Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Servicio de Gestión de Sesiones de Usuarios en Ambientes Móviles y Heterogéneos

Tesis que presenta

Ing. Eduardo Antonio Dávalos Camarena

para obtener el Grado de

Maestro en Ciencias en Computación

Director de la Tesis

Dr. José Guadalupe Rodríguez García

México, D.F. **Octubre 2009**

Resumen

El área de cómputo móvil, ha ido evolucionado gracias a la introducción de nuevas tecnologías. El uso de tecnologías inalámbricas se ha vuelto indispensable en las organizaciones, y poco a poco se vuelve más popular gracias a la introducción de nuevos dispositivos móviles en el mercado. Actualmente, estos dispositivos ya cuentan con tarjetas de red inalámbricas, tecnología *Bluetooth*, mayor capacidad de procesamiento, entre otras características. Lo anterior otorga la capacidad a los usuarios de acceder a redes inalámbricas desde sus dispositivos móviles, y eventualmente tener acceso a servicios como correo electrónico, envío de mensajes instantáneos y transferencia de archivos, entre otros. Esto da como resultado que los usuarios con dispositivos móviles quieran tener acceso, a los mismos servicios a los que pueden acceder con sus *Laptops* o computadoras de escritorio. Los usuarios pueden disponer de varios dispositivos. Esto conlleva, a que se pueda dar preferencia a la ejecución de algunos servicios en alguno de sus dispositivos en especial y que puedan acceder a los servicios con más de un dispositivo simultáneamente.

En esta tesis presentamos un servicio de administración de sesiones activas en uno o varios dispositivos simultáneamente para un mismo usuario. Además, este servicio permite al usuario dar preferencia al tipo de tráfico que desea recibir en sus diferentes dispositivos, por ejemplo, puede definir que las peticiones de llamadas de voz sobre IP (VoIP) sean enviadas primero a su *Smartphone*, si no está activa la sesión en este dispositivo, entonces las peticiones se dirijen al Asistente Personal Digital (*Personal Digital Assistant* - PDA), y de no estar activo éste, las peticiones se envían eventualmente a su *Laptop*.

El usuario además puede interactuar con otros usuarios del sistema de manera transparente, por ejemplo, enviar un mensaje de texto a otro usuario, sin necesidad de saber en qué dispositivo se recibe, pero sí con la certeza de que el usuario destino recibe el mensaje en caso de tener su sesión activa en al menos un dispositivo.

Este servicio se ejecuta a través de una aplicación creada usando como base el Protocolo de Inicio de Sesión (SIP) [1], ya que este ayuda a resolver los problemas ligados a la heterogeneidad de dispositivos, es decir, puede ser utilizado tanto en dispositivos móviles (*Smartphones* y PDAs), como en dispositivos con mayor capacidad (*Laptops* y computadoras de escritorio), con ayuda del lenguaje de programación *Python* que proporciona las herramientas necesarias para brindar portabilidad a la aplicación. Además, nuestro sistema presenta una arquitectura híbrida, ya que combina los mecanismos cliente-servidor (C/S) [2] y punto a punto (*Peer to Peer* - P2P)

[3]. Finalmente la aplicación puede ejecutarse en dispositivos que estén conectados a la red de manera alámbrica o inalámbrica. Con esta combinación se logra que el usuario pueda acceder a los servicios de presencia, transferencia de archivos, mensajes de texto, y llamadas de VoIP de una manera muy cómoda. El usuario puede además ejecutar su aplicación e iniciar sesión en uno o más dispositivos al mismo tiempo, cambiar las preferencias de los servicios con respecto a sus dispositivos y comunicarse de manera transparente con otros usuarios de la aplicación.

Para la evaluación de nuestro sistema, realizamos pruebas dentro de la red del departamento de computación del CINVESTAV, creando varias sesiones de usuarios y asignándoles diferentes dispositivos. De esta manera, comprobamos la utilidad de que el usuario pueda tener una sesión activa en varios dispositivos simultáneamente, y de poder elegir el tipo de tráfico que se quiere recibir en los distintos dispositivos.

Abstract

Thanks to the evolution of new technologies the mobile computing is gaining a lot of popularity. Wireless technology are becoming necessary into several kind of organizations and people use it everyday everywhere. As result today there exist different kinds of mobile devices in the market many of them with several ways to get a connection: through Bluetooth, Wi-Fi or GPRS.

Given the availability in connectivity and the fact that mobile devices have more capabilities of processing users want to get services like e-mail, instant message, file transfer, etc. i.e. the same services they have in a Personal Computer or Laptop. However users can have several devices connected at the same time and maybe they want to give preference to some devices for some type of services or use some devices for a specific service.

In this work we present an administration service for active sessions of users over one or several devices connected in a simultaneous way. Moreover our service allows to specify the preference about the kind of traffic to be received for device, i.e. user can specify that IP Voice (VoIP) sessions must be received in the smartphone in first place. If phone device is not ready VoIP sessions can be routed to the some other capable VoIP device.

Interaction between users is transparent, for example, users can send text messages without need to know which device is being used by the addressee user, but they have the certainty that user at destination have at least one open session over some device.

Our service works through an application developed over the Session Initiation Protocol (SIP) [1]. SIP is a helpful protocol for solving the problems related to the heterogeneity of devices. It can be used over mobile devices, PC's, or Laptops. Together with Phyton programming language give us the tools for the development of portable applications.

The service we are proposing works through the client-server (C/S) [2] and peer-to-peer (P2P) [3] communication mechanisms. This is necessary because users must to interact between them and with the server.

The application can be executed over devices connected on wireless or wired networks . On any of these networks users can have access to services like: presence, file transfer, text messages, and VoIP calls, without to be concerned about physical location of users. User can execute the application and start the session over one or different devices at same time. It is possible to change the user's preferences on devices and start communication at any time.

We have testes our application on the network of Computing Sciences of CINVESTAV, where we have created several user's counts working in different sessions on several devices. In particular we have tested the possibility of having opened several sessions in a simultaneously way on different devices and choosing the kind of traffic for every available device.

Agradecimientos

Agradezco a mis padres por haberme apoyado durante todo este tiempo, por haber respetado mis decisiones, por darme las bases para lograr ser quien soy y desvelarse conmigo en los tiempos difíciles. A mi papá Antonio Dávalos Piña por enseñarme la virtud de la paciencia, los buenos hábitos y a nunca rendirme. A mi madre Alma Camarena Reyes, por apoyarme en esos momentos difíciles donde todo parecía tan oscuro, y por enseñarme a luchar por lo que creo.

A mi hermana Alejandra Dávalos Camarena por haberme acompañado en momentos cruciales de mi vida, por compartir conmigo su creatividad y por haberme brindado gran parte de su tiempo.

A Idalia, por darme tanto amor y cariño a pesar de la distancia. Me recordaste las cosas más importantes de la vida y me has apoyado todo momento desde el día en que te conocí.

A mis abuelos maternos Lalo Camarena y Gloria Reyes, porque han sido como mis segundos padres. A mis tíos y primos Camarena en especial Edu, Koreano, Daniel, Ricardo, Eduardo, Alicia y Blanquis, que están conmigo en todo momento. A mi familia paterna, en especial a mi abuelita Eva Piña, Fer, André, Estéfani, Anselmo, Juan, Martín, Pato, Paty y Andrea, pues aunque no los veo tan seguido, siempre han estado al pendiente de mi y me han brindando su apoyo incondicional. También a mi familia de Ixtlán del Río.

A mis mejores amigos Christian, Madai y Cheche que estuvieron conmigo estos dos años de maestría. A mi amigo Alejandro de la ESCOM que siempre me ha apoyado, y no menos importantes a mis amigas Brenda, Lucero, Alix, Anamar, Marifer, Blanca y Belem, y a mis amigos Fome, Rafa, Luis, Mario, Jorge, July y Adrián.

A mi asesor el Dr. José G. Rodríguez García por la confianza que tuvo en mí a lo largo de la tesis, por compartir sus conocimientos conmigo, por sus comentarios y tiempo dedicado en revisar mi trabajo.

A la doctora Sonia Mendoza Chapa y al doctor Manuel Aguilar Cornejo por darse el tiempo para revisar y aportar ideas para el desarrollo de la tesis.

A Sofy pues desde que entré al posgrado siempre me has brindado tu apoyo.

Al CONACYT por la confianza y el apoyo financiero durante mi estancia en el CINVESTAV.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

El desarrollo de dispositivos móviles como teléfonos celulares y Asistentes Personales Digitales (*Personal Digital Assistant* - PDA) ha brindado una mayor capacidad en cuanto a *hardware* y *software*. Muchos de estos dispositivos ya cuentan con mayor poder de procesamiento, una aceptable capacidad de almacenamiento, y tarjetas de red inalámbricas [4]. Gracias a este avance y a la aparición de las redes inalámbricas [5] ha surgido el cómputo móvil. Hoy en día es muy común observar organizaciones en donde los empleados utilizan PDAs, *Smartphones*, *Laptops* y en ocasiones más de un dispositivo simultáneamente. Es además muy común observar que en las organizaciones utilicen tecnologías inalámbricas y que tengan acceso Internet [6].

Lo anterior permite a los usuarios tener acceso a redes inalámbricas desde sus dispositivos móviles [7], y esto conlleva a que puedan acceder a servicios como correo electrónico, *chat*, transferencia de archivos, entre otros, desde distintos dispositivos [8].

1.1. Antecedentes

La convergencia en la tecnología celular [9], tiende hacia un servicio basado completamente en el Protocolo de Internet (IP). De acuerdo a la evolución de la tecnología celular, en el 2006 fue introducida la generación 3.5 G, la cual ya soporta una tasa de transferencia de 10 - 50 Mbps, además de agregar la capacidad de acceso a Internet y permitir *High Speed Wireless Internet* (Internet inalámbrico de alta velocidad), incluyendo la descarga de archivos multimedia. La tendencia apunta entonces a una cuarta generación (4G), que se piensa será introducida a partir del 2010 y tendrá un soporte para la transferencia de datos de 100Mpbs a 1Gbps, completamente orientada a IP [4].

De acuerdo con esta tendencia, es posible que dentro de pocos años sea indispensable para los teléfonos celulares contar con tarjetas de red inalámbricas, y esto permitirá que los usuarios puedan tener acceso a Internet en todo momento y en cualquier lugar, sin necesidad de utilizar la tecnología celular [4].

El Protocolo de Inicio de Sesión (SIP) [1] da soporte a diferentes tipos de servicios

como presencia, mensajes de texto, llamadas VoIP, entre otros [10]. Además, SIP puede ser utilizado no sólo en computadoras, sino también en dispositivos tipo PDAs y *Smartphones* [11]. Definido en 1996 por la Fuerza de Trabajo de Ingenieros de Internet (IETF por sus siglas en inglés), es el resultado de la fusión de los proyectos Protocolo de Invitación de Sesión (SIPv1), desarrollado por Eve Schooler y Mark Handley, y Protocolo Simple de Invitación a Conferencia (SCIP) desarrollado por Henning Schulzrinne [12], SIP fue pensado originalmente como un protocolo que permitiera que la telefonía fuera un servicio más de Internet [10].

A pesar de la idea original de SIP, este protocolo es más que un simple protocolo de aplicación para la telefonía, ya que se complementa con protocolos como el Protocolo de Transporte en Tiempo Real (RTP) [13] y el Protocolo de Descripción de Sesiones (SDP) [14]. Como resultado podemos considerar a SIP como un *Framework* para el desarrollo de sistemas de comunicaciones [11].

Los mecanismos de comunicación cliente-servidor (C/S) [2] y punto a punto (P2P) [3] son muy populares hoy en día para el desarrollo de aplicaciones distribuidas. Debido a esto, necesitamos definir una arquitectura híbrida, combinando ambos mecanismos de comunicación, de tal manera que los usuarios puedan conectarse a un servidor que les proporcione servicios básicos (envío de mensajes de texto, transferencia de archivos y llamadas VoIP), y a su vez puedan interactuar con otros usuarios sin necesidad de pasar la información a través de un servidor.

1.2. Planteamiento del problema

Existen diferentes tipos de aplicaciones de mensajería instantánea como Messenger, Kopete, aMSN [62], Skype [63], Nimbuzz [64] y otras más, que permiten a los usuarios acceder a los servicios de envío de mensajes de texto, intercambio de archivos, llamadas de VoIP, y otros servicios. Algunas de estas aplicaciones se encuentran disponibles en su versión para dispositivos móviles.

Apesar de lo anterior, casi ninguna de estas aplicaciones le permite al usuario iniciar su sesión en varios dispositivos simultáneamente, además, ninguna permite a los usuarios administrar el tipo de tráfico que deseen recibir en sus diferentes dispositivos.

El sistema que proponemos, soporta los servicios de VoIP, presencia, transferencia de archivos y mensajes de texto, por medio de una aplicación que puede ser ejecutada en diferentes plataformas. Esta aplicación además da al usuario la capacidad de poder especificar las preferencias al tipo de tráfico que desee recibir en sus dispositivos, e interactuar con otros usuarios.

Para lograr lo anterior, realizamos el diseño de una arquitectura híbrida, que además de combinar los beneficios de las arquitecturas C/S y P2P, y explotar algunas de las herramientas que brinda SIP, permite a los usuarios interactuar entre sí, dando el soporte para la ejecución de la aplicación en diferentes sistemas operativos: sistemas no embebidos como *Windows* y *Linux*, y embebidos como *Symbian* y *Windows Mobile*. Además, soporta comunicaciones a través de medios alámbricos e inalámbricos.

De lo anterior, obtenemos como resultado que la arquitectura creada debe administrar las sesiones de los usuarios en un ambiente heterogéneo y móvil.

1.3. Propuesta de solución

El servicio que proponemos está constituido por siete componentes, que en conjunto conforman la arquitectura híbrida mencionada y le brinda al usuario las bondades descritas anteriormente. De manera general, nuestra propuesta se presenta a continuación:

La **Aplicación** permite al usuario tener acceso a todos los servicios, el usuario podrá realizar y responder peticiones de servicio.

Los usuarios pueden mantener hasta cuatro sesiones activas (suponiendo que cuenten con una PC, una *Laptop*, un PDA y un *Smartphone*), esto se logra gracias a una **Sesión Virtual**, que se encarga de mostrar la misma información al usuario en todos sus dispositivos.

El **Dispositivo**, se encarga de establecer y mantener activas las sesiones entre el dispositivo y el servidor y entre distintos dispositivos. Una sesión entre un dispositivo y el servidor, se conoce como **Sesión Real (SR)**, por lo tanto, un usuario puede tener hasta cuatro SR activas.

Cuando dos dispositivos se comunican entre sí, se lleva a cabo una **Sesión de Comunicación**, esta sesión se lleva a cabo usando el mecanismo de comunicación P2P [15].

El usuario puede tener la certeza de que las peticiones y las respuestas son enviadas y recibidas de manera satisfactoria y cuando esto no sucede, la aplicación no se bloquea ni produce errores, ya que de esto se encarga el componente de **Red**.

Finalmente nuestra propuesta tiene como base un **Servidor**, que se encarga de mantener el control de las sesiones y de la información de todos los usuarios que cuenten con esta aplicación dentro de la misma red.

1.4. Objetivo general

El objetivo principal de nuestro trabajo, consiste en crear una aplicación para dispositivos móviles que proporcione un servicio de administración de sesiones de usuario, con el fin de incrementar la eficiencia en las comunicaciones entre usuarios.

1.5. Objetivos particulares

Existe una serie de objetivos particulares, que en conjunto nos llevan a la obtención del objetivo principal, estos se enlistan a continuación:

- Utilizar tecnologías de vanguardia para el desarrollo de sistemas móviles.

- Utilizar la tecnología de red inalámbrica para administrar la interacción entre usuarios.

- Presentar un servicio que haga frente a los problemas presentes en los ambientes heterogéneos.

- Utilizar mecanismos para la implementación de sistemas en ambientes móviles.

1.6. Metodología

Para alcanzar los objetivos, se cumplió la siguiente metodología:

- Revisar documentación sobre el desarrollo de aplicaciones móviles, así como también documentación sobre SIP, sus utilidades, su funcionamiento, y los protocolos sobre los cuales trabaja.

- Revisar documentación acerca del desarrollo de aplicaciones móviles, lenguajes de programación, e implementaciones de SIP en sistemas heterogéneos.

- Llevar a cabo la lectura del estado del arte, trabajos relacionados, estudio de herramientas con características similares al proyecto.

- Realizar la instalación del *API PJSUA* en distintas plataformas, y probar las aplicaciones con que cuenta para los diferentes dispositivos.

- Realizar comparativas sobre lenguajes de programación y bibliotecas gráficas, con el fin de seleccionar las más adecuadas para el desarrollo de la aplicación en los diferentes dispositivos.

- Realizar una interfaz gráfica que pudiera servir de soporte a SIP y que permita de manera amigable el uso de los servicios.

- Crear la arquitectura híbrida utilizando la interfaz anterior para la manipulación de las sesiones de los usuarios.

- Instalar el servidor en una red local y realizar pruebas con diferentes dispositivos y usuarios.

- Reportar los datos obtenidos en cuanto a desempeño y utilidad de la aplicación.

- Investigar posibles aplicaciones futuras en las que pueda servir nuestro trabajo.

1.7. Contenido

El resultado de la metodología se presenta en este documento, organizado en siete capítulos de la siguiente manera:

- En el capítulo 2 presentamos un análisis de los antecedentes y el estado del arte con respecto al proyecto, exponemos de una forma clara el planteamiento del problema y finalmente los trabajos relacionados con nuestro proyecto.

- En el capítulo 3 mostramos las herramientas que utilizamos para el desarrollo del proyecto, la comparativa entre lenguajes de programación y finalmente los protocolos que intervinieron en el desarrollo del proyecto junto con las herramientas que utilizamos para explotar los mismos.

- En el capítulo 4 vemos una descripción detallada de la arquitectura del sistema, abordando los componentes de la misma y explicando las funciones de la arquitectura híbrida por separado.

- En el capítulo 5 explicamos la manera en que se debe instalar nuestra aplicación tanto en la parte del servidor, como en los clientes, además mostramos el funcionamiento de cada uno de los módulos de la misma.

- En el capítulo 6 presentamos las pruebas y el análisis de los resultados, y una comparativa de los mismos con respecto a trabajos relacionados, concluyendo el mismo con las ventajas y desventajas del sistema.

- Finalmente en el capítulo 7 concluimos la tesis y mostramos una visión de posible trabajo futuro que pudiera realizarse utilizando la aplicación como base o como extensión de otra.

Capítulo 2

Sesiones de usuarios en ambientes móviles y heterogéneos

En este capítulo presentamos las principales características de los ambientes móviles y heterogéneos. Mostramos además una visión general de las tecnologías existentes que brindan apoyo a la movilidad y la heterogeneidad en los ambientes de trabajo actuales.

Abordamos y combinamos los diferentes conceptos de sesión existente, para conformar las sesiones de usuarios dentro de ambientes móviles y heterogéneos.

La movilidad es uno de los factores más importantes en los ambientes laborales, gran cantidad de empresas en todo el mundo otorgan a sus empleados dispositivos móviles (teléfonos celulares o PDA's), para mantener contacto con ellos y para facilitarles ciertas operaciones laborales. Dichos dispositivos permiten a los usuarios desplazarse dentro y fuera de la organización sin necesidad de mantenerlos conectados por cable como sucede con las PC convencionales.

Sin embargo, es común que los usuarios cuenten con más de un dispositivo tanto en el hogar como en la organización en la que laboran (e.g. PC, *Laptop*, teléfono celular, etc.). Por lo tanto, surge en los usuarios, el interés de poder tener todos los servicios con los que cuenta en su PC, en todos sus dispositivos (e.g. correo electrónico, mensajería instantánea, transferencia de archivos, etc.).

2.1. Conceptos básicos

Antes de comenzar el abordaje de la tesis, es necesario que el lector conozca algunos conceptos importantes para el desarrollo de nuestro trabajo:

- **Sesión de comunicación**: es el intercambio de información que se da entre dos entidades.
- **Sesión de usuario**: es una entidad lógica que por medio de un identificador y una contraseña, le permite al usuario acceder a uno o más servicios por medio de una aplicación.

