@servicio/movil'
  id='12345'
  to='jose@servicio/pc'>
 <body>Hola Mundo!</body>
</message>
```

  - Cliente receptor determina que el usuario leyó el mensaje y envía la notificación de mensaje leído

```
<message
  from= 'jose@servicio/pc'
  id='67890'
  to='pedro@servicio/movil'>
 <read xmlns='urn:xmpp:read' id='12345'/>
</message>
```

- **Consideraciones del Protocolo**
  - **Tag**
    El *tag* empleado para los paquetes XMPP Message de notificación de mensaje leído es "read".
  - **Namespace**
    El *namespace* empleado en los paquetes XMPP Message de notificación de mensaje leído es "urn:xmpp:read".

- **Agradecimientos**
  Gracias a Daniel Cestari por sus aportaciones en la definición de esta extensión del protocolo XMPP.

ID: IM_CU_01	Caso de uso: Iniciar Sesión
Descripción: Permite acceder al sistema	
Requerimiento: IM_RF_01 (Iniciar Sesión)	
Precondición: El actor debe ejecutar la aplicación de mensajería instátanea.	
FLUJO BASICO:	
ACTOR	**SISTEMA**
1 Introduce el Password	
2 Click en el botón OK	3 Verifica el dato introducido
	4 Consulta el password en la base de datos
	5 Accede al Sistema
FLUJOS ALTERNOS	
ACTOR	**SISTEMA**
	3.1 Campo vacío, mostrar mensaje de advertencia.
3.2 Volver al paso 1	
	4.1 Usuario o Contraseña invalida, mostrar mensaje de error.
4.2 Volver al paso 1	
Poscondición: El Actor ha iniciado sesión.	
Puntos de extensión: IM_CU_02 (Cerrar Sesión) IM_CU_05 (Enviar Mensaje) IM_CU_06 (Editar Perfil)	

Apéndice N. Caso de Uso - Iniciar Sesión

ID: IM_CU_02	Caso de uso: Cerrar Sesión
Descripción: Permite salir del sistema	
Requerimiento: --	
Precondición: El actor debe haber iniciado sesión	
FLUJO BASICO:	
ACTOR	**SISTEMA**
1 Click en Tecla Menú	
2 Click en Cerrar Sesión	
	3 Cerrar el sistema
	4 Volver al Inicio. IM_CU_01
Poscondición: El Actor ha cerrado sesión.	
Puntos de extensión: N/A	

Apéndice O. Caso de Uso - Cerrar Sesión

ID: IM_CU_03	Caso de Uso: Recordar Nombre de Usuario y Contraseña
Descripción: Permite recordar el nombre de usuario y contraseña	
Requerimiento: --	
Precondición: El actor debe ejecutar la aplicación	
FLUJO BASICO:	
ACTOR	SISTEMA
1 Click en las opciones de recordar usuario y recordar contraseña.	
2 Iniciar Sesión (IM_CU_01)	
	3 Crear/Actualizar registro en la base de datos nativa de la aplicación.
Poscondición: El actor ha otorgado permisos a la aplicación para que esta almacene sus datos.	
Puntos de Extensión: N/A	

Apéndice P. Caso de Uso - Recordar Nombre de Usuario y Contraseña

ID: IM_CU_04	Caso de Uso: Recordar Nombre de Usuario
Descripción: Permite recordar el nombre de usuario.	
Requerimiento: --	
Precondición: El actor debe ejecutar la aplicación	
FLUJO BASICO:	
ACTOR	SISTEMA
1 Click en la opción de recordar usuario, la opción de recordar contraseña debe estar deseleccionada.	
2 Iniciar Sesión (IM_CU_01)	
	3 Crear/Actualizar registro en la base de datos nativa de la aplicación.
Poscondición: El actor ha otorgado permisos a la aplicación para que esta almacene sus datos excepto su contraseña.	
Puntos de Extensión: N/A	

Apéndice Q. Caso de Uso - Recordar Nombre de Usuario

ID: IM_CU_05	Caso de uso: Enviar Mensaje
Descripción: Permite acceder al sistema	
Requerimiento: IM_RF_02 (Enviar Mensaje)	
Precondición: El actor debe haber iniciado sesión.	
FLUJO BASICO:	
ACTOR	**SISTEMA**
1 Click sobre un contacto	
2 Redactar mensaje	
3 Click en Enviar	4 Verifica el dato introducido
	5 Empaquete el mensaje
	6 Envío del mensaje al servidor
FLUJOS ALTERNOS	
ACTOR	**SISTEMA**
	4.1 Campo vacío
4.2 Volver al paso 2	
	6.1 No hay conexión. Indicar el estado del mensaje.
Poscondición: El Actor ha enviado un mensaje a un contacto	
Puntos de extensión: N/A	

Apéndice R. Caso de Uso - Enviar Mensaje

ID: IM_CU_06	Caso de uso: Editar Perfil
Descripción: Permite acceder al sistema	
Requerimiento: IM_RF_3 (Editar Perfil)	
Precondición: El actor debe haber iniciado sesión.	
FLUJO BASICO:	
ACTOR	**SISTEMA**
1 Click en la barra del usuario o en Configuración	
2 Editar uno o mas campos	
3 Confirmar	4 Empaquete los datos ingresados.
	5 Envío del paquete IQ al servidor
FLUJOS ALTERNOS	
ACTOR	**SISTEMA**
	5.1 No hay conexión.
Poscondición: El Actor ha actualizado uno o más datos personales	
Puntos de extensión: N/A	

Apéndice S. Caso de Uso - Editar Perfil