- **Usuario**: es la entidad física que cuenta con uno o más dispositivos móviles y que puede iniciar una o varias sesiones de usuario simultáneamente.
- **Dispositivo**: es el medio físico en el cual se instala la aplicación que le permite al usuario acceder a los servicios.
- **Cuenta de usuario**: es una entidad lógica que engloba los datos del usuario; es necesaria para que éste pueda acceder a los servicios que le brinde cierta aplicación.
- **Sesión real**: es la comunicación que se presenta entre el dispositivo y el servidor. Decimos que se establece una sesión real entre el dispositivo y el servidor, cuando un usuario inicia sesión desde nuestra aplicación. Existe una sesión real por cada dispositivo.
- **Sesión virtual**: es la representación que se le da a los datos que se reciben en una sesión real. De esta manera el usuario tiene la impresión de tener una misma sesión real en diferentes dispositivos. Se tiene una sesión virtual por usuario (no importando en cuantos dispositivos tenga activa una sesión real).

Estos son los conceptos básicos que aparecen a lo largo de la tesis. Sin embargo, a través de la misma, irán apareciendo más conceptos de los cuales daremos su explicación en su momento.

2.2. Movilidad y tecnologías móviles

El manejo de la información es de vital importancia para toda organización y en general, para la mayoría de las personas. Gracias a Internet, las personas pueden acceder, modificar y compartir la información, sin necesidad de que se encuentren en una localidad específica, es decir, que se puede acceder a la información desde cualquier lugar que tenga acceso a Internet.

El concepto de movilidad ha madurado, gracias a la introducción de las redes inalámbricas, las personas pueden acceder a la información sin necesidad de utilizar cables [5], esto vuelve aún más versátil la manipulación de la información. Con la introducción de las tecnologías móviles, las personas pueden acceder desde distintos dispositivos a la información. Hoy en día es común que una persona pueda navegar en Internet con ayuda de un teléfono celular de última generación (*Smartphone*), o un Asistente Digital Personal (PDA) y otros dispositivos como *IPods*, *Play Station Portable (PSP)*, *Nintendo DS*, etc.

Estos dispositivos cuentan con una capacidad de procesamiento y almacenamiento superior a la que existía en las PC estándares de los años 80's y principios de los 90's, inclusive, ya cuentan con tarjetas de red inalámbrica, antenas de Sistema de Posición Global (*Global Position System GPS*), tarjetas gráficas muy poderosas, entre otras características.

Por lo anterior, se creó una nueva rama de la computación conocida como cómputo móvil. El cómputo móvil se encarga de la creación de aplicaciones para dispositivos móviles (con poca capacidad de almacenamiento, bajo consumo de batería y en la mayoría de las veces, con conexión inalámbrica) [16]. Empresas como *Nokia*, *Microsoft* y *Macintosh* entre otras, desarrollan herramientas que permiten a los programadores crear aplicaciones que exploten las características de los dispositivos.

Tener un dispositivo móvil implica muchas ventajas para los usuarios, no solo la eliminación de cables, sino también la manipulación de la información de manera más cómoda y práctica. Gracias al cómputo móvil y al desarrollo de las aplicaciones móviles, día con día se incrementa la cantidad de usos que se le puede dar a los dispositivos. Actualmente las aplicaciones van desde los juegos, hasta aplicaciones que implican transacciones bancarias, o módulos de sistemas distribuidos. En la Figura 2.1 se puede apreciar un ejemplo de movilidad, en donde un usuario puede acceder a la información desde un dispositivo móvil, sin necesidad de permanecer en una localidad fija.

Figura 2.1: Usuario con un dispositivo móvil puede utilizar sus beneficios en "cualquier" ubicación

2.3. Tecnologías inalámbricas

Desde la introducción de las *Laptops* y los primeros teléfonos celulares, se pretendía eliminar el uso de los cables. Aunque se presentan desventajas como el incremento en la pérdida de información y la reducción de velocidad de transferencia en comparación con las tecnologías alámbricas, las principales ventajas, son la eliminación de cables y la movilidad [5].

Gracias al uso de las redes inalámbricas, y al avance de la tecnología celular, los dispositivos móviles pueden acceder a Internet. Esto genera como resultado una nueva necesidad: que los usuarios puedan acceder a los servicios con los que cuentan en sus PC estándares y Laptops [17], desde sus dispositivos móviles (e.g., correo electrónico, transferencia de archivos, etc.).

Existen tecnologías inalámbricas como *Worldwide Interoperability for Microwave Access (WIMAX)*, con los estándares 802.16 del IEEE [18], que se están desarrollando actualmente, y que permiten un alcance de hasta 50 km de distancia entre los puntos de acceso y los equipos móviles.

La mayoría de los dispositivos móviles cuentan con distintos tipos de conexión inalámbrica [19], entre los más usados encontramos: Infrarrojo (*Infrared Data Association - IrDA*), *Bluetooth* (con los estándares 802.15 de la IEEE), Servicio General de Radio por Paquetes (GPRS), GPS y *WiFi* (con los estándares de la IEEE 802.11) [6].

2.3.1. Infrarojo

El método de transmisión infrarrojo es uno de los tipos de conexión inalámbrico con los que cuentan gran cantidad de dispositivos móviles modernos, sin embargo, poco a poco ha sido desplazado por el *Bluetooth*, las principales características del método infrarrojo son las siguientes [6]:

- Presenta un rango típico de separación entre emisor y receptor de 5 a 60 cm, aunque esto depende mucho del dispositivo.
- Los transceptores (transmisor-receptor) se comunican por pulsos infrarrojos con un ángulo de inclinación de hasta 15 grados.
- El tipo de comunicación es *half-duplex*, es decir, que los datos solo viajan en un solo sentido, del emisor al receptor, sin embargo se puede emular la comunicación *full-duplex* cambiando el sentido de la comunicación en un momento dado con algún evento especial, además el puerto de comunicación es el RS-232.

2.3.2. Bluetooth

Es la especificación 802.15.1 de la IEEE, se utiliza para la transmisión de datos por radiofrecuencia a corto alcance [20]. La principales ventajas que presenta *Bluetooth* son la facilidad de comunicar equipos móviles y fijos, eliminación de cables, además proporciona la capacidad de brindar nuevas funciones a otros dispositivos, por ejemplo, la capacidad de crear una red inalámbrica que trabaje a través de *Bluetooth* [6].

La mayoría de los dispositivos que cuentan con un tipo de conexión *Bluetooth*, tienen un alcance de hasta 10 mts, esto es debido a que la mayoría de esos dispositivos utilizan baterías y ampliar el rango de comunicación implicaría un consumo mayor de energía, sin embargo, el alcance de *Bluetooth* puede ser de hasta 100 mts (que sería muy similar al de la tecnología *WiFi*, pero que consumiría demasiada energía comparado con este último).

2.3.3. Red inalámbrica (WiFi)

El protocolo *WiFi* o IEEE 802.11 es un estándar que define el uso de los dos niveles más bajos de la arquitectura del modelo OSI (capas física y de enlace de datos). Existen diversas variaciones del protocolo (802.11a, 802.11b, 802.11g, etc.) de acuerdo a la velocidad de transmisión de datos soportada, y al método de seguridad implementado [6].

El estándar 802.11n promete ser el mejor, soportando una velocidad de hasta 600 Mbps y trabajando con las frecuencias 2,4 GHz (utilizada por 802.11b y 802.11g) y 5 GHz (utilizada por 802.11a).

Actualmente *WiFi* es uno de los estándares más utilizados (la versión 802.11b y 802.11g) [17], aunque ya existen equipos que soportan una versión beta de 802.11n, alcanzado una velocidad de transferencia de datos de hasta 300 Mbps. *WiFi* es compatible con todas las *Laptops* que se construyen actualmente, y con los nuevos dispositivos móviles como *Smartphones*, PDA's, *IPods* y hasta consolas de videojuegos, que ya incorporan este estándar. Esto nos indica una clara tendencia a manipular toda la información a través de Internet.

2.4. Tecnología celular

Si hablamos de la tecnología celular, podemos hacer una comparación de las distintas generaciones que se han desarrollado, y podemos mostrar la tendencia que está siguiendo la misma.

Algunos autores consideran una generación 0 (0G), que aparece en la segunda guerra mundial y que fue lanzada por *Motorola* como "*Handle Talkie* H12-16", que fuese un dispositivo que transmitía ondas de radio en una frecuencia de 550 Mhz, y que alcanzaría su apogeo en la década de los 50 y 60 con el lanzamiento de los famosos "*Walkie Talkie*" [6].

La Primera Generación (1G) de teléfonos inalámbricos consiste en teléfonos celulares análogos; es implementada a mediados de los 70's, sin embargo, esta generación tenía poca seguridad, ya que una persona con un sintonizador de radio podría escuchar llamadas ajenas [6]. La segunda generación (2G) fue introducida en 1991, se caracteriza por la introducción de teléfonos digitales y mensajes cortos [4].

A partir de la introducción de la generación 2.5 (2.5G) en 1999, con mayor capacidad (en velocidad de transferencia de datos) que la 2G [4] y manejo de datos en forma de paquetes, comenzó a mostrarse cierta tendencia en la tecnología celular hacia el envío de datos con mayor velocidad. Con la introducción del GPRS que proporciona datos por conmutación de paquetes a través de las redes del Sistema Global para comunicaciones Móviles (GSM) [9], aparece la ventaja de poder asignar diferentes tipos de datos al mismo canal de comunicación, y esto trae como resultado, la invención de navegadores Web para los teléfonos celulares, para que se pudiera acceder a Internet por medio del estándar GPRS [6].

Para el año 2001, aparece la tercera generación (3G) con una tasa de transferen-

cia de 2Mbps y soporte de video, voz y datos por el mismo canal, introduciéndose el estándar de Sistema Universal de Telecomunicaciones Móviles (UMTS) [21], que mejora por mucho, las velocidades de transmisión, y reduce las interferencias [6].

Para el 2006 se implementa la generación 3.5G que soporta una tasa de transferencia de 10 - 50Mbps y brinda la capacidad de acceso a Internet con soporte de Internet Inalámbrico de Alta Velocidad (*High Speed Wireless Internet*), incluyendo la descarga de archivos multimedia e incorporando televisión digital móvil. Actualmente los navegadores Web de los celulares, ya cuentan con reproductores *Flash* para reproducción de archivos multimedia, y soportan conexiones por USB, inalámbrica (*WiFi*, en caso de que el equipo cuente con tarjeta de red inalámbrica) o por antenas celulares (usando el servicio que provee la empresa de telefonía celular) [4].

La tendencia apunta hacia una cuarta generación (4G) que se piensa será introducida en el 2010 y tendrá un soporte para la transferencia de datos de 100 Mbps a 1Gbps y será completamente orientada a IP [21]. Esto podría estar muy de la mano con la introducción de los estándares inalámbricos WIMAX mencionados anteriormente, para el soporte de la comunicación celular.

2.5. Ambientes distribuidos

Desde la introducción de las redes de computadoras, y con su divulgación a nivel mundial que se ha venido dando desde el siglo anterior, poco a poco se han ido desplazando a los sistemas centralizados, para introducir aplicaciones y sistemas distribuidos.

Un sistema distribuido es aquel en donde los elementos del mismo se encuentran separados y comparten información a través del paso de mensajes, de manera transparente para los usuarios, es decir, aparentando un sistema centralizado. Los sistemas distribuidos tienen la ventaja de poder ejecutar diversas funciones de manera simultánea, ahorrando trabajo y tiempo a los usuarios del mismo [3].

Un ambiente distribuido por lo tanto, implica la interacción entre aplicaciones y usuarios, haciendo que estos puedan comunicarse y resolver un problema de manera más eficaz que siendo un solo usuario en un ambiente centralizado. Hoy día, se pueden encontrar distintos tipos de aplicaciones distribuidas, que van desde juegos, editores de texto, hasta aplicaciones que se dedican a resolver algoritmos matemáticos.

Gracias a la introducción de los sistemas distribuidos, se deriva un gran número de conceptos en que los usuarios pueden explotar este tipo de sistemas, de entre los conceptos más importante encontramos a los sistemas colaborativos y el cómputo ubicuo.

Los sistemas colaborativos (*groupware*) son utilizados por grupos de personas que trabajan con la misma información pero que se encuentra distribuida [22], existen editores de texto, y juegos como *Quake* o *Microsoft Age of Empires*, que son ejemplos de groupware, ya que la información se comparte entre los usuarios y pueden modificarla simultáneamente. Algunos sistemas colaborativos, insertan el concepto de conciencia de grupo, que se refiere a que cada usuario pueda conocer el estado (definido por el

sistema) del resto de los usuarios [23].

El cómputo ubicuo es la integración de las tecnologías de la información en la vida diaria, es decir, que los usuarios perciban a las computadoras como un elemento más del ambiente en que se encuentran [24]. Para esto, el cómputo ubicuo propone la inserción de componentes computacionales que interactúen con las personas y sus actividades cotidianas de manera natural [25].

2.5.1. Mecanismos de comunicación cliente-servidor (C/S)

Cuando hablamos de un sistema distribuido, nos referimos a dos o más dispositivos que funcionan en conjunto para lograr un objetivo común [3]. Para que estos dispositivos se comuniquen entre sí, debe existir un mecanismo de comunicación estándar entre ellos. Uno de los mecanismos de comunicación más populares que se utiliza para la elaboración de sistemas distribuidos, es el mecanismo de comunicación C/S [2].

El mecanismo C/S funciona de la siguiente manera: existe en principio uno o más equipos llamados servidores, que por medio de una aplicación, proporcionan diversos servicios (e.g., servidores de tiempo, servidores de noticias, etc.), existen además otros equipos conocidos como clientes, los cuales pueden enviar o recibir información de los servidores dependiendo la aplicación.

Para que el mecanismo C/S funcione correctamente, se debe definir la manera en la que el cliente obtendrá los servicios. Existen dos formas en las que el cliente puede obtener los servicios, ya sea a) *Pull*, que es cuando el cliente realiza una petición al servidor y este responde, o b) *Push*, que es cuando el servidor sabe a qué cliente enviar información sin que éste realice ninguna petición.

Un ejemplo de un mecanismo C/S que utiliza el método *Pull* es un servidor de tiempo: un equipo se conecta al servidor, realiza la petición y posteriormente el servidor responde con la hora actual [26].

Un ejemplo de un mecanismo C/S que utiliza el método *Push* es un servidor de noticias: un equipo se da de alta en el servidor y el servidor envía las noticias periódicamente, como es el caso del servicio de UNONOTICIAS de Telcel [65].

2.5.2. Mecanismos de comunicación punto a punto (P2P)

A pesar de la eficiencia del mecanismo C/S, existe un problema con el mismo. Cuando hay una gran cantidad de clientes en la red y el número de servidores es limitado, puede haber una saturación en el tráfico de paquetes por la red y los clientes pueden no recibir respuesta a las peticiones, o que dichas respuestas sean muy lentas.

Para corregir este problema, se creó el mecanismo de comunicación punto a punto (P2P) [3]. Este mecanismo propone que cada dispositivo que se encuentre dentro del sistema, pueda ser cliente y servidor simultáneamente. Otra de las diferencias con el mecanismo C/S es que un mecanismo P2P utiliza aplicaciones pares, mientras que en C/S se debe crear una aplicación *cliente* y una aplicación *servidor*.

Algunas aplicaciones que utilizan mecanismos P2P son por ejemplo *Napster*, *Ares* y *Limewire* [27] que permiten que varios usuarios compartan archivos entre sí. Otras

aplicaciones, aún más sencillas, son juegos para celular, en donde dos personas pueden conectarse por medio del *Bluetooth*, ejecutar el juego e interactuar entre sí.

El cuadro 2.1 [27] presenta las dos formas de desarrollo de aplicaciones existentes en P2P:

Cuadro 2.1: Tipos de aplicaciones P2P

Tipo	**Ventajas**	**Desventajas**	**Ejemplos**
P2P puro	No hay servidor central, Mayor tolerancia a fallas, Arquitectura muy simple.	Más consumo de recursos de red, Poca escalabilidad.	Gnutella Freenet
P2P mediado por un servidor	Menor consumo de recursos de red, Alta escalabilidad.	Dependencia de un servidor, Menor tolerancia a fallas.	Napster Limewire Kazaa

2.5.3. Mecanismos de comunicación basados en eventos

Estos mecanismos de comunicación pueden ser de tipo C/S o P2P, para poder definir su funcionamiento, primero es necesario mencionar el concepto de evento.

En la rama de tecnologías de la información, un evento es una acción que puede suceder en todo momento y que trae como consecuencia la ejecución de una o más funciones. Por ejemplo, en un teléfono celular, el evento *recibir un mensaje* puede activar un sonido, la vibración del teléfono y que además despliegue un mensaje en la pantalla.

La idea de implementar mecanismos de comunicación basados en eventos, es básicamente que la comunicación sea asíncrona, es decir, que en cualquier momento de la ejecución mientras los usuarios comparten el espacio de trabajo, puedan surgir eventos que ejecuten ciertas funciones en las aplicaciones de los usuarios [28].

2.6. Heterogeneidad

De acuerdo con el concepto de heterogeneidad [**?**], podemos definir un ambiente heterogéneo como un ambiente distribuido donde se suma la capacidad de poder incluir diferentes plataformas de manera simultánea.

La heterogeneidad es definida como la composición de un todo en partes de distinta naturaleza [66]. Si se traslada el concepto anterior a las tecnologías de la información, se pueden obtener dos tipos de heterogeneidad útiles para esta tesis.

Heterogeneidad de conexión, esta implica que los usuarios puedan conectarse y tener acceso a un servicio sin necesidad de utilizar un tipo de conexión específico, por ejemplo, que puedan acceder de manera alámbrica o inalámbrica.

Heterogeneidad de dispositivo, que propone que un usuario tenga acceso a un servicio desde diferentes tipos de dispositivos, por ejemplo: una PC estándar, una *Laptop*, un *Smartphone*, etc.

Dado lo anterior podemos definir un ambiente heterogéneo, como un sistema colaborativo, distribuido, en el cual los usuarios puedan compartir información, y puedan interactuar entre sí, no importando el tipo de conexión o el dispositivo con el cual accedan al sistema.

2.7. Ambientes móviles y heterogéneos

Una vez presentada la tecnología sobre la cual se sustenta nuestro trabajo, podemos definir un ambiente móvil y heterogéneo de la siguiente manera: es un ambiente al cual pueden acceder los usuarios a través de una aplicación que puede ser ejecutada en dispositivos con diferentes plataformas, conectándose de manera alámbrica o inalámbrica.

Podemos apreciar un ejemplo de un ambiente móvil y heterogéneo si observamos la Figura 2.2.

Figura 2.2: Un ambiente móvil y heterogéneo permite interactuar dispositivos con diferentes plataformas y con métodos de conexión distintos

2.8. Sesiones de usuarios y planteamiento del problema

Existen diversas aplicaciones que proporcionan diferentes servicios por medio del uso de sesiones de usuario. El concepto de sesión, se define de acuerdo al contexto en que se utilice. En el caso de nuestro trabajo, la sesión de usuario se considera como un mecanismo que permite al usuario acceder a cierto tipo de servicios. Un ejemplo de sesión de usuario se da cuando un usuario, que posee una aplicación de mensajería instantánea, puede iniciar sesión con los datos de su cuenta para poder acceder a los servicios que la aplicación le proporciona (correo electrónico, mensajería instantánea, transferencia de archivos, etc.).

En la mayoría de los casos, las sesiones aparecen como un mecanismo de manipulación de servicios único y bloqueante, esto quiere decir, que cuando un usuario inicia sesión en alguna aplicación para acceder a ciertos servicios, el usuario solo puede mantener activa dicha sesión, y si se llega a iniciar sesión en otro dispositivo, en la mayoría de las aplicaciones existentes, la sesión actual se cierra.

Es común que los usuarios tengan diversos tipos de cuentas para acceder a ciertos servicios, desde correo electrónico, hasta visualización de televisión por Internet. Por otro lado, como mencionamos anteriormente, en la actualidad la mayoría de los usuarios cuentan con más de un dispositivo y es común que un usuario desee tener los beneficios de sus servicios en cualquiera de sus dispositivos. Sin embargo, es difícil encontrar aplicaciones que den soporte para la heterogeneidad de plataformas, y permitan al usuario acceder a sus servicios de manera simultánea y en diferentes dispositivos.

De ahí podemos plantear la siguiente situación: podemos imaginar una aplicación con las funciones básicas de un servicio de mensajería instantánea (IM), es decir, transferencia de archivos, intercambio de mensajes instantáneos y llamadas de VoIP (ver Figura 2.3).

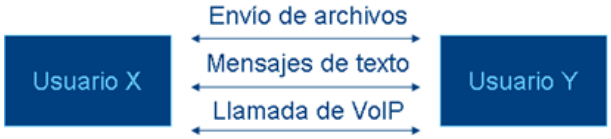

Figura 2.3: Tipo de servicios disponibles en una aplicación de Mensajería Instantánea estándar

Supongamos ahora, que el *UsuarioX* está recibiendo un archivo de algún otro usuario, pero eventualmente tiene que cambiar de ubicación. Si el *UsuarioX* pudiera iniciar sesión en su PDA o en su teléfono celular, la sesión activa en su computadora se cerraría automáticamente, y por lo tanto, se cancelaría la transferencia del archivo (ver Figura 2.4). Esto nos representa un problema, es decir, los usuarios no pueden tener más de una sesión activa, y esto los obliga a permanecer frente a un dispositivo o finalizar sus tareas con la posibilidad de que estas no hayan sido terminadas.

Figura 2.4: *UsuarioX* cambia de ubicación y la transferencia de archivos se cancela automáticamente

Siguiendo con el ejemplo anterior, si *UsuarioX* pudiera iniciar su sesión en más de un dispositivo, y mantener la sesión en ambos simultáneamente, entonces la transferencia del archivo seguiría en curso, y el usuario podría utilizar un servicio igual o diferente desde su otro dispositivo, mientras cambia su ubicación actual (ver figura 2.5).

Figura 2.5: *UsuarioX* mantiene su sesión de usuario activa en diferentes dispositivos de manera simultánea

2.9. Trabajos relacionados

Existen algunos trabajos que tocan ciertos puntos relacionados con este proyecto, de entre los mas importantes podemos encontrar los siguientes:

Vijay Gehlot y Anush Hayrapetyan desarrollaron un modelo formalizado para aplicaciones móviles, basado en el protocolo de presencia de SIP. Ellos toman la estructura del modelo de presencia de SIP dando detalles de su arquitectura y presentan su trabajo de *Colored Petri Net* (CPN) como un modelo ejecutable para dicha arquitectura [29].

Vega García propone su Sistema de Comunicación con Niveles de Servicio Basado en SIP para Ambientes Heterogéneos. En su trabajo, presenta una plataforma de VoIP que integra aspectos tales como: interoperabilidad entre diferentes sistemas operativos, heterogeneidad con respecto a la red, refiriéndose a alámbrica e inalámbrica, y

calidad de servicio, entre otros. Además de que la base del sistema está creada bajo el protocolo SIP [30].

Skulason Magnus propone en su tesis de maestría, que los dispositivos móviles puedan ser utilizados como proveedores de servicios Web. El crea un *framework* y una interfaz para el manejo y el control de acceso a los servicios Web por medio de dispositivos móviles [31].

Windows Live Messenger es probablemente una de las aplicaciones más utilizadas en la actualidad por la mayoría de los usuarios de Internet, Messenger permite como funciones básicas la mensajería instantánea, la transferencia de archivos, llamadas de voz sobre IP [67], e inclusive las video llamadas. En su última versión, *Windows Live Messenger* ya permite a los usuarios tener una sesión activa en diferentes computadoras al mismo tiempo.

Nimbuzz, es un proyecto que apareció en Mayo del 2008. Se encuentra disponible para dispositivos móviles, para computadoras y directamente en la Web. Brinda los servicios de mensajería instantánea, compartimiento de localización global (donde los usuarios pueden enviar sus coordenadas GPS), conferencias de VoIP y transferencias de archivos. *Nimbuzz* además fusiona tecnologías como *Messenger* (MSN), *Skype*, *Yahoo! Messenger*, *ICQ*, *GoogleTalk*, *AIM*, entre otros, en una sola sesión, es decir, el usuario inicia su sesión en *Nimbuzz* y automáticamente se activan las sesiones que haya dado de alta en el servicio. Un usuario de *Nimbuzz* puede realizar una conferencia de VoIP con usuarios que estén activos en servicios diferentes, por ejemplo, en *Skype* y en MSN [64].

Capítulo 3

Herramientas de desarrollo para un servicio móvil y heterogéneo

La desventaja que se encuentra en *Nimbuzz*, y en general en las aplicaciones del mismo género, es que **no** se puede iniciar una sesión en diferentes dispositivos simultáneamente. Una ventaja de esta posibilidad es la de permitir a los usuarios dar preferencia al tipo de tráfico que quieren para sus diferentes dispositivos.

Por lo anterior, nuestra propuesta es crear un servicio que proporcione a los usuarios la capacidad de iniciar sesiones en sus diferentes dispositivos móviles de manera simultánea. Este servicio además brinda soporte para su ejecución en ambientes *Windows*, *Linux*, *Symbian* y *Windows Mobile*, sin importar si el medio de conexión es por cable o inalámbrico, dando como resultado un servicio para ambientes móviles y heterogéneos.

3.1. Servicio propuesto

Como podemos observar, *"la problemática principal aparece cuando los usuarios que cuentan con más de un dispositivo, desean tener simultáneamente los mismos servicios en todos sus dispositivos"*. Tomando en cuenta que los usuarios prefieren ejecutar algunos servicios en ciertos dispositivos más que en otros, proponemos como solución a esta problemática, un **Servicio de Gestión de Sesiones de Usuarios** (SEGEUS), que brinde a los usuarios los siguientes servicios:

- **Servicio de Mensajes de Texto**: este servicio permite al usuario, conversar con otros usuarios por medio de una aplicación tipo *chat*.
- **Servicio de Transferencia de Archivos**: el servicio básico para el intercambio de archivos entre usuarios.
- **Servicio de Llamadas VoIP**: basado en el servicio de llamadas VoIP SIP, permite a los usuarios comunicarse por medio de los puertos de audio de sus dispositivos móviles.

- **Servicio de Presencia**: permite a los usuarios cambiar de estado (actualmente sólo se cuenta con los estados conectado y desconectado) y conocer el estado de los otros usuarios, además permite al usuario conocer los dispositivos en los cuales los usuarios tienen activa su sesión. Por otro lado los usuarios pueden cambiar su alias, incorporando así el concepto de conciencia de grupo.

- **Servicio de Preferencias**: permite a los usuarios definir preferencias de uso de dispositivo para ejecutar ciertas funciones (ver sección 5.3.5).

Nuestro servicio funciona a través de eventos, es decir, cada operación que realiza un usuario genera eventos, de los cuales algunos son notificados a los otros usuarios, y otros únicamente operan en el servidor.

Para la creación de nuestro servicio, lo primero que hicimos fue evaluar los lenguajes de programación existentes para aplicaciones móviles y su compatibilidad entre plataformas distintas, luego evaluamos las herramientas de desarrollo de interfaces para la creación de una Interfaz Gráfica de Usuario (GUI), posteriormente investigamos sobre las herramientas que explotan los servicios de SIP [10] para elegir la herramienta que más se adecuara a nuestras necesidades, finalmente elaboramos la aplicación que brinda los servicios arriba descritos y la denominamos SEGEUS.

3.2. Lenguajes de programación y aplicaciones móviles

Para el desarrollo de cualquier sistema, es necesario elegir un lenguaje de programación (LP) adecuado para el mismo [32]. Existe una gran gama de LP, que van desde el ensamblador, hasta lenguajes visuales. Algunos de los lenguajes más populares, son: *C*, *C++*, *Java* [33], *Python* [68], *Ruby*, *HTML*, *JXTA* (para desarrollo de aplicaciones P2P) [34], etc.

El desarrollo de aplicaciones móviles ha tenido un auge desde la introducción de la tecnología 2.5 G en los celulares (ver sección 2.4). A partir de entonces surge una nueva rama de la computación llamada cómputo móvil, que se encarga del desarrollo de aplicaciones móviles que explotan las características con que cuentan los dispositivos.

Dado que los dispositivos día con día incrementan sus capacidades (memoria, conectividad, procesamiento, etc.), el desarrollo de aplicaciones más poderosas conlleva el diseño de LP con mayores funciones. Esto ha llevado a que los LP que se utilizan hoy en día, para el desarrollo de aplicaciones móviles, sean versiones ligeras de los LP que se utilizan para el desarrollo de aplicaciones para *Laptops* y PCs convencionales.

Es por esta razón, que la elección de un LP adecuado fue uno de los puntos más importantes para el desarrollo de nuestro trabajo. Para elegir el LP, tomamos en cuenta las siguientes características:

- Los sistemas operativos en que se ejecuta la aplicación son: *Windows*, *Linux*, *Symbian S60* y *Windows Mobile*.

Cuadro 3.1: Comparativa de lenguajes de programación

Característica del Lenguaje	**Lenguaje**		
	Python	**Java**	**C y C++**
Dispositivo	En los tres casos existe una versión para cada sistema operativo		
Portabilidad	Muy portable (las funciones básicas se ejecutan sin modificar el código	Semiportable (con J2ME se debe modificar el código)	Semiportable (debe modificarse el código)
Tamaño en Kb	En los tres casos son aplicaciones ligeras (depende de la habilidad del desarrollador)		
Implementación de sockets	Es la misma en todas sus versiones	Difiere de acuerdo a la versión de Java	Difiere de acuerdo a los sistemas operativos
Soporte multihilos	En los tres casos se soporta la programación multihilos		
Creación de interfaces gráficas de usuarios	Sencillas de crear Existen bibliotecas gráficas especializadas	Sencillas de crear, pero difíciles de implementar	Difíciles de crear e implementar

- Los dispositivos móviles (*Smartphone* y PDA) tienen poca capacidad de almacenamiento, y cuentan con una capacidad limitada de procesamiento si se compara con la de una *Laptop* o una PC convencional.
- Portabilidad, es decir, debe ser sencillo pasar la aplicación de un dispositivo a otro.
- Soporte para conexiones de tipo TCP y UDP (*sockets*) [35].
- Soporte para el desarrollo de aplicaciones multihilos.
- Finalmente, que el lenguaje tenga facilidad para el desarrollo de interfaces gráficas.

La tabla 3.1, muestra una comparativa de los lenguajes *C* [8], *C++*, *Java* [33] y *Python* [68].

Elegimos *Python*, no solo porque cumple con todos los requisitos mencionados anteriormente, sino también por la facilidad que da al programador para el desarrollo de aplicaciones, y porque la portabilidad es mucho mayor que en los otros lenguajes, puesto que *Python* es un lenguaje interpretado (i.e., no se tiene que compilar para su ejecución). *Python* además es sencillo de instalar, solo es necesario descargar el intérprete para el sistema operativo que se desee (los intérpretes pueden ser descargados del sitio oficial de *Python* [68]). Finalmente, otra de las ventajas de *Python*, es

que las versiones que proporciona para dispositivos móviles, cuentan con la mayoría de librerías que se presentan en el *Python* estándar. Por lo tanto, el programador puede reutilizar gran parte de un código, por ejemplo, una aplicación de *Python* para Windows puede ser compatible en *Python* para *Linux*.

3.3. Diseño y desarrollo de interfaces

Dado que SEGEUS es flexible para su ejecución en distintos dispositivos, el diseño y desarrollo de las GUI fue una parte importante del trabajo.

Lo primero que hicimos, fue evaluar las librerías gráficas para *Python* estándar. Existen tres librerías que trabajan sobre *Python* y que pueden ser instaladas en diversos sistemas operativos: *WxPython*, *Python Imaging Library (PIL)* [36] y *Tkinter* [37].

Básicamente, las tres librerías ofrecen las mismas características, sin embargo, *Tkinter* tiene la ventaja de ser más sencilla de instalar puesto que ya viene incluido en los archivos de instalación de *Python* para su distribución en *Windows*, i.e., se instala junto con *Python*, y para la distribución en *Linux*, *Tkinter* es sencillo de instalar ya que solo es necesario instalar la dependencia `python-tk` con el siguiente comando [38]:

```
sudo aptitude install python-tk
```

Tkinter permite la creación de ventanas, menús, botones, listas, entre otras herramientas gráficas. Existen dos maneras de utilizar *Tkinter* en *Python* [38]:

```
import Tkinter
```

O bien:

```
from Tkinter import *
```

Una aplicación básica de *Tkinter* puede ser [38]:

```
from Tkinter import *
root = Tk()
w = Label(root, text="Hola Mundo!")
w.pack()
root.mainloop()
```

El resultado de ejecutar este código se muestra en la Figura 3.1.

Como se puede notar, *Tkinter* es sencillo de utilizar y puede brindar ambientes amigables, además, la aplicación podría ejecutarse en los sistemas *Linux* y *Windows* sin necesidad de modificar el código. El único problema que se presenta con *Tkinter*, es que no proporciona un módulo que se pueda utilizar en la versión de *Python* para los sistemas *Symbian*.

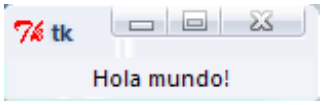

Figura 3.1: Ejemplo básico de una aplicación con el módulo *Tkinter*

Sin embargo, la versión de *Python* para *Symbian S60* (*PyS60*) [39], cuenta con un conjunto de módulos, los cuales son archivos que contienen funciones relacionadas y datos agrupados. En el caso de *PyS60*, cuenta con módulos como *messaging*, que proporciona funciones para hacer uso de SMS's, la cámara, y otros elementos [40].

De entre los módulos con que cuenta *PyS60*, se encuentra *appuifw*, que provee los elementos para la creación de GUI [40]. De igual manera que con *Tkinter*, el módulo *appuifw* puede importarse en un código de la siguiente manera:

```
import appuifw
```

O bien:

```
from appuifw import *
```

Si queremos realizar una aplicación similar a la mostrada en la Figura 3.1, en *PyS60*, el código sería el siguiente [41]:

```
import appuifw
appuifw.note(u"Hola Mundo!")
```

El resultado de ejecutar este código se muestra en la Figura 3.2.

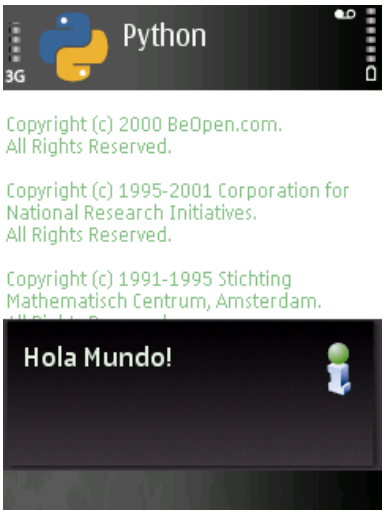

Figura 3.2: Ejemplo básico de una aplicación con el módulo *appuifw* (*Pys60*)

De acuerdo con lo anterior, podemos hacer una aplicación que sea compatible para diferentes sistemas operativos. Lo único que debemos hacer es crear funciones independientes de los elementos gráficos, de esta manera, la funcionalidad de la aplicación será la misma (si las capacidades de los dispositivos lo permiten), y la interfaz será adaptable al sistema operativo en que se ejecute la aplicación.

Un ejemplo de una aplicación adaptable al sistema operativo (para sistemas *Symbian*, *Windows* y *Linux*) es el siguiente:

```
import sys

if str(sys.platform) == ‘‘symbian_s6’’:
        import appuifw
        appuifw.note(u‘‘Hola Mundo!’’)
else:
        from Tkinter import *
        root = Tk()
        w = Label(root, text=‘‘Hola Mundo!’’)
        w.pack()
        root.mainloop()
```

El resultado de ejecutar del código sería la Figura 3.1 en caso de ser ejecutado en un sistema *Windows*, la Figura 3.2 si se ejecuta en un sistema *Symbian*, y la Figura 3.3 si se ejecuta en un sistema *Linux* (*Ubuntu*).

Figura 3.3: Ejecución de un ejemplo básico de *Tkinter* para el sistema operativo *Linux* (*Ubuntu*)

3.4. Protocolo de Inicio de Sesión (SIP)

Una vez que se determinó la manera en que se iban a crear las interfaces de usuario en los diferentes sistemas operativos, fue necesario revisar las herramientas que en conjunto, dan funcionamiento al proyecto.

Una de las herramientas más importantes para este proyecto, es el Protocolo de Inicio de Sesión (*Session Initiation Protocol - SIP*).

SIP es un protocolo que trabaja dentro de la capa de aplicación en la red, se utiliza para iniciar, modificar y terminar sesiones colaborativas y de comunicación sobre IP, estas sesiones son conocidas como sesiones multimedia, por ejemplo, llamadas de VoIP [42], juegos en línea, reproducción de música remota, etc. [10].

El protocolo SIP nace a mediados de los 90's, con la fusión de dos proyectos: a) Protocolo de Inicio de Sesión, por Eve Schooler y Mark Handley y b) Protocolo Simple de Invitación a Conferencias (SCIP) por Henning Schulzrinne. Aprobado con el RFC 2543 por la Internet Engineering Task Force (IETF) en marzo de 1999 [14]. Finalmente, el estándar se liberó con el RFC 3261 [1] en el 2002.

SIP nació originalmente, con la idea de hacer de la telefonía un servicio más sobre Internet, sin embargo, SIP es más que un protocolo que proporciona el servicio de llamadas sobre IP. Existen diversos usos que se pueden dar al protocolo, ya que éste puede interactuar con otros protocolos como: Protocolo de Descripción de Sesión (SDP) [43] que sirve para describir los parámetros de las sesiones multimedia; Protocolo de Transporte en Tiempo-Real (RTP) [44] que se utiliza para el envío de audio y video por Internet, Protocolo de Control de Transporte en Tiempo-Real (RTCP) [45] que provee información para el control del flujo de datos RTP, etc. En sus últimas versiones, ya cuenta con un módulo de seguridad y soporta múltiples usuarios en una misma sesión.

3.4.1. Funcionalidad de SIP

SIP puede trabajar bajo una arquitectura cliente-servidor (C/S) o punto a punto (P2P).

Las redes SIP consisten en dos componentes básicos: agente de usuario SIP y servidor de red SIP.

SIP maneja algunos conceptos que son necesarios presentar a continuación para comprender de mejor manera su funcionamento [14]:

- Agente de usuario: (UA) es un dispositivo que puede originar y recibir llamadas SIP.

- Agente de usuario servidor: en una aplicación (C/S), un UA se comporta como un cliente si es él quien realiza las peticiones, y se comporta como servidor cuando es él quien recibe las peticiones.

- Back to Back UA (B2BUA): cuando una entidad SIP actúa como cliente y servidor se denomina B2BUA. Esta entidad genera peticiones que determinan como serán respondidas las peticiones entrantes.

- Servidor Proxy: es un componente clave en SIP, se comporta como un servidor Proxy Web. Esta es la entidad que recibe todas las peticiones salientes de un UA.

- Secretario (REGISTRAR): es un repositorio de información para la localización de UAs. Asocia direcciones IP a direcciones SIP. Un usuario puede tener muchas direcciones IP almacenadas.

- Servidor de redireccionamiento: responden a una petición SIP con una dirección a la cual el mensaje SIP debe ser redireccionado. Este mapea una dirección destino o varias en el mensaje.

Toda la comunicación en SIP se maneja por medio del intercambio de **mensajes**. Existen dos tipos de mensajes: **peticiones y respuestas**, los cuales comparten un cierto formato que consiste en una **línea de inicio**, uno o más **campos de encabezado**, una **línea vacía** que indica el fin de los encabezados, y un cuerpo de mensaje opcional (Figura 3.4) [14].

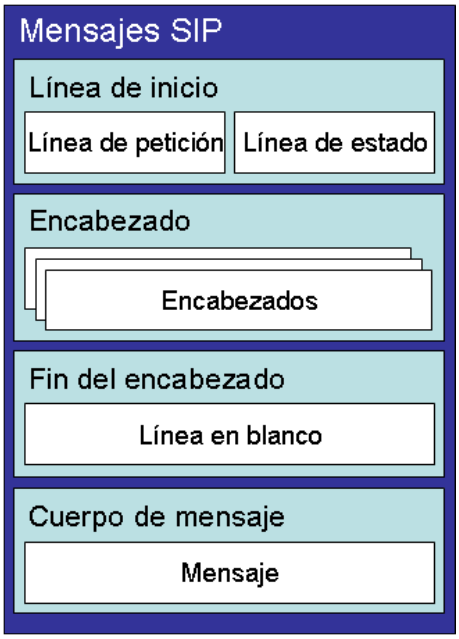

Figura 3.4: Mensajes SIP

La línea de inicio puede ser de estado o petición, en caso de ser del segundo tipo, indica el tipo de petición. En los mensajes de respuesta, la línea de estatus indica si la petición se completó o aún no [14].

3.4.2. Sesiones SIP

Para establecer una sesión en SIP, es necesario que se realice una petición, i.e., que la línea de inicio sea de tipo petición. Dicha línea tiene tres campos separados por espacios, los cuales llevan el siguiente orden: *Método*, *Petición-URI* (Identificador de Recurso Uniforme, para identificar participantes y recursos en sesiones de comunicación), y la *Versión* de SIP que se está utilizando [14]. Los métodos varían

dependiendo de la versión, sin embargo, los métodos básicos de SIP se mantienen en todas las versiones, algunos de estos métodos son [14]:

- *REGISTER*.- Da al secretario información sobre la localidad del UA y si se encuentra disponible para peticiones SIP. Cuando un UA cambia su localidad, se envía otro mensaje REGISTER para actualizar la base de datos.
- *INVITE*.- Se utiliza para iniciar una comunicación entre dos UA. Aunque también se puede utilizar para una llamada multiusuario.
- *ACK*.- Indica que se ha recibido la respuesta final.
- *CANCEL*.- Termina peticiones pendientes.
- *OPTIONS*.- Se utiliza para preguntarle a un Proxy sus capacidades.
- *BYE*.- Indica el término de una sesión.
- *SUSCRIBE* y *NOTIFY* (de la extensión RFC 3265 [46]).- Se utilizan en conjunto para eventos basados en notificaciones. Por ejemplo, un usuario se suscribe a eventos tales como información de presencia de otro usuario [29]. El usuario se suscribe al servidor de presencia, cuando el otro usuario se muestra disponible, el servidor envía un *NOTIFY* [14].

La *petición-URI* (*Request-URI*) se utiliza para indicar el usuario o servicio al cual se direcciona la petición. Cuando se recibe una petición (excepto una de tipo ACK) se dispara una respuesta que se envía como un mensaje de respuesta SIP. Los mensajes de respuesta SIP se distinguen porque tienen una línea de estado (*Status-line*) en su línea de inicio. Esta línea de estado consta de tres campos: Versión de SIP, código de estado y *Reason-phrase* separados por un espacio [14].

El Código de estado (*Status-code*) es un código de tres dígitos (100 - 699) que representa el resultado del proceso de la petición. El primer dígito indica la clase de respuesta, por ejemplo, 2XX se utiliza para indicar "éxito", es decir, que la petición fue recibida, entendida y procesada con éxito.

Finalmente, para que una persona pueda entender el código de estado, se creó el campo *Reason-phrase*, que contiene una descripción corta del código de estado, la cual es legible para cualquier persona.

El conjunto de intercambios de mensajes realizados por SIP (que inician con una petición y todas las respuestas relacionadas a dicha petición) se consideran **Transacciones SIP**. Los componentes SIP que asocian peticiones y respuestas a transacciones son llamados **componentes de estado**. Estos componentes obtienen identificadores de transacción únicos de los mensajes y son capaces de actualizar la información de estado de la transacción [14]. Finalmente, un conjunto secuencial de transacciones se considera como un **Diálogo**. De esta manera es como se pueden establecer sesiones de comunicación con SIP. Un ejemplo de una sesión de comunicación se observa en la Figura 3.5.

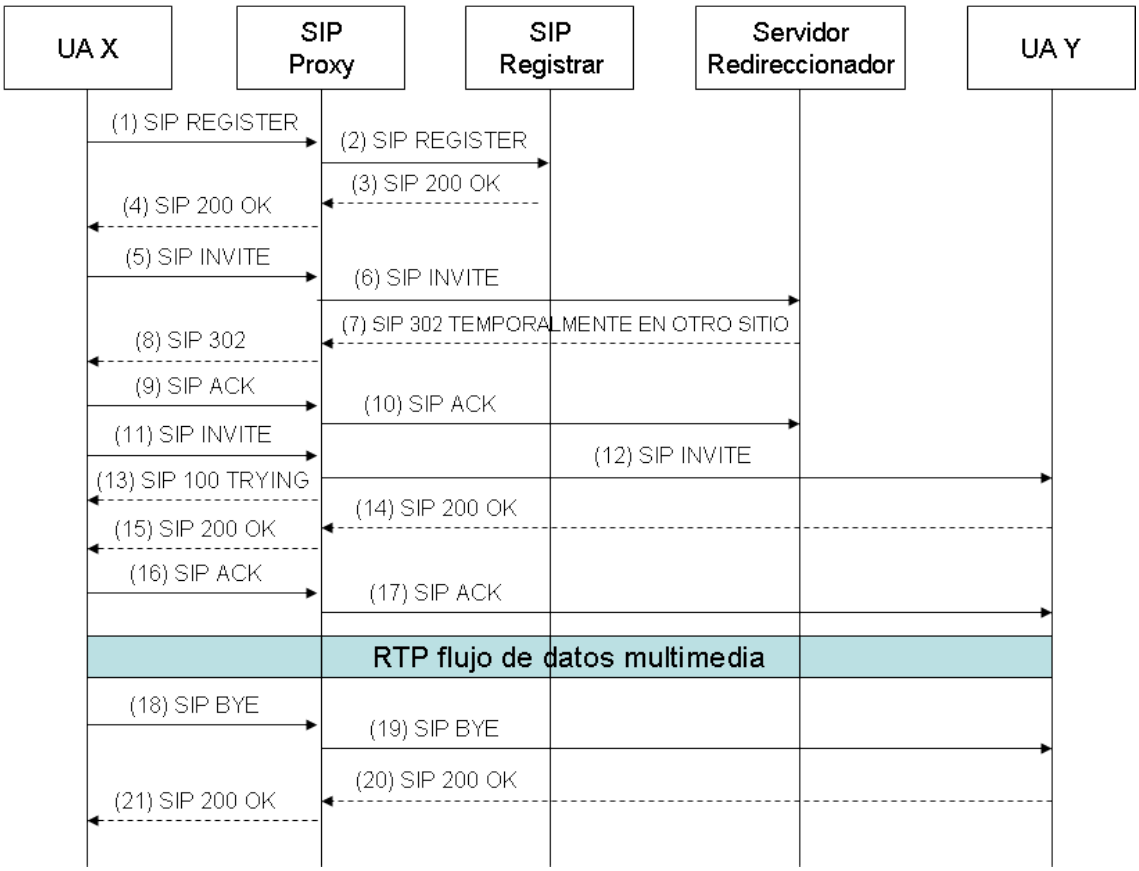

Figura 3.5: Ejemplo de sesión SIP (llamada)

3.5. Aplicaciones y herramientas para el uso de SIP-PJSUA

Existen diversos tipos de herramientas que dan soporte a los desarrolladores para utilizar el protocolo SIP. Cada herramienta da soporte para algunos lenguajes de programación en específico. Algunas de las herramientas que trabajan con SIP se mencionan a continuación.

SIPp es una herramienta de código abierto para generar tráfico con el protocolo SIP [47]. Provee escenarios para establecer y liberar múltiples llamadas con los métodos INVITE y BYE de SIP. Esta herramienta permite desplegar estadísticas sobre las pruebas en ejecución (llamadas, mensajes, retardos). Además soporta IPv6, transmisiones UDP, manejo de errores (expiración de llamadas, defensa de protocolo), etc. [48].

SIPp fue desarrollado por *Hewlett Packard* (HP) y puede ser descargado del sitio `http://sipp.sourceforge.net/` [69], en donde se cuenta con una guía de instalación, uso y pruebas de todas las herramientas que proporciona.

Ekiga es otra herramienta de software libre que utiliza SIP (es conocida formalmente como *GnomeMeeting*). Esta herramienta es utilizada como una aplicación de VoIP y videoconferencia para usuarios GNOME. Tiene soporte para diferentes *codecs* de audio y video, y es compatible con otras herramientas de SIP e inclusive con *Microsoft NetMeeting* [70].

PJSUA es una *API* que permite explotar las funciones de SIP y da soporte para distintas plataformas (*Windows*, *Linux*, *Symbian*, etc.) y para los lenguajes de programación *C*, *C++*, *Python* y *ActiveX* (para el uso de *Visual Basic* o *C# .Net*). Se elige *PJSUA* dado que esta herramienta está en constante actualización y porque tiene soporte para mayor número de sistemas operativos. *PJSUA* puede descargarse de su sitio oficial [71]. *PJSUA* cuenta con una arquitectura muy robusta y su uso no es complicado. Su arquitectura se muestra en la Figura 3.6.

Como podemos observar en la Figura 3.6, la arquitectura de PJSUA cuenta con los siguientes módulos [49]:

PJLIB es una biblioteca de código abierto, desarrollada en lenguaje *C* bajo *Licencia Pública General* (*GPL*), que se utiliza para crear aplicaciones escalables. *PJLIB* es la biblioteca básica de *PJSUA*, su principal ventaja es que se desarrolló con el fin de presentar portabilidad extrema (que va desde procesadores de 16 bits, hasta multiprocesadores, o de sistemas embebidos a no embebidos).

PJLIB-UTIL es la documentación auxiliar que viene junto con *PJLIB*, en esta documentación podemos encontrar información acerca de los algoritmos de cifrado que utiliza el *API* (*Application Programming Interface*), información sobre la configuración y los mensajes de error que pueden desplegarse al momento de ejecutar el *API*, la manipulación de cadenas, los protocolos que soporta y finalmente el formato que presentan los archivos del *API*.

PJNAT es una biblioteca que proporciona funciones de traducción de direcciones

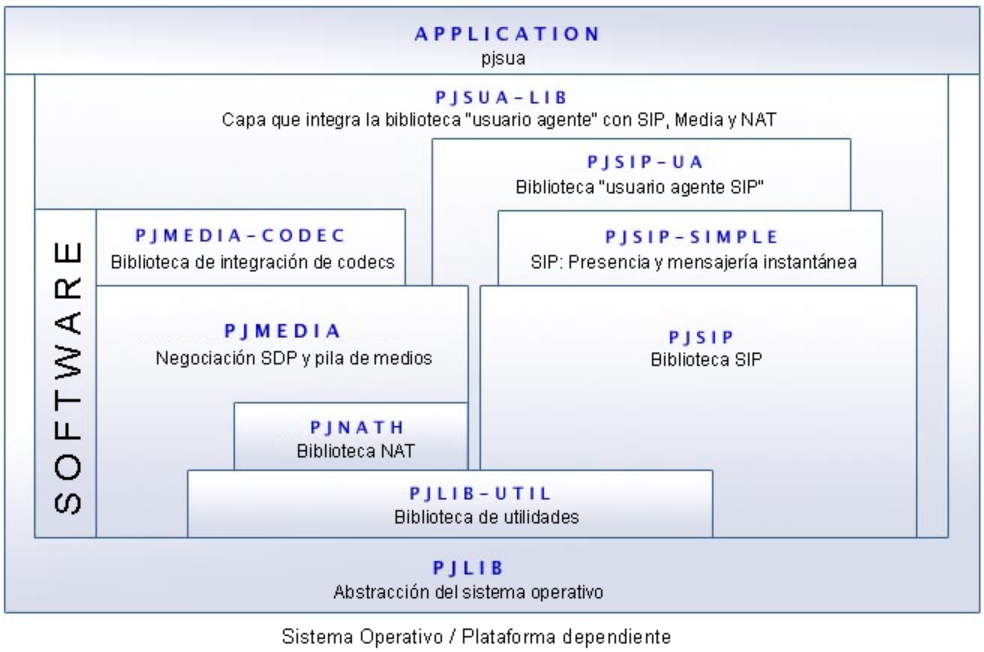

Figura 3.6: Arquitectura de PJSUA

de red (*Network Address Translation - NAT*) para utilizar estándares basados en protocolos tales como *STUN (Simple Transversal of UDP over NATS)*, que permite a los clientes *NAT* conocer su dirección IP pública, el tipo de *NAT* en el que se encuentran y el puerto de internet asociado al puerto local a través de *NAT*; *TURN (Traversal Using Relay NAT)* que permite a los elementos que se encuentren detrás de un *firewall* recibir datos sobre *TCP* [50] y *UDP* [51], y *ICE (Interactive Connectivity Establishment)* que se utiliza como metodología de *NAT* para dar soporte a *SIP* [71].

PJMEDIA da soporte para las herramientas multimedia que puedan existir en las diferentes plataformas donde se ha de instalar *PJSUA*, los módulos que implementa son: el reloj, *codecs* para multimedia, sesiones media (*RTCP*, *RTP* y *SDP*), transporte media, puertos media (puerto bidireccional, puente de conferencia, *API* para la acústica, búfer de memoria, canal de medios, etc.), algoritmos de manipulación de audio, formatos de archivo (*WAVE*), *API* para los dispositivos de audio [71].

PJMEDIA-CODEC es una biblioteca estática que posee varios *codecs* para la implementación del *API*.

PJSIP, es una pequeña implementación de código abierto de la pila del protocolo *SIP*, que presenta un alto rendimiento y una gran flexibilidad. Esto vuelve a *PJSUA* un *API* de alto nivel, que permite crear aplicaciones *agentes usuarios SIP*, y que se caracteriza por permitir: registrar múltiples clientes, establecer sesiones de alto nivel (llamadas), crear una lista de contactos (permitiendo enviar mensajes y conocer el estado de los contactos) y finalmente utilizar de manera completa los elementos multimedia.

PJSIP-SIMPLE es un *framework* que permite el control de eventos y el servicio de presencia.

PJSIP-UA es la biblioteca de agente usuario (*user agent*), la cual provee las funciones de: invitación para iniciar una sesión multimedia (*INVITE*), registro de clientes y llamadas VoIP sobre SIP [71].

PJSUA-LIB es la manera en la que se almacena la biblioteca de *PJSUA* dentro de algún dispositivo, es lo que la vuelve portable, y permite mandarla a llamar con algún comando en específico dependiendo la plataforma donde se vaya a utilizar [49].

Finalmente *APPLICATION* es el conjunto de todos los elementos mencionados, que permite realizar las siguientes funciones: realizar llamada, realizar una llamada múltiple, contestar llamadas, colgar, silenciar una llamada, reestablecer una llamada, enviar actualización, seleccionar una llamada de entre varias que se tengan activas, agregar usuarios a la lista de contactos, borrar usuarios, enviar mensajes de texto, etc [52].

En la siguiente sección mostraremos como compilar y portar el código a los sistemas embebidos y no embebidos. Como se mencionó anteriormente, el lenguaje que se eligió para desarrollar el servicio es *Python*, y *PJSUA* encaja perfectamente dado que soporta el desarrollo en el mismo. La mayor ventaja será que el *script* pueda trabajar en diferentes sistemas operativos sin hacer modificaciones, o en su defecto, modificaciones mínimas.

Al iniciar el desarrollo de este proyecto, se comenzó utilizando la versión 1.0 de la biblioteca, sin embargo, dado que ésta se actualiza constantemente, las pruebas se concluyeron con la versión 1.2.

3.5.1. Instalación de PJSUA en Linux

La manera de instalar la biblioteca no varía entre las diferentes versiones que utilizamos (1.0 hasta la 1.2), pero la operatividad de la última versión implementa nuevas funciones y corrige errores (*bugs*) a medida que se ha ido actualizando.

La compilación de la biblioteca depende del sistema operativo en el cual se vaya a instalar. Compilar *PJSUA* en *Linux* es sencillo, lo primero que requerimos es descargar la biblioteca, posteriormente descomprimir los archivos en algún directorio, para efectos de lectura se llamará a la carpeta `PJDIR`. Dentro de ese directorio se encontrará la carpeta `pjproject`. Se requieren las siguientes herramientas [71]:

- GNU *make*
- GNU *binutils*
- GNU *gcc*
- GNU *autoconf*
- *Python* 2.x[1].

[1]se utilizó la versión 2.6 de *Python*, aunque también se realizaron pruebas para la versión 2.5 y los resultados fueron satisfactorios

```
$ cd PJDIR/pjproject
$ sudo ./configure
```

Una vez completa la configuración, procedemos con lo siguiente:

```
$ cd PJDIR/pjproject
$ sudo make dep
$ make
$ make clean
```

Hasta este punto se cuenta con todas las bibliotecas y ejemplos compilados. Hay dos módulos de *Python* con que cuenta la biblioteca. La diferencia es que uno maneja todas las funciones de *PJSUA* por completo (llamadas, mensajes, presencia, etc.), y el otro módulo las manipula con *scripts* distintos.

Para instalar el primer módulo se hace lo siguiente:

```
$ cd PJDIR/pjproject/pjsip-apps/src/py_pjsua
$ python setup.py build
```

Esto guarda el módulo de *Python* en el directorio `build` que se encuentra en el directorio actual. Otra manera de hacerlo, es utilizar el siguiente comando:

```
$ python setup.py install
```

Que instala el módulo en el directorio `site_packages` de *Python*.

El ejemplo con que cuenta este módulo es el *script* `pjsua_app.py`. La ejecución de este *script* se muestra en la Figura 3.7.

El otro módulo de *Python* se encuentra en: `PJDIR/pjproject/pjsip-apps/src/python`. De igual manera que con el primer módulo, se ejecuta el comando:

```
$ python setup.py install
```

Para efectos del proyecto, el módulo que se genera en el directorio `site_packages` de *Python*, lo modificamos para que el puerto siempre fuese el 5060, ya que la instalación en *Windows Mobile* y *Symbian* otorgan ese puerto por defecto para el uso de SIP.

La instalación de todos los módulos la realizamos en la distribución *Ubuntu* 8.04 y 8.10. y la ejecución de las pruebas fue satisfactoria.

3.5.2. Instalación de PJSUA en Windows

De igual manera que en *Linux*, descargamos el proyecto *PJ* de la página. Este proyecto cuenta con archivos para la herramienta *Microsoft Visual Studio*. Utilizamos la versión *Visual Studio 2005 Pro* en este caso.

Las herramientas que se necesitan para la instalación en Windows son las siguientes:

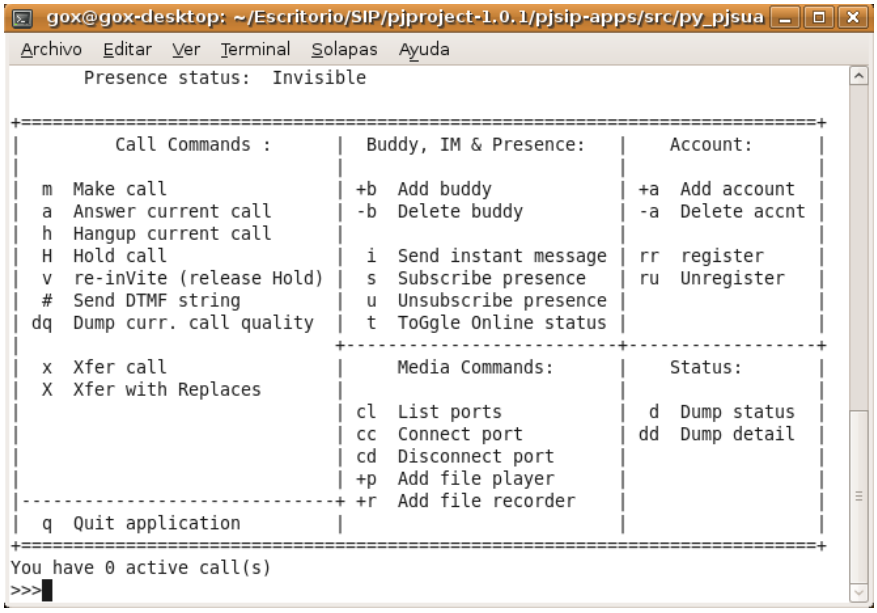

Figura 3.7: Menú principal de la aplicación PJSUA (el mismo para Windows y Linux)

- *Platform Kit* de Desarrollo de Software (SDK) para *Windows XP* con *Service Pack 2*
- *DirectX* SDK 2007
- *Python* 2.x[2].
- *Visual Studio 2005 Pro*

Procedemos a agregar las rutas necesarias de las bibliotecas *Platform SDK*, *DirectX SDK* y *Python* en *Visual Studio 2005*. Para proceder con la instalación de las aplicaciones y los ejemplos hacemos lo siguiente [30]:

- Abrir el archivo `pjproject-vs8.sln` que se encuentra en `PJDIR\pjproject` desde *Visual Studio 2005*.
- Seleccionar `pjsua` como proyecto activo.
- Elegir la opción *Debug* o *Release build* en Visual Studio.
- Construir el proyecto.

Hasta este punto contamos ya con la aplicación *pjsua* en el directorio `bin` de la carpeta actual y las bibliotecas en el directorio `lib` de cada proyecto.

Para compilar los ejemplos hacemos lo siguiente:

[2]De igual manera que en *Linux*, se utilizó la versión 2.6 de *Python*

- En el mismo proyecto, se selecciona *samples* como proyecto activo.
- Construir el proyecto.

De esta manera las aplicaciones ejemplo se encontrarán en el directorio `PJDIR\pjproject\pjsip-apps\bin\samples` y las bibliotecas en el directorio `lib` de cada proyecto.

Como mencionamos anteriormente, se cuentan con dos módulos de *Python*. Para instalar el primer módulo abrimos el proyecto `pjsip-apps.dsw` ubicado en `PJDIR\pjproject\pjsip-apps\build`.

- Seleccionamos `py_pjsua` como proyecto activo.
- Compilamos el proyecto.

Y esto guardará el proyecto en el directorio `PJDIR\pjproject\pjsip-apps\lib`

Para instalar el segundo módulo, descargamos la aplicación *pjsua-1.2.win32-Py26* de un foro. Sin embargo, el foro ya no se encuentra en línea desde Junio del 2009. Aunque también descargamos las versiones para *Python* 2.4 y 2.5 de la misma herramienta que funciona para la versión 1.2 de *PJSUA*. Estos archivos de instalación van incluidos en la carpeta de instalación que se entrega junto con esta tesis.

Al instalar estos paquetes, el módulo de *pjsua* se almacena en el directorio `lib\site_package` de *Python* (dependiendo de la versión), al igual que en *Linux*, modificamos el módulo de *pjsua* para que el puerto de SIP fuese el 5060.

3.5.3. Instalación de PJSUA en Windows Mobile

Para la instalación requerimos nuevamente de la herramienta *Microsoft Visual Studio 2005 Pro*. Se utilizó una PDA *HP iPAQ* 216 que cuenta con un sistema operativo *Windows Mobile 5.0* [30].

Requerimos las siguientes herramientas para compilar los programas de *Windows Mobile 5.0*:

- Una computadora con Windows.
- Microsoft Visual Studio 2005 Pro [53].
- Windows Mobile 5.0 Pocket PC SDK.

Para compilar el programa *WinCE* que viene como ejemplo realizamos lo siguiente:

- Abrir el archivo `PJDIR\pjproject\pjsip-apps\build\wince-evc4\wince demos.vcw` con Visual Studio 2005 (este archivo se convertirá de manera automática).
- Seleccionar el proyecto `pjsua\_wince` como proyecto activo.

- Seleccionar el SDK Windows Mobile 5.0 Pocket PC SDK [53].
- Seleccionar *Debug* o *Release build*.
- Seleccionar la configuración *device* o *emulator*.
- Construir el proyecto.

Según se haya elegido *device* o *emulator*, la aplicación se almacena en `PJDIR\pjproject\pjsip-apps\src\wince`. Posteriormente se debe enviar el ejecutable al PDA, para hacer modificaciones es necesario realizarlas directamente en el código fuente y volver a construir el proyecto.

3.5.4. Instalación de PJSUA en Symbian

Para la instalación de la biblioteca en *Symbian*, requerimos las siguientes herramientas [71]:

- Una computadora con *Windows XP*
- *Symbian S60 3rd Edition*
- *Carbide C++* 2.0
- Dispositivo *Nokia* 3ra edición
- *Active Perl 5*

Se procede a crear un archivo `config_site.h` que contenga lo siguiente:

```
#include <pj/config_site_sample.h>
```

Este archivo se guarda en el directorio `PJDIR\pjproject\pjlib\include\pj`.

Ejecutamos *Carbide* y seleccionamos un *Workspace*. Posteriormente seguimos los siguientes pasos:

- Ir a *File* `->` *Import* y en el cuadro de diálogo que aparece, seleccionar *Symbian OS* y su ramificación *Symbian OS Bld.inf file*.
- Luego en la siguiente ventana, buscar el archivo `bld.inf` que se encuentre en el directorio `PJDIR\pjproject\build.symbian`
- Posteriormente seleccionar los componentes que correspondan al dispositivo, en este caso fue *S60_3rd_FR* y las ramificaciones *Phone Debug* y *Phone Release* (GCCE en ambos casos)
- Seleccionar todos los archivos *MMP* en la siguiente ventana

- Finalmente terminar de importar el proyecto.

Posteriormente procedemos a compilarlo y crear el archivo `.sis` que será instalado en el dispositivo. Para esto debemos realizar lo siguiente [71]:

- Ir a la pestaña *Project* y en la opción *Active Build Configuration* seleccionar *Phone Release*.

- En la ventana del proyecto, dar clic derecho en `bld.inf` y seleccionar la opción *Build Target Only*.

Una vez hecho esto procedemos a crear el archivo `.sis`:

- Dar clic derecho en *pjproject* e ir a las propiedades.

- Revisar que en *Configuration* aparezca la opción de *Phone Debug* (GCCE) y en *PKG File* aparezca el archivo `symbian_ua_udeb.pkg` al final de la ruta.

- Volver a dar clic derecho en `blf.inf` y refrescar.

- Al hacer esto aparecerá en la ventana del proyecto la ramificación `symbian_ua.pkg`.

- Ahora solo resta dar clic derecho en `bld.inf` y seleccionar de nuevo *Build Target Only*.

Ahora estarán los archivos `Symbian_ua.sis` y `Symbian_ua.sisx` en el directorio `PJDIR\pjproject\buid.symbian`, lo único que procede es copiar los archivos al dispositivo y ejecutar el archivo `.sisx` [71].

El menú principal de la aplicación para *Symbian* se muestra en la Figura 3.8.

Para nuestro proyecto probamos únicamente los servicios de presencia, mensajes de texto y llamadas de VoIP. La ejecución de los servicios fue satisfactoria y mostró compatibilidad en *Windows* y *Linux*. Sin embargo para *Symbian* y *Windows Mobile*, el servicio de mensajes de texto no existe, únicamente se reciben los mensajes pero no está creada la función para enviarlos.

Después de realizar las pruebas anteriores, optamos por utilizar únicamente el servicio de llamadas VoIP que proporciona *PJSUA* e incorporarlo a la aplicación, y los otros servicios fueron creados con *Python*. En el caso de *Symbian* y *Windows Mobile* creamos un **intermediario** que comunica las aplicaciones de *PJSUA* y SEGEUS para poder utilizar el servicio de llamadas de VoIP que el *API* proporciona.

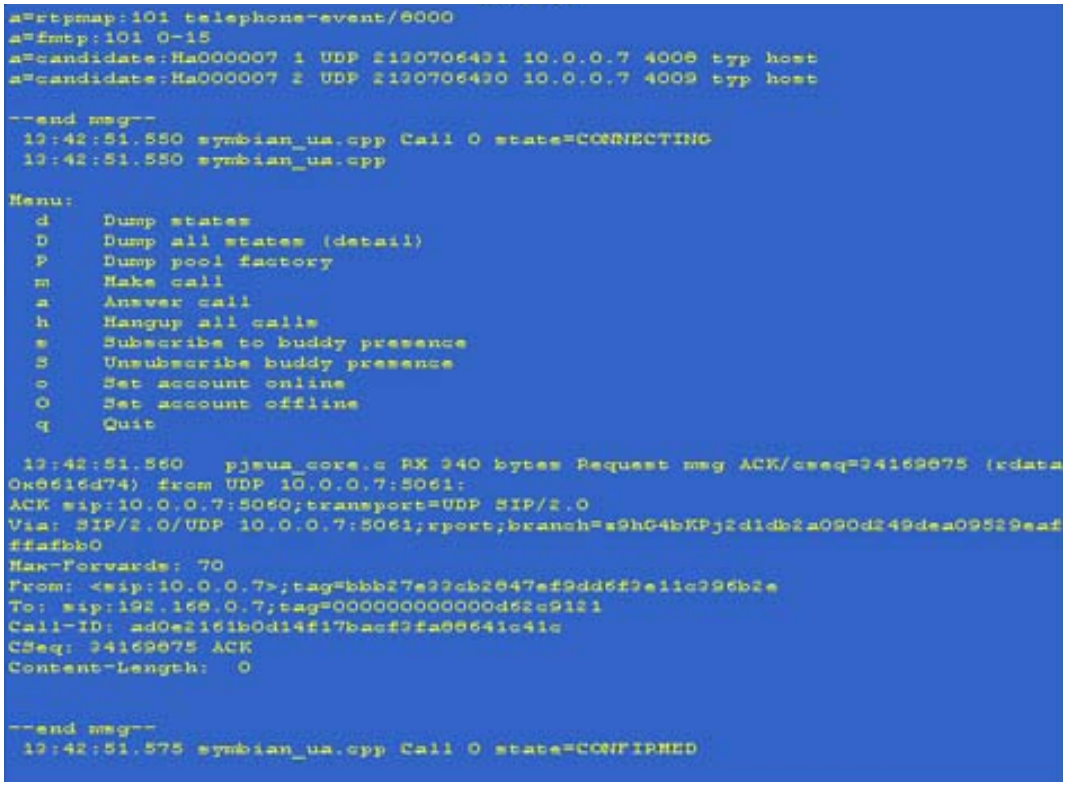

Figura 3.8: Menú principal de la aplicación PJSUA (Symbian)

3.6. Gestión de sesiones de usuarios

Para realizar la gestión de las sesiones de usuarios, es necesario contar con un servidor que proporcione a los usuarios los servicios descritos anteriormente. Este servidor debe controlar los eventos generados por los usuarios, que en ocasiones debe notificar a otros usuarios, y en otros casos únicamente actualizar datos de algún usuario.

Para lograr lo anterior, el servidor cuenta con una base de datos que se encarga de controlar las sesiones y dispositivos de todos los usuarios (ver sección 4.1).

El servidor fue programado en *Python*, y se encarga de comunicar la aplicación creada en *Python* con la aplicación creada en *C* tanto para *Symbian* como para *Windows Mobile*, y la base de datos se creó en *MySQL* dada la facilidad de uso y dado que *Python* cuenta con un módulo que facilita la conexión con la BD.

Finalmente, la comunicación entre usuarios y entre usuario-servidor fue creada con sockets puros, esto da la posibilidad de escalabilidad para la creación de otros servicios, excepto para el caso de las llamadas VoIP, ya que *PJSUA* se encarga de la comunicación.

Capítulo 4

Servicio de gestión de sesiones de usuarios en ambientes móviles y heterogéneos (SEGEUS)

De acuerdo con el análisis mostrado en el capítulo 3, optamos por utilizar *Python* (en combinación con *C* para el caso de *Windows Mobile* y *Symbian*) para la programación, *Tkinter* (*appuifw* en el caso de *Symbian*) para la creación de la interfaz gráfica y *PJSUA* para el uso del servicio de llamadas VoIP de SIP.

En este capítulo analizaremos el diagrama de componentes de SEGEUS, explicando de manera detallada cada componente, además presentaremos la arquitectura del servicio, detallando cada uno de sus módulos.

4.1. Componentes

SEGEUS es un servicio que presenta como principal ventaja, el que los usuarios puedan tener una sesión activa en más de un dispositivo simultáneamente [54], además, de poder dar preferencias al tipo de tráfico que deseen recibir en sus dispositivos.

En la Figura 4.1 podemos apreciar el diagrama de componentes de SEGEUS, estos se leen de abajo hacia arriba, es decir: *Usuario*, *Sesión virtual*, *Dispositivo*, *Sesión real*, *Sesión de comunicación*, *Red* y finalmente el *Servidor*, en las siguientes subsecciones se hablará a detalle de cada uno de los componentes, mostrando su funcionalidad dentro de SEGEUS.

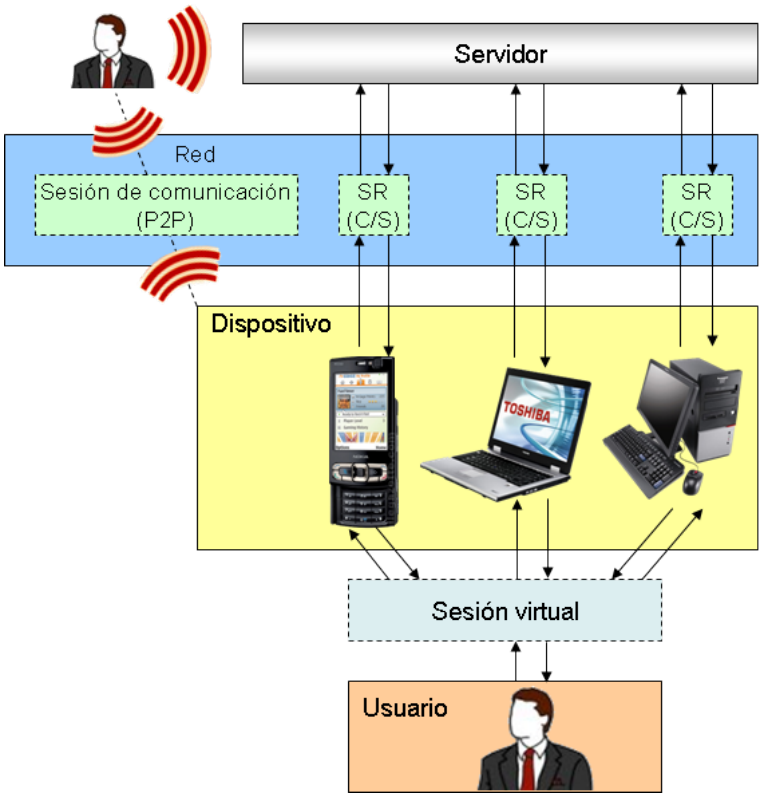

Figura 4.1: Diagrama de componentes SEGEUS

4.1.1. Usuario

El componente **Usuario** (también podemos denominarlo componente **Aplicación**) se refiere a la aplicación, i.e., es la interfaz que se presenta a los usuarios en cada uno de sus dispositivos y que le brinda de manera amigable los servicios que proporciona SEGEUS.

Para iniciar la aplicación es necesario que el usuario ingrese su identificador (*ID*) y contraseña, así como el dispositivo en el cual está iniciando sesión. Una vez que el usuario inicia sesión, la aplicación se "adapta" al dispositivo en el cual la esté usando, creamos tres aplicaciones distintas que pueden ser descargadas del servidor directamente, una que funciona en los ambientes *Windows* y *Linux*, otra que funciona en cualquier dispositivo con sistema operativo *Symbian S60* y otra que funciona en los dispositivos con *Windows Mobile 5.0 y 6.0*.

La aplicación muestra las mismas funciones para todos los dispositivos (siempre y cuando éstos la soporten), ésta trabaja por medio de eventos que pueden ser generados por los usuarios o por el servidor. Dichos eventos pueden ser:

- Usuario: cambio de alias o estado, envío de archivos, inicio de una conversación, envío de petición de una llamada VoIP, cambio de preferencias y término de

sesión.

- Servidor: inicio de sesión de usuarios (por primera vez), inicio de sesión en más de un dispositivo, cambio de estado y alias de usuarios, y término de sesión de usuarios.

- Entre usuarios: peticiones entrantes de llamadas o archivos e inicio de conversaciones.

4.1.2. Sesión virtual

Este componente es el encargado de dar la apariencia al usuario de que su sesión activa es la misma en todos sus dispositivos, funciona en colaboración con el componente **Usuario**, puesto que la aplicación muestra los mismos datos para todos los dispositivos que tenga el usuario.

El funcionamiento de este componente es el siguiente: el usuario toma cualquiera de sus dispositivos con SEGEUS e inicia sesión, cuando esto sucede, se establece una comunicación entre el dispositivo y el servidor. Sin embargo, para el usuario, esta comunicación dispositivo-servidor es transparente, i.e., el usuario ve la misma información en todos sus dispositivos, como si fuese una sola comunicación entre el servidor y todos sus dispositivos.

Una **Sesión Virtual** es creada para permitir que los cambios que realice el usuario en un dispositivo cualquiera, se reflejen en todos sus dispositivos. Por ejemplo, supongamos un *UsuarioX* que cambia de alias en su PDA, automáticamente el nuevo alias es desplegado en su dispositivo; si el *UsuarioX* tiene más de una sesión activa, su nuevo alias será desplegado en sus otros dispositivos. Lo mismo sucedería si el *UsuarioX* cambia sus preferencias desde su *Smartphone*, cuando indique desde otro dispositivo que desea cambiar las preferencias y el servidor envíe la última lista de preferencias, el *UsuarioX* verá la lista que modificó en el *Smartphone*.

El mecanismo que se utiliza define una sesión virtual por cada usuario, es decir, el usuario se conecta con uno o más dispositivos con la misma sesión virtual.

4.1.3. Dispositivo

Dispositivo es el complemento del componente **Sesión Virtual**. Se encarga de mantener una conexión activa con el servidor. En función de las capacidades de los dispositivos, este componente es donde se realizan y responden las peticiones de servicios. De acuerdo a los parámetros que se envían y reciben, podemos dividir las peticiones en dos tipos:

- **Peticiones virtuales**: estas peticiones reciben como parámetros de entrada: a) el identificador del usuario destino y b) el tipo de servicio que se desea ejecutar (llamada VoIP, transferencia de archivos, envío de mensajes de texto). Las peticiones virtuales se envían desde el dispositivo al servidor y pueden recibir

dos tipos de respuesta: a) la *URL* destino, que corresponde al dispositivo al cual se le debe hacer la pericón real o b) un mensaje de error (el usuario no existe o el usuario no se encuentra conectado).

- **Peticiones reales**: una vez que el dispositivo recibe la respuesta del servidor, identifica el tipo de respuesta, en caso de no ser un error, se trata de una dirección URL destino. Esta *URL* corresponde al dispositivo (con sesión activa) de mayor preferencia del usuario destinatario al cual se le quiere enviar la petición de servicio. Posteriormente, el dispositivo se encarga de enviar una petición real a la URL destino, los parámetros de las peticiones reales varían de acuerdo al tipo de servicio que se desee ejecutar:

  - La petición de **transferencia de archivo**, lleva como parámetros: a) el identificador del usuario origen, b) el nombre del archivo que se desea enviar y c) el tamaño en bytes del archivo.
  - Los **mensajes de texto** se envían sin llevar una petición real, i.e., una vez que el dispositivo recibe la *URL* destino, el mensaje se envía con el siguiente formato: el identificador del usuario origen al inicio, para que el destinatario sepa quién originó el mensaje, seguido de la cadena especial `_:_\msgtxt_:_` y finalmente el mensaje de texto.
  - La petición de **llamada VoIP** es un caso particular de los mensajes de texto, ya que antes de realizar la petición de la llamada, se envía un mensaje de texto al mismo puerto, a donde se envían los mensajes de texto, el formato es el siguiente: el identificador del usuario origen al inicio seguido de la cadena especial `_:_\\llamada`. Posteriormente, se envía la petición de llamada VoIP que genera el *PJSUA*, hacia el usuario destino.

El dispositivo únicamente recibe peticiones reales, o mensajes de texto, en ambos casos se generan eventos que avisan al usuario. En caso de tratarse de una petición, el usuario es el encargado de decidir que hacer con la petición y la respuesta es enviada directamente al dispositivo que realizó la petición. En caso de ser un mensaje de texto, el usuario recibirá la notificación del mensaje y abrirá una ventana o pestaña de *chat* (dependiendo del dispositivo). Si el usuario cuenta con una ventana de *chat* abierta, el dispositivo únicamente desplegará el mensaje en dicha ventana. En el caso particular de las llamadas de VoIP, el dispositivo recibe la petición SIP junto con el mensaje especial que indica una petición de llamada VoIP, de acuerdo con lo que responda el usuario, el dispositivo generará una respuesta SIP que será enviada al dispositivo que generó la petición.

En la sección 4.2 mostramos un ejemplo de cómo funcionan las peticiones.

4.1.4. Sesión real (SR)

La **Sesión Real** es el componente que representa la comunicación entre un dispositivo y el servidor. Es importante no confundir **Sesión Real** con **Sesión Virtual**.

La diferencia principal radica en que la **Sesión Virtual** es a nivel aplicación (los usuarios ven la misma información en todos sus dispositivos), y la **Sesión Real** es la comunicación a bajo nivel (un *socket* que conecta el dispositivo con el servidor [35]). Por lo tanto, es posible decir que una sesión virtual consta de al menos una sesión real.

Una SR corresponde a un dispositivo, por lo tanto, si el usuario inicia sesión, tendrá activa una única **Sesión Virtual**, pero el número de SR será igual al número de dispositivos con que inicie su sesión.

A través de este componente se realiza la comunicación C/S: inicio y término de sesión, cambios (alias, estado, preferencias y contraseña), actualizaciones eventuales (todo lo que implica el servicio de presencia), y envío de peticiones virtuales y respuestas de las mismas.

4.1.5. Sesión de comunicación

Este componente representa toda la comunicación P2P que se produce, es decir, la comunicación entre dos usuarios (dos dispositivos). Actúa cuando un usuario envía o responde una petición real. También se utiliza para el envío de mensajes de texto.

Cuando se realiza una transferencia de archivos, una llamada de VoIP o el envío de un mensaje de texto, dos usuarios interactúan entre sí sin necesidad de que los paquetes de datos pasen por el servidor, esta comunicación se realiza a través de un mecanismo P2P por medio del componente de *Sesión de comunicación*.

Como el lector puede apreciar, este componente se encarga de todo lo que implique interacción entre dos usuarios, o bien, es el que da soporte a la ejecución de los servicios.

4.1.6. Red

El componente de *Red* es la base de los componentes *Sesión real* y *Sesión de comunicación*. Su función principal es dar soporte a la comunicación C/S y P2P. El soporte consiste en cuatro funciones:

- Inicio de comunicación.- Para la comunicación C/S implica que el dispositivo puede conectarse con el servidor. Para la comunicación P2P [55] implica que un dispositivo puede comunicarse con otro sin importar que sean distintos, ni el mecanismo que usen para conectarse a la red (ya sea alámbrica o inalámbricamente).

- Mantenimiento de comunicación.- Para la parte C/S, implica que el dispositivo siempre está en contacto con el servidor, hasta el término de la sesión. En el caso P2P únicamente da soporte para la transferencia de archivos y llamadas VoIP, para que se lleven a cabo satisfactoriamente.

- Término de comunicación.- En el caso C/S, se encarga de terminar la sesión y avisar a todos los usuarios con sesiones activas. Para la comunicación P2P

se refiere a que se concluya una sesión de comunicación satisfactoriamente, y para el caso del envío de mensajes de texto, se encarga de que los mensajes se entreguen satisfactoriamente.

- Control de errores.- Esta parte es la más importante, ya que la comunicación podría bloquearse entre dos usuarios o entre un dispositivo y el servidor. El control de errores se utiliza para que en caso de que la aplicación se cierre de manera inesperada, la comunicación, ya sea C/S o P2P concluya sin producir errores.

4.1.7. Servidor

El *Servidor* mantiene el control de las sesiones reales. El *Servidor* no distingue sesiones virtuales, i.e., envía los mensajes de manera indistinta a todos los dispositivos que cuenten con una sesión activa.

Para mantener el control de los cambios, el servidor cuenta con una base de datos. La estructura de la base de datos se muestra en la Figura 4.2.

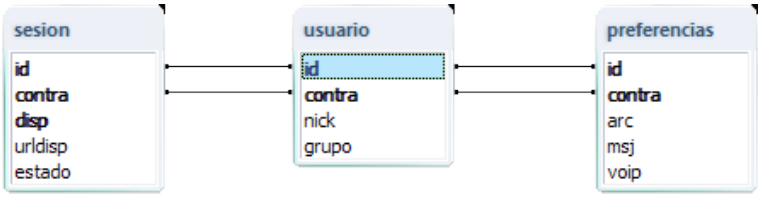

Figura 4.2: Estructura de la base de datos

La tabla *usuario* cuenta con los siguientes campos: identificador de usuario (*id*), contraseña (*contra*), alias (*nick*) y el grupo (*grupo*). Los primeros dos campos se utilizan como llave primaria, el identificador es único para cada usuario, el alias es la manera en que el usuario es observado en las ventanas de chat, finalmente el campo *grupo* no tiene ninguna función actual, sin embargo, se pensó para que se utilice en versiones futuras de SEGEUS. Por ejemplo, a) se pueden crear grupos para que los usuarios tengan ciertos privilegios con respecto al tráfico en la red, o b) si se crea un servicio de transferencia de archivos no presencial (i.e., que los usuarios puedan aceptar archivos sin necesidad de estar frente al dispositivo), se puede seleccionar a usuarios de un cierto grupo para que se reciban los archivos de esos usuarios, en caso de que el archivo viniera de algún usuario ajeno al grupo, será descartado.

La tabla *sesion* cuenta con los campos: identificador de usuario (*id*) y contraseña (*contra*) como llaves foráneas, dispositivo (*disp*), dirección (*urlsip*) y estado (*estado*). Los primeros dos en conjunto con *disp*, se utilizan para identificar las sesiones reales, ya que lo único que varía es el dispositivo. La *urlsip* se utiliza para almacenar la dirección actual donde el dispositivo está conectado al servidor, esta dirección es devuelta en caso de que se realice una petición virtual. Finalmente, *estado* (*online* y

offline) mantiene el estado de cada dispositivo, en versiones futuras el estado puede tener más tipos. A cada usuario le corresponden cuatro sesiones, ya que se pensó para tener hasta cuatro dispositivos (PC, *Laptop*, PDA y *Smartphone*) con sesiones activas simultáneamente.

La tabla *preferencias* cuenta con los siguientes campos: identificador de usuario (*id*), y contraseña (*contra*) como llaves foráneas, preferencias de recibo de archivos (*arc*), preferencias de recibo de mensajes (*msj*) y preferencias de recibo de llamadas VoIP (*voip*). Al igual que en la tabla *sesion*, los primeros dos campos sirven para identificar al usuario al cual le pertenece la lista de preferencias. Los campos restantes, es decir, *arc*, *msj* y *voip* se utilizan de la misma manera, es decir, el usuario elige de mayor a menor preferencia el dispositivo al cual desea que reciba la petición, o en su defecto el mensaje de texto.

En la Figura 4.3 podemos observar la secuencia de pasos necesarios desde el inicio hasta el término de la sesión de un usuario. Cuando un usuario inicia sesión (1), el servidor verifica que el identificador y la contraseña coincidan (2), si coinciden se le envía de regreso su alias y una lista que contiene los identificadores, alias y estados de los otros usuarios (3), de esta manera el usuario conoce los dispositivos en los cuales se encuentran en línea los otros usuarios. Una vez que un usuario inicia sesión, el servidor notifica al resto de los usuarios (4) y almacena la dirección del dispositivo en la tabla *sesion* (5). En ese momento, el usuario puede realizar las actividades que requiera con su sesión dentro de ese dispositivo (6). Cuando el usuario termina sesión (7) en un dispositivo (de manera normal o por error), el servidor notifica al resto de dispositivos con sesión activa (8) y posteriormente pone en cero la dirección en la tabla *sesion* (9).

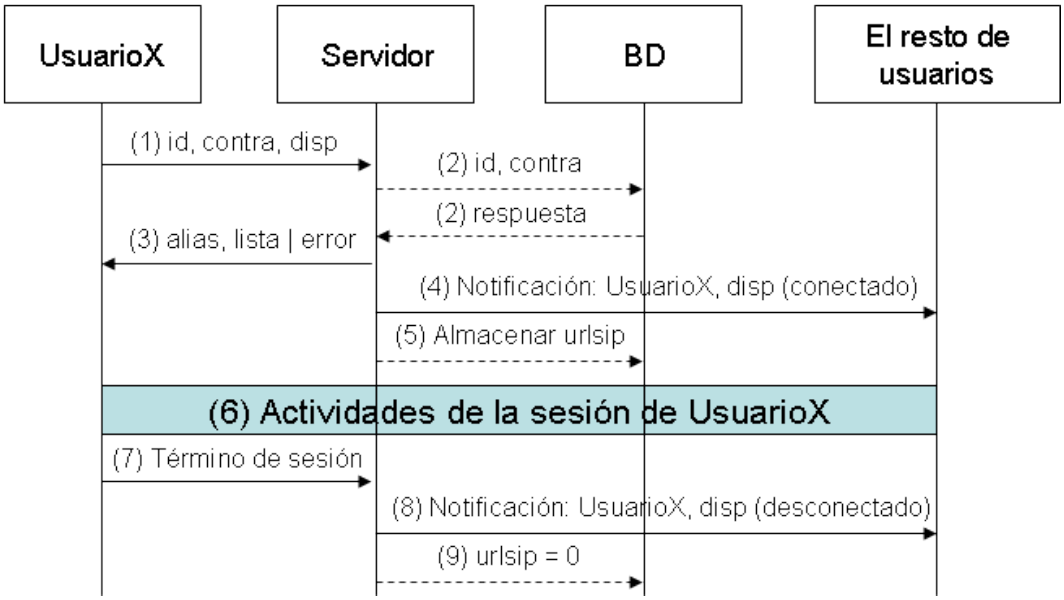

Figura 4.3: Diagrama de secuencia de inicio y término de sesión

En la Figura 4.4 podemos observar el diagrama de secuencia que muestra la manera en que el servidor atiende una petición virtual: cuando un usuario hace una petición virtual (1), es decir, envía el tipo de servicio que quiere ejecutar y el ID del usuario destino con el cual quiere interactuar. El servidor busca en la base de datos la lista de preferencias que corresponden al ID del usuario destino, dentro de la lista, busca al dispositivo que tenga asignada la mayor preferencia, posteriormente busca en la tabla *sesion* la dirección IP que corresponde al dispositivo (2), y si este dispositivo se encuentra en línea, devuelve al dispositivo que hizo la petición virtual la dirección a la cual debe realizar la petición real (3). En caso de que el usuario destino no exista, el servidor envía un mensaje de error (3), de igual manera en caso de que el usuario no tenga ningún dispositivo en línea (no confundir estado en línea con tener una sesión activa), el servidor envía mensaje de error al dispositivo que realizó la petición (3).

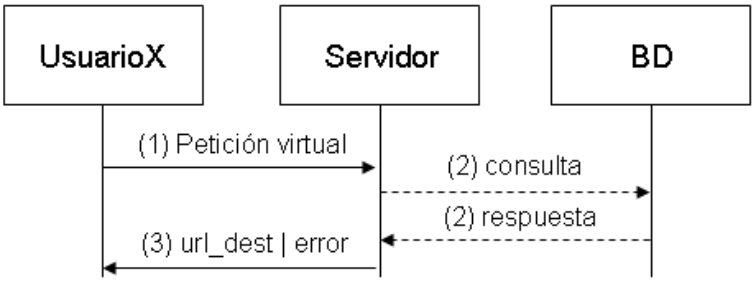

Figura 4.4: Diagrama de secuencia: respuesta de peticiones virtuales

El algoritmo que utiliza el servidor cuando recibe una petición virtual funciona de la siguiente manera:

```
1) Servidor recibe petición virtual (ID_destino, tipo_servicio)
2) Si el usuario existe, ir a 3) sino ir a 9)
3)    Servidor busca dispositivos de mayor a menor preferencia
      de acuerdo al tipo de servicio y los almacena en una lista
4)    Mientras no se termine de recorrer la lista de dispositivos,
      ir a 5), de lo contrario ir a 8)
5)        El dispositivo actual se encuentra en línea
6)            Enviar dirección destino y terminar.
7)        De lo contrario regresar a 4)
8)    Enviar mensaje de error y terminar
9) Enviar mensaje de error y terminar
```

4.2. Arquitectura de SEGEUS

Una vez entendido como funcionan los componentes de SEGEUS, es necesario abordar la arquitectura del servicio. La arquitectura con que cuenta el servicio es

híbrida, se compone de dos tipos de arquitecturas como se observó en la sección anterior: una arquitectura cliente-servidor (C/S) y una arquitectura punto a punto (*peer to peer* - P2P). Por otra parte, la arquitectura de SEGEUS es orientada a servicios, a continuación se abordarán cada una de las partes que conforman esta arquitectura.

4.2.1. Arquitectura orientada a servicios (service oriented architecture - SOA)

Una arquitectura orientada a servicios se define como aquella que se encarga de separar los procesos de negocios de las funciones automatizadas. Estas funciones son organizadas y catalogadas en diccionarios de servicios, es decir, los usuarios pueden acceder a los servicios ubicados en los diccionarios y utilizarlos como funciones, sin afectar el funcionamiento general de los procesos de negocios [56].

Dos de los estándares más importantes que implementan SOA son el de *Microsoft* y el de *Sun*. La arquitectura SOA ofrece un marco de diseño que permite la integración de aplicaciones independientes en un entorno de red para que se pueda acceder a sus servicios de manera remota. Una manera de implementar esta clase de servicios es utilizando los servicios Web [57].

Un servicio Web es una funcionalidad concreta que se encuentra ubicada en la red y que realiza alguna tarea específica. En general estas aplicaciones utilizan estándares para el transporte, codificación y el protocolo de intercambio de información. La principal ventaja que presentan, es que permiten la comunicación entre diferentes plataformas [57].

A pesar de que SEGEUS no implementa los estándares SOA de *Sun* ni de *Microsoft*, la arquitectura fue pensada como una arquitectura orientada a servicios. Esto se debe a que básicamente, todos los servicios son independientes entre sí. Y el proyecto se realizó de esta manera, para permitir su escalabilidad, i.e., la introducción de nuevos servicios a futuro, además de dar al desarrollador mayor legibilidad en el código y portabilidad en el mismo. Esto quiere decir, que para implementar un nuevo servicio, este debe programarse, portarse como una nueva función en el código, darlo de alta en el servidor, y el servicio podrá ser utilizado en la versión modificada de SEGEUS, haciéndola compatible con las versiones anteriores.

4.2.2. Arquitectura cliente-servidor (C/S)

Una de las arquitecturas de comunicación de SEGEUS es la arquitectura C/S. Este tipo de arquitectura consiste en dos aplicaciones distintas:

La aplicación **Cliente** es aquella que realiza peticiones de servicio, las peticiones son enviadas a la aplicación **Servidor**, esta comunicación se realiza a través del componente de **Red**. Para nuestro proyecto desarrollamos tres aplicaciones **Cliente** diferentes (*clienteSegeus*, *clienteSegeusCE*, *clienteSegeusS60*), que practicamente realizan las mismas funciones, aunque trabajan de manera diferente por la implementación de

PJSUA que se realizó en los diferentes sistemas operativos [35].

La aplicación *Servidor* es aquella que da soporte a las peticiones realizadas por los clientes, y entrega respuestas a los mismos, generalmente para crear un servidor, es necesario trabajar con una aplicación multihilos que atienda a cada cliente en un hilo independiente [35].

SEGEUS cuenta con un servidor, como mostramos en la sección de componentes, el cual se encarga de atender a las peticiones virtuales de los dispositivos. En este caso la aplicación cliente corresponde al componente *Usuario*.

En la Figura 4.5 se muestra la parte C/S de SEGEUS.

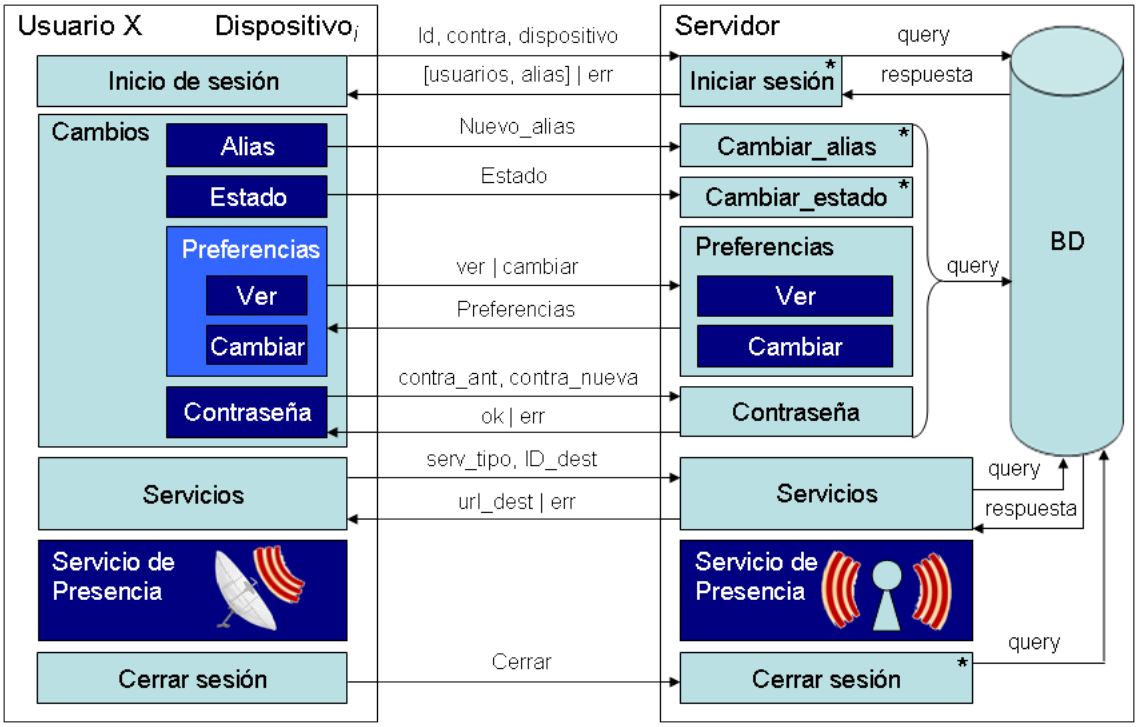

Figura 4.5: Parte C/S de SEGEUS, intercambio de mensajes entre un dispositivo y el servidor

Podemos observar en la Figura 4.5, que la aplicación **Cliente** (donde se tiene *UsuarioX* en su *Dispositivo$_i$*) cuenta con los módulos: *Inicio de sesión*, *Cambios* (con los submódulos *Alias*, *Estado*, *Preferencias* y *Contraseña*), *Servicios*, *Servicio de presencia* y *Cerrar sesión*. Además podemos observar que el *Servidor* cuenta con una base de datos y los módulos que forman la contraparte de los módulos de la aplicación *Cliente*.

La base de datos se explicó en la sección anterior. Por cada operación que realiza el servidor (a excepción del servicio de presencia) ejecuta consultas o realiza actualizaciones en la base de datos.

En el módulo **Inicio de sesión** se le pide al usuario que introduzca su ID, su contraseña y el dispositivo en el cual inicia su sesión (1). Este módulo se comunica con el módulo *Iniciar sesión* del servidor, el cual recibe el ID, la contraseña y el dispositivo que se está conectando el usuario. Posteriormente, realiza una consulta a la BD en la tabla *usuario* (2), y responde al cliente con su alias y una lista con los ID, usuarios, o en su defecto con un mensaje de error indicando al usuario que ha producido un error (3). Finalmente el *Servidor* avisa al resto de los usuarios sobre el inicio de sesión (4) y modifica la tabla *sesion* que corresponde al usuario (5), asignándole una dirección al dispositivo que inició sesión.

La siguiente función es el cambio de alias, que comunica al submódulo de **Alias** que se encuentra dentro del módulo **Cambios** en la aplicación **Cliente**, con el módulo **Cambiar_alias** de la aplicación **Servidor**. En el diagrama de secuencia (ver Figura 4.6) podemos observar que para realizar el cambio de alias, el usuario debe enviar el nuevo alias del usuario (1), una vez que el servidor recibe el nuevo alias, realiza la modificación a la BD en la tabla *usuario* (2), posteriormente el servidor notifica (3) a todos los usuarios sobre el cambio (inclusive al usuario que realizó el cambio).

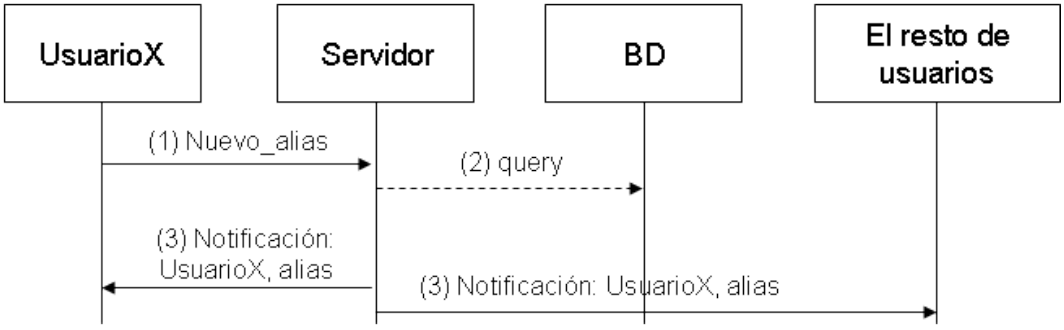

Figura 4.6: Diagrama de secuencia: cambiar alias

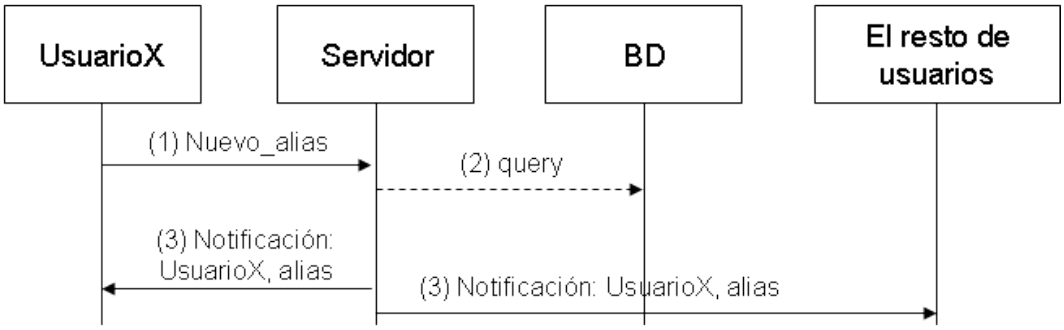

Figura 4.7: Diagrama de secuencia: cambiar estado

Continuando con la función cambio de estado, podemos observar en la Figura 4.5, que esta operación implica el submódulo **Estado** del módulo *Cambios* de la aplicación

Cliente con el módulo **Cambiar_estado** de la aplicación **Servidor**. Si vemos el diagrama de secuencia (ver Figura 4.7), esta operación implica que el usuario envíe al servidor, un mensaje indicando que va a cambiar el estado (1), posteriormente el servidor hará la modificación en la tabla *sesion* de la base de datos (2), en la tupla que corresponde al usuario y al dispositivo que realiza la petición. Finalmente el servidor enviará una notificación al resto de los usuarios, indicando el usuario y el dispositivo en donde se realizó el cambio de estado (3).

El siguiente submódulo de la aplicación cliente es el de **Preferencias**, este a su vez cuenta con los submódulos **Ver** y **Cambiar** que se comunican con el los submódulos con el mismo nombre que se encuentran en el módulo de **Preferencias** localizado en el servidor. Podemos observar en el diagrama de secuencia (Figura 4.8), que el usuario por medio del submódulo **Ver** realiza una petición al servidor para poder acceder a sus preferencias (1), el sevidor realiza una consulta en la tabla *preferencias* (2) y posteriormente envía la lista de preferencias al dispositivo que realizó la petición (3). Finalmente el usuario decide si cambia o no las preferencias, en el caso de que decida cambiarlas, es el submódulo **Cambiar**, quien envía al servidor la petición de cambio junto con la nueva lista de preferencias (4) y finalmente el servidor actualiza la BD con la nueva lista de preferencias (5).

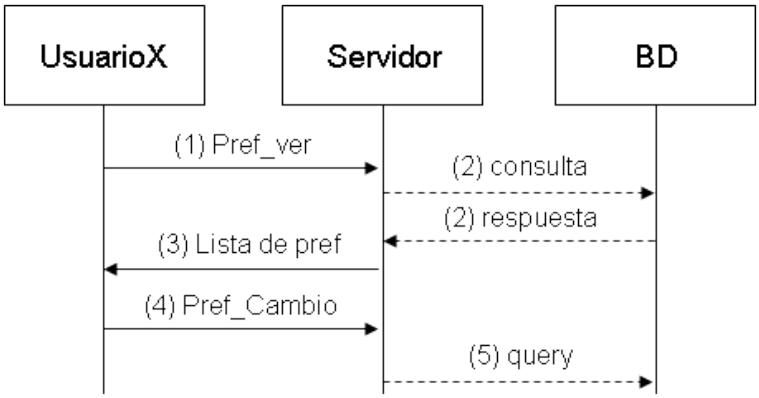

Figura 4.8: Diagrama de secuencia: cambiar preferencias

Para concluir con las funciones de *Cambios*, la operación cambiar contraseña involucra a los submódulos **Contraseña** de la aplicación **Cliente** con el módulo del mismo nombre de la aplicación **Servidor**. Si observamos el diagrama de secuencia (Figura 4.9) el cliente envía la contraseña anterior y la contraseña nueva (ésta última dos veces) al servidor (1). El servidor primero verifica que la contraseña anterior sea la correcta en la BD, y de ser así verifica que la contraseña nueva haya sido escrita correctamente, y en este caso hace la modificación en la BD (2). Finalmente si se efectúa el cambio, el servidor responde con un mensaje *OK* al dispositivo que realizó el cambio (3). En caso de que no se realice el cambio, se envía un mensaje de error al dispositivo (3).

Retomando la Figura 4.5, los servicios con tres submódulos (**Mensajes de texto**,

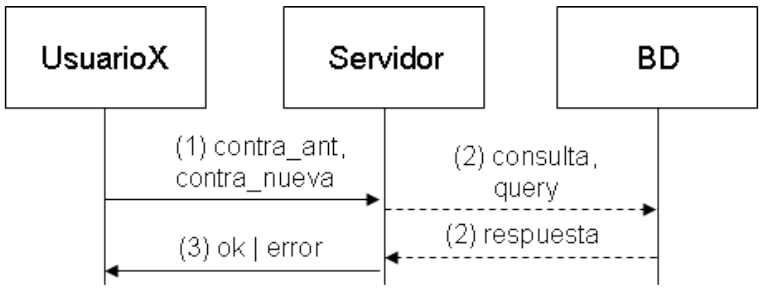

Figura 4.9: Diagrama de secuencia: cambiar contraseña

Llamadas VoIP y **Transferencia de archivos**) se observan como un único módulo (tanto en la aplicación **Cliente** como en el **Servidor**). Este módulo se encarga de atender las peticiones virtuales buscando en la tabla *preferencias* y en la tabla *sesion* el dispositivo adecuado. En cualquier caso envía un mensaje al dispositivo que realizó la petición, ya sea indicándole la dirección destino o el error. En la sección 4.2.3 explicaremos a detalle cómo operan estos módulos.

El servicio de presencia es considerado independiente de los otros servicios. Si observamos de nuevo la Figura 4.5, podemos notar que los módulos *Iniciar sesión*, *Cambiar_alias*, *Cambiar_estado* y *Cerrar sesión*, de la aplicación **Servidor** tienen un caracter *, esto implica que los mensajes que sean recibidos por parte del **Cliente** en estos módulos activará un evento que notifique al resto de los clientes.

Finalmente el módulo **Cerrar sesión** de la aplicación **Cliente** se comunica con su contraparte del **Servidor**, retomando la Figura 4.3 observamos que el término de una sesión lo único que implica es enviar un mensaje del cliente al servidor (7), el servidor avisará al resto de los usuarios (8) y finalmente realizará la modificación a la BD (9).

4.2.3. Arquitectura punto a punto (peer to peer - P2P)

El otro mecanismo de comunicación que utilizamos en la arquitectura híbrida, es el mecanismo P2P. Un sistema que trabaja bajo una arquitectura P2P se caracteriza (aunque no necesariamente) porque las aplicaciones son pares, es decir un dispositivo y otro se comunican utilizando aplicaciones similares. Actualmente la arquitectura P2P se refiere a dos aplicaciones que se comunican entre sí, sin utilizar un servidor de por medio (o por lo menos, no un servidor centralizado) [27].

Una aplicación que se utilice como un punto en una arquitectura P2P, puede funcionar como servidor y cliente a la vez, es decir, puede realizar peticiones o responderlas.

En el caso de SEGEUS, la aplicación **Cliente**, que se mostró en la sección anterior, funciona también como un punto en la parte P2P. La comunicación P2P en SEGEUS se implementa cuando se ejecutan los servicios de mensajes de texto, llamadas VoIP

y transferencia de archivos.

En la Figura 4.10 se observa como se establece la comunicación P2P. Las flechas de color azul corresponden a la comunicación C/S, las flechas de color verde sólo representan comunicación C/S de las aplicaciones hechas para *Windows Mobile* y para *Symbian* y las flechas de color naranja corresponde a la comunicación P2P.

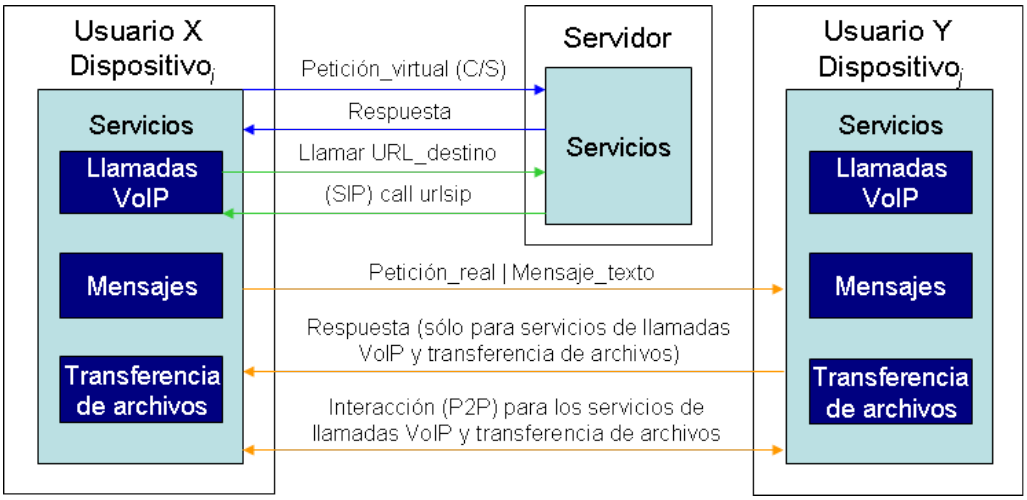

Figura 4.10: Comunicación híbrida, de C/S a P2P

De acuerdo con la Figura 4.10, si complementamos su contenido con el diagrama de secuencia de la Figura 4.4 podemos generar un nuevo diagrama de secuencia (ver Figura 4.11), este diagrama de secuencia muestra la manera en que se establece una sesión de comunicación: observemos que el usuario realiza una petición virtual (1). El servidor busca en la base de datos la dirección IP que corresponde al dispositivo (2) al cual se le debe enviar la petición real, y la devuelve en caso de que el dispositivo se encuentre en línea (3). En caso de que el usuario destino no exista, el servidor envía un mensaje de error (3), de igual manera en caso de que el usuario no tenga ningún dispositivo en línea (no confundir estado en línea con tener una sesión activa), el servidor envía mensaje de error al dispositivo que realizó la petición (3).

Si el usuario recibe la dirección destino, realiza la petición real (4), o en su defecto envía el mensaje de texto al dispositivo destino (4). En el caso de *Windows Mobile* y *Symbian* (para el servicio de llamadas de VoIP), antes de realizar la petición real, la aplicación **Cliente** envía un mensaje al servidor indicándo la dirección SIP a la cual desea enviarle la petición real (3.1), posteriormente el servidor envía un mensaje SIP al dispositivo que hizo la petición, este mensaje indica al dispositivo que debe realizar una petición SIP de llamada VoIP (3.2), y posteriormente se realiza la petición real (4).

Si el usuario destino recibe una petición real corresponde una respuesta (5), si la respuesta fue positiva finalmente puede establecerse la sesión de comunicación entre dos usuarios (6).

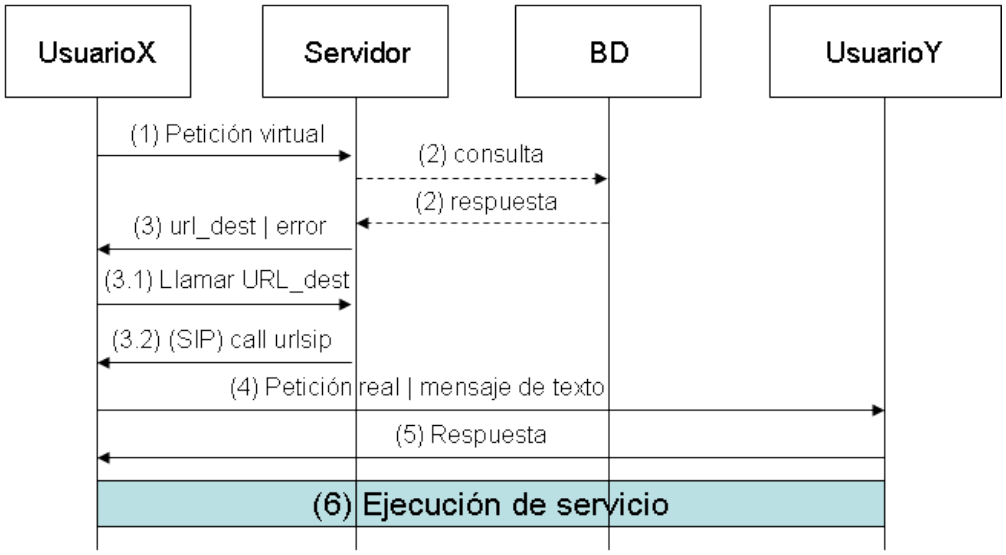

Figura 4.11: Diagrama de secuencia: establecimiento de una sesión de comunicación

Capítulo 5

Implementación, ejecución y funcionamiento

Una vez que hemos presentado la arquitectura de SEGEUS y explicado el funcionamiento de los componentes, nos enfocaremos a explicar la manera de cómo instalar la aplicación servidor y posteriormente la instalación de los clientes en los diferentes sistemas operativos. Finalmente analizaremos el funcionamiento de la aplicación y la manera en que operan las diferentes funciones.

5.1. Instalación y ejecución del servidor

El proceso de instalación de SEGEUS en una red local es sencillo, en las siguientes subsecciones explicaremos los pasos que deben seguirse para poder instalar las aplicaciones clientes y el servidor.

Como observamos en las secciones de componentes (ver sección 4.1.7) y arquitectura (ver sección 4.2.2), el **Servidor** es la entidad funcional que da el soporte para todo el funcionamiento de SEGEUS.

El **Servidor** puede ser instalado tanto en *Windows* como en *Linux*, para esto es necesario que tengamos instalado *Python 2.x* en el equipo donde se va a ejecutar el **Servidor**. Lo primero que debemos hacer es crear la base de datos. Para esto, es necesario tener instalado *MySQL* [72] (que se puede descargar del respositorio [73]) en la máquina que será el servidor.

En los archivos incluidos con este trabajo, agregamos el *script* encargado de crear la BD que nosotros diseñamos para el **Servidor**.

Una vez creada la BD, es necesario introducir los datos, para esto desarrollamos una aplicación que cuenta con una interfaz gráfica como aparece en la Figura 5.1. Con esta aplicación, podemos crear usuarios y de esta forma llenar los campos necesarios para poder darlos de alta dentro del servidor de SEGEUS.

Podemos crear usuarios en cualquier momento, esta aplicación está instalada en el servidor, sin embargo, hemos creado también un servicio para administrar la base de datos de manera remota (ya sea desde un sistema *Windows*, *Linux*, *Windows Mobile* o

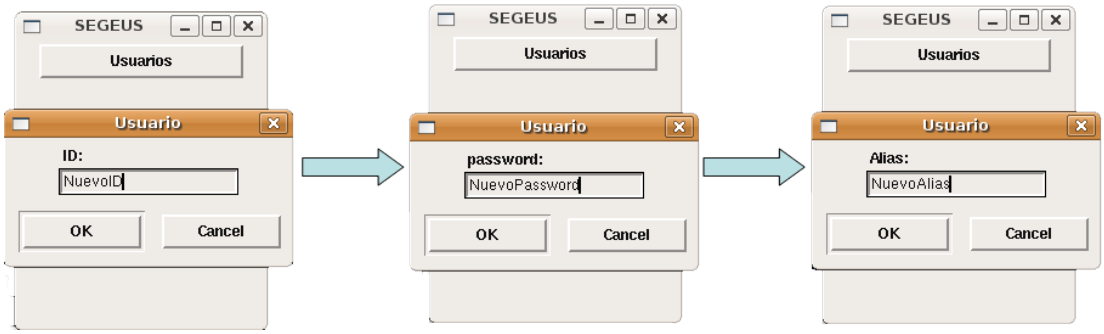

Figura 5.1: Aplicación para llenar la base de datos de SEGEUS

Symbian S60), este servicio lo que permite es que un usuario de SEGEUS con derechos administrativos (en este caso le llamamos *Admin*) pueda acceder de manera remota a la base de datos y realizar operaciones tales como observar quién se encuentra en línea y con qué dispositivos, agregar nuevos usuarios, modificar los usuarios ya existentes y/o eliminarlos.

Ejecución del servidor

Una vez instalada la base de datos y creada la aplicación para agregar nuevos usuarios, necesitamos instalar el módulo de *Python* de *PJSUA* en el servidor (ver sección 3.5.1 para *Linux* y sección 3.5.2 para *Windows*).

Para ejecutar el servidor en *Linux*, únicamente debemos dirigirnos al directorio donde se encuentre el archivo `servidorSegeus.py` y posteriormente ejecutar el siguiente comando:

```
$ python servidorSegeus.py
```

En el caso de *Windows* se ejecuta `servidorSegeus.py` dando doble clic en el ícono correspondiente al archivo.

Figura 5.2: Aplicación servidorSegeus.py (A)

Cuando ejecutamos el **Servidor**, lo primero que aparece es una ventana en la cual es necesario teclear la IP donde se aloja el **Servidor** (ver Figura 5.2), posteriormente la aplicación levantará una serie de módulos correspondientes al *API PJSUA* que

corresponden a los servicios de SIP. Además, el **Servidor** creará un cuenta virtual de SIP que esté asociada a la dirección IP del equipo donde se ejecuta y al puerto 5060, como podemos observarlo en la parte inferior de la Figura 5.3. Posteriormente el servidor se dispondrá a esperar clientes.

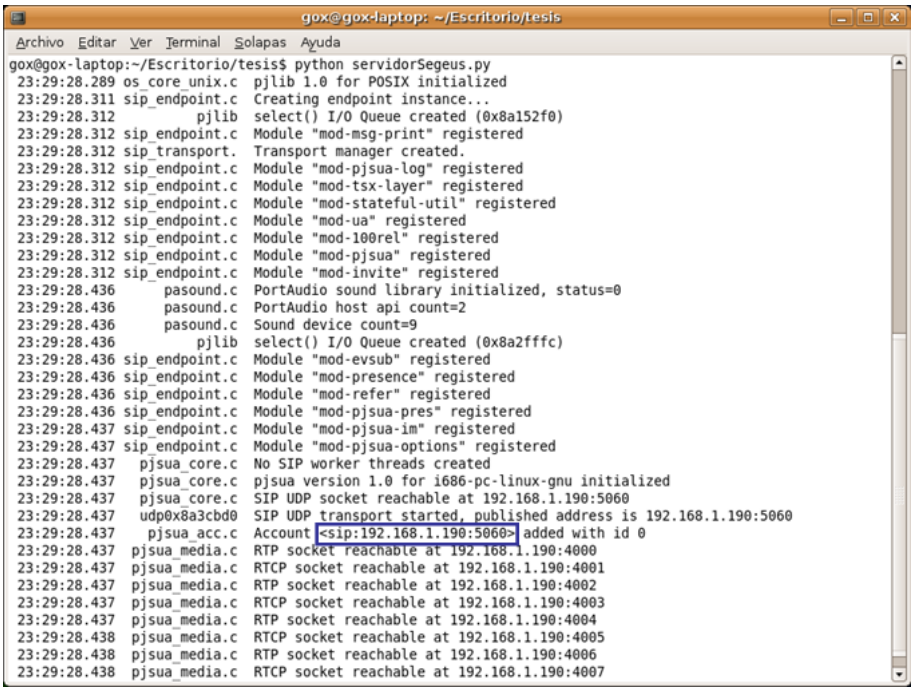

Figura 5.3: Aplicación servidorSegeus.py (B)

5.2. Instalación de la aplicación cliente

Existen tres aplicaciones cliente creadas para el desarrollo de nuestro proyecto: *clienteSegeus* (que funciona en *Windows* y *Linux*), *clienteSegeusS60* (para ejecutar en los dispositivos *Symbian S60*) y *clienteSegeusCE* (que funciona bajo un ambiente *Windows Mobile* 5.0 y 6.0).

5.2.1. Instalación de la aplicación cliente en Windows

Para instalar la aplicación *clienteSegeus* en el sistema operativo *Windows*, necesitamos contar con las siguientes herramientas:

- *Python* 2.x (2.4, 2.5 o 2.6) en su versión para *Windows*.
- El segundo módulo de *Python* de *PJSUA* para *Windows* (ver sección 3.5.2).

Una vez que contamos con estas herramientas, debemos ejecutar el `script setup.py` (también desarrollado por nosotros). Este archivo se encarga de copiar la biblioteca `pjsua` de *Python* que modificamos, para que la aplicación pueda comunicarse con los otros sistemas operativos.

El `script setup.py` modifica la biblioteca de `pjsua` para las versiones 2.3, 2.4 y 2.5 de *Python* instaladas en *Windows*.

Para ejecutar el **Cliente** en *Windows*, lo único que procede, es ejecutar `cliente Segeus.py` o `clienteSegeus.pyc` (para ejecutar sin modo consola).

5.2.2. Instalación de la aplicación cliente en Linux

Cómo mencionamos en la sección 5.1.2, la aplicación *clienteSegeus* es compatible con *Windows* y con *Linux*, es decir, la misma aplicación funciona en ambos sistemas operativos. Para instalar esta aplicación en *Linux* (*Ubuntu* 8.04 u 8.10) requerimos tener las siguientes herramientas:

- *Python* 2.x (2.4, 2.5 o 2.6) en su versión para *Linux*.
- El segundo módulo de *Python* de *PJSUA* para *Linux* (ver sección 3.5.1).

Una vez que tengamos estas herramientas, debemos ejecutar el archivo `setup.py` de la siguiente manera:

```
$ sudo python setup.py
```

De igual manera que en *Windows*, este *script* se encarga de copiar la biblioteca `pjsua` de *Python* modificada, para que la aplicación pueda comunicarse con los otros sistemas operativos.

Para ejecutar el **Cliente** en *Linux*, lo único que debemos hacer es ejecutar `cliente Segeus.py` como sigue:

```
$ python clienteSegeus.py
```

5.2.3. Instalación de la aplicación cliente en Symbian S60

Para instalar la aplicación *clienteSegeusS60* en el sistema operativo *Symbian S60*, necesitamos las siguientes herramientas:

- *PythonForS60* 1.4.4 o 1.4.5.
- *PythonScriptShell* 1.4.4 o 1.4.5 (el que corresponda a la versión de *PythonForS60* seleccionada).

Cómo mencionamos en la sección 3.5.4, creamos un **intermediario** [58] para la comunicación entre *PJSUA* para la versión de *Symbian S60* y *Python*. Lo que hicimos fue modificar el código de *PJSUA*, para generar los archivos `Symbian_ua.sis` y `Symbian_ua.sisx`, que al ser instalados en el dispositivo S60, proporcionan la comunicación con el servidor de SEGEUS directamente (ver sección 5.2).

Con el *Symbian_ua* (la versión modificada) instalado, lo único que resta es copiar el archivo `clienteSegeus.py` a la carpeta de *Python* que se encuentra en la memoria masiva del dispositivo móvil.

Para ejecutar `clienteSegeusS60.py` es necesario ejecutar la aplicación *Symbian_ua* antes, y posteriormente ejecutar el *script* `clienteSegeusS60.py` desde la consola de *Python*.

5.2.4. Instalación de la aplicación cliente en Windows Mobile

Para instalar la aplicación *clienteSegeusCE* en el sistema operativo *Windows Mobile*, es necesario tener instaladas las siguientes herramientas:

- *Active Sync* 4.5 (requerido por el *Windows Mobile SDK*).
- *PythonCE* (*Python* para *Windows Mobile*).
- *Tkinter* para *PythonCE*.

De igual manera que en la versión de *Symbian*, creamos un **intermediario** que comunica la versión de *C* de *PJSUA* con *PythonCE*. Lo que hicimos fue modificar el código de *PJSUA*, de tal manera que el archivo ejecutable `segeus.exe` generado, realice la función de llamadas VoIP y se ejecute detrás de `clienteSegeusCE.py` (ver sección 5.2).

Debemos instalar *Active Sync* en una computadora con sistema operativo *Windows* para que nos permita copiar archivos de la PC al PDA. Una vez instalado, debemos copiar el ejecutable `segeus.exe` y `clienteSegeus.py` en el dispositivo móvil (no es necesario tener una carpeta en específico, para el proyecto creamos una carpeta llamada `segeus` en donde alojamos todos los archivos) [53].

Necesitamos tener instalado *PythonCE* (en nuestro caso utilizamos la versión 2.3.4) en el dispositivo móvil, una vez instalado *PythonCE*, es necesario que tengamos *Tkinter* instalado. En caso de no tener instalado *Tkinter*, debemos de realizar lo siguiente:

- Descargar *Tkinter* del repositorio [74].
- Descomprimir el archivo `Tkinter-Files.zip` (este contiene la carpeta `Windows` y `tcl8.4.3`).
- Copiar el contenido de la carpeta `Windows` a la carpeta `Windows` del dispositivo móvil.

- Crear una carpeta con el nombre `lib` en el directorio `Archivos de programa` del dispositivo.
- Dentro del directorio `tcl8.4.3` encontraremos dos carpetas `library` y `tk8.4`. Debemos copiar el contenido de esta última carpeta, dentro del directorio que creamos (`lib`).
- Dentro de `lib` debemos crear otro directorio llamado `tcl8.4`. Finalmente copiaremos el contenido de la carpeta `library` dentro del último directorio creado [75].

Para ejecutar `clienteSegeusCE.py` lo único que necesitamos es ejecutar la aplicación `segeus.exe` antes y posteriormente ejecutar el *script* `clienteSegeusCE.py`.

5.3. Funcionamiento

Una vez que hemos explicado la manera de instalar la aplicación SEGEUS (servidor y clientes) en las diferentes plataformas, es momento de presentar la aplicación mostrando su funcionamiento y la compatibilidad entre plataformas.

En cualquiera de las plataformas en las que ejecutemos la aplicación lo primero que debemos hacer es teclear la dirección IP del servidor (donde se encuentre en ejecución *servidorSegeus*). Esto establecerá la conexión entre el dispositivo y el servidor. Si la conexión se establece correctamente, lo que procede es teclear el ID de usuario, la contraseña (para autenticación [54]) y seleccionar el tipo de dispositivo que estamos utilizando.

5.3.1. Inicio de sesión

Si el ID y la contraseña son correctos, se inicia una **sesión real** entre el dispositivo y el servidor. Si es la **primer sesión real** que se inicia, entonces se activa la **sesión virtual**.

En la Figura 5.4 mostramos la ventana estándar de inicio de sesión (que se repite en todos los dispositivos excepto en *Symbian*), y en la Figura 5.5 presentamos la ventana que corresponde al inicio de sesión en un dispositivo *Symbian*.

Figura 5.4: Inicio de sesión estándar

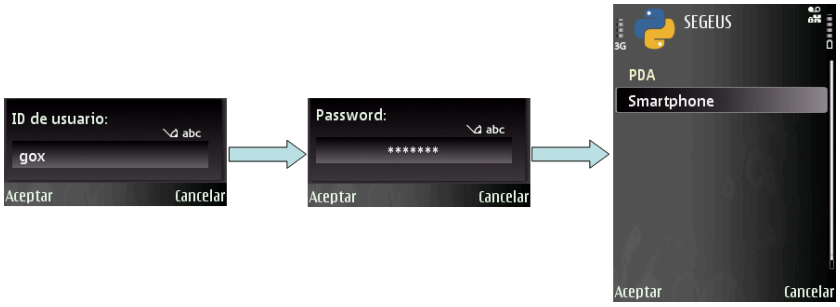

Figura 5.5: Inicio de sesión en dispositivos Symbian

5.3.2. Ventana principal clienteSegeus

En la Figura 5.6 mostramos la ventana principal de la aplicación cliente en *Windows* (es la misma ventana en *Linux*) que puede ser ejecutada tanto en una *Laptop* como en una PC convencional.

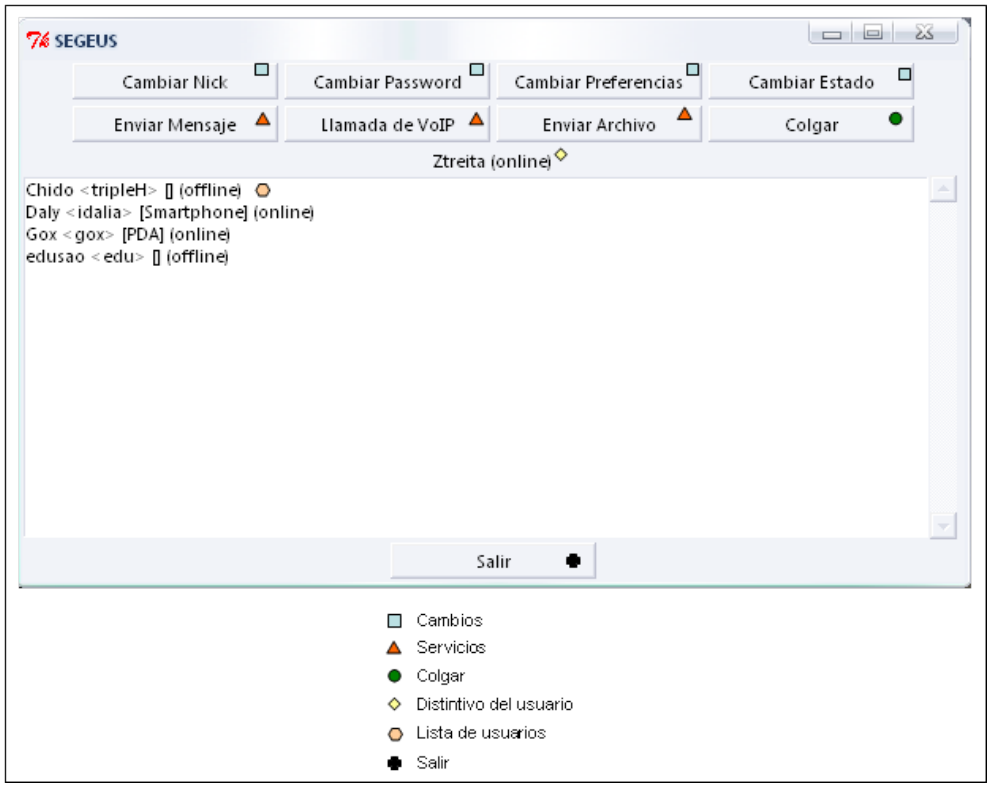

Figura 5.6: Aplicación clienteSegeus

La ventana principal de *clienteSegeus* cuenta con las siguientes características:

En la parte superior aparecen los botones donde se localizan las funciones principales de SEGEUS: Los botones de arriba, son las funciones de *Cambios* (*Nick*, *Password*, *Preferencias* y *Estado*), debajo de estos aparecen los botones de *Servicios* (*Mensajes de texto*, *Llamadas de VoIP* y *Transferencia de archivos*), y el botón de *Colgar*.

El siguiente elemento es el *distintivo de usuario*. El elemento está conformado por el alias del usuario y entre paréntesis se encuentra el estado local (*online* y *offline*), el estado que aparece en este elemento, únicamente corresponde al dispositivo (sesión real).

Debajo del *distintivo de usuario* aparece la lista de usuarios SEGEUS. El formato de la lista es el siguiente:

```
Alias <ID> [Dispositivo(s)] (Estado global)
```

De acuerdo con este formato, un usuario puede conocer el Alias, ID y los dispositivos con los que se encuentren en línea el resto de los usuarios. El estado global pertenece a la **Sesión Virtual** del usuario, se encuentra *online* cuando se tiene al menos un dispositivo en línea, y *offline* cuando se tienen todos los dispositivos en este estado. Por ejemplo, si observamos la Figura 5.6, notaremos que el usuario **idalia** se encuentra con una sesión activa en su ***Smartphone***.

Finalmente en la parte inferior aparecerá el botón *Salir*.

5.3.3. Ventana principal clienteSegeusCE

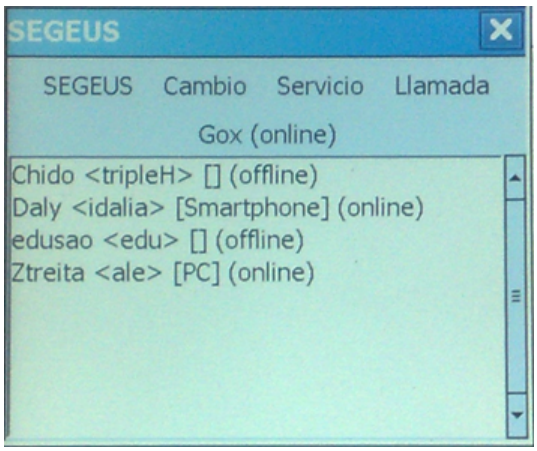

Figura 5.7: Aplicación clienteSegeusCE

La ventana principal de *clienteSegeusCE* es casi idéntica a la de *clienteSegeus* (que se ejecuta tanto en sistemas *Windows* y *Linux*), sin embargo, dado el tamaño de la pantalla de un dispositivo móvil (PDA o *Smartphone*) omitimos los botones.

En la parte superior de la ventana, colocamos una barra de menú donde se encuentran las funciones que corresponden a los botones de la aplicación *clienteSegeus* (ver Figura 5.7), es decir:

- **SEGEUS**.- En este submenú aparece la opción de *salir*.
- **Cambio**.- Aquí es donde aparecen las opciones de cambios (*Alias*, *Estado*, *Preferencias* y *Contraseña*).
- **Servicio**.- Aquí aparecen los servicios de *Transferencia de archivos*, *Mensajes de texto* y *Llamadas de VoIP*.
- **Llamada**.- Finalmente en este submenú aparece la opción de *colgar*.

5.3.4. Ventana principal clienteSegeusS60

En la ventana *clienteSegeusS60* la estructura de la aplicación cambia con respecto a las anteriores. En *clienteSegeusS60* aparece la lista de usuarios, siendo el primer elemento de la lista el *distintivo de usuario*. En la Figura 5.8 podemos observar la ventana principal de la aplicación *clienteSegeusS60*.

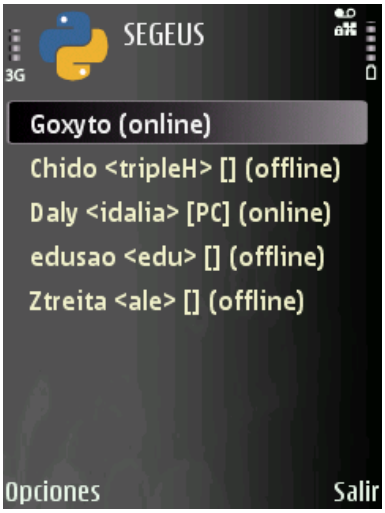

Figura 5.8: Aplicación clienteSegeusS60

Los siguientes elementos de la lista son los usuarios SEGEUS siguiendo el formato que tenemos en las versiones mostradas en las secciones anteriores.

El menú que aparece al oprimir el botón opciones, presenta las mismas opciones que las de la barra de menú mostrada en *clienteSegeusCE* y los botones presentados en *clienteSegeus*.

Si presionamos el botón central en el primer elemento de la lista, aparecen las opciones de cambio (*Alias*, *Estado*, *Preferencias*). Si presionamos el botón central en

alguno de los usuarios aparecen las opciones de servicio (*Mensaje de texto*, *Transferencia de archivos* y *Llamada VoIP*).

5.3.5. Funciones administrativas

Las funciones administrativas se dividen de la siguiente manera: *Cambio de alias*, *Cambio de estado*, *Cambio de preferencias* y *Cambio de contraseña*.

Cambio de alias

El cambio de alias implica que el usuario proporcione su nuevo alias. El nuevo alias es enviado al servidor, y este lo distribuye a todos los dispositivos que tengan una sesión real activa (incluyendo los dispositivos que pertenecen al usuario que realizó el cambio), sin importar el estado (ver Figura 5.9). Luego los dispositivos detectan el cambio y lo actualizan en la ventana de la aplicación.

Figura 5.9: Cambio de alias

Cambio de estado

El cambio de estado (*online* y *offline*) se realiza de la siguiente manera: un usuario decide cambiar de estado en cualquiera de sus dispositivos, automáticamente ese dispositivo hace el cambio para la aplicación que se ejecuta en él (lo llamamos cambio de *estado local*). Esta acción se le notifica al servidor, a diferencia del cambio de alias, el servidor únicamente avisa a los dispositivos que no pertenecen al usuario que realizó el cambio.

Si el usuario que hace el cambio tiene sus otros dispositivos desconectados o inactivos, entonces la notificación llega a los otros usuarios por medio de un cuadro de diálogo (ver Figura 5.10). Por el contrario, si el usuario que realizó el cambio tiene otros dispositivos en línea, entonces la notificación únicamente se refleja en la lista de usuarios de la aplicación (quitando o poniendo el dispositivo al que corresponde el

Figura 5.10: Cambio de estado (Sesión Virtual)

cambio). El ejemplo de esto último lo podemos observar en la Figura 5.11.

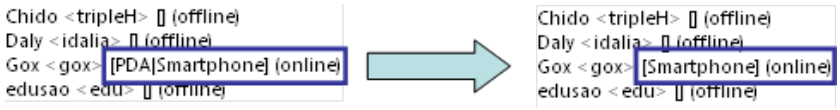

Figura 5.11: Cambio de estado (Sesión Real)

Cambio de preferencias

El cambio de preferencias del usuario se refiere al cambio de dispositivos en los que el usuario desea ejecutar los servicios. Por ejemplo, supongamos que el *UsuarioX* decide que el servicio de mensajes de texto se ejecute con las siguientes preferencias: *Laptop*, PC, *Smartphone* y PDA; de acuerdo con las preferencias elegidas por el usuario, cuando el servidor reciba una petición virtual para enviar un mensaje de texto al *UsuarioX*, el servidor buscará de acuerdo a la lista de preferencias, cuál es el dispositivo con mayor preferencia y cuyo estado sea *online*. Posteriormente el servidor responde la petición virtual con la dirección IP del dispositivo al cual debe enviarse el mensaje de texto. Si *UsuarioX* no tiene ningún dispositivo en línea, el servidor responde con un mensaje de error.

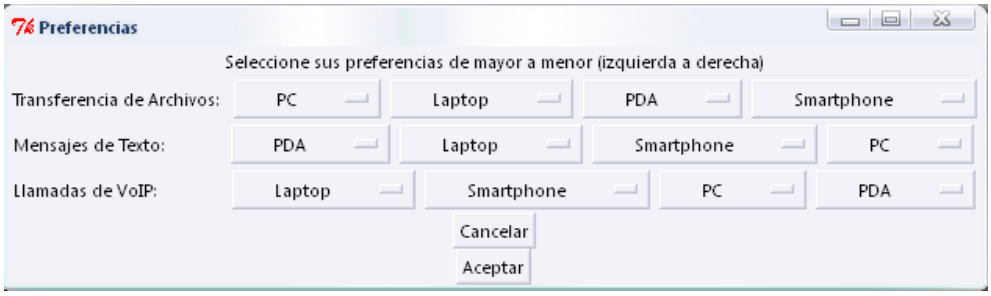

Figura 5.12: Cambio de preferencias (Windows)

El cambio de preferencias funciona de la siguiente manera. Un usuario decide realizar un cambio de preferencias. La petición que realiza el dispositivo al servidor

es la de consultar las preferencias. Un ejemplo de lista de preferencias aparece en la Figura 5.12, las preferencias se encuentran ordenadas de mayor a menor, es decir, de lado izquierdo se encuentra el dispositivo con mayor preferencia de acuerdo al servicio que se indica. Si el usuario realiza algún cambio en las preferencias procede a enviar el cambio realizado al servidor. En el caso específico de *clienteSegeusS60*, las preferencias se dividen en tres ventanas diferentes, una ventana para cada servicio (Transferencia de archivos, Mensajes de texto y llamadas VoIP). En la Figura 5.13 mostramos el ejemplo del cambio de preferencias en un dispositivo *Symbian*.

Figura 5.13: Cambio de preferencias (Symbian)

Cambio de contraseña

Para terminar con la sección de funciones administrativas, explicamos en qué consiste el cambio de contraseña. Para brindar un nivel de seguridad simple, el usuario puede cambiar la contraseña con la que inicia sesión en SEGEUS.

Para que el usuario cambie su contraseña, debe primero teclear su contraseña actual, esto para que el servidor verifique que realmente es el usuario dueño de la cuenta, quien está cambiando la contraseña. Una vez que el usuario escribe la contraseña actual, debe escribir su nueva contraseña dos veces, para evitar errores de escritura. Una vez que se realiza el cambio de contraseña, se envía al usuario una notificación indicando que la contraseña se modificó correctamente, o bien, en caso de que haya error de escritura (ya sea en la contraseña actual, o que la nueva contraseña no se haya escrito igual dos veces), el servidor responde con un mensaje de error.

En la Figura 5.14 observamos el ejemplo de la función cambiar de contraseña de la aplicación *clienteSegeusS60*.

5.3.6. Funciones de servicio

Como hemos mencionado, existen cuatro servicios principales (*Mensajería instantánea*, *Transferencia de archivos*, *Llamadas de VoIP* y *Presencia*).

Figura 5.14: Cambio de contraseña (Symbian)

Mensajería instantánea

Para comenzar explicaremos el funcionamiento del servicio de Mensajería instantánea o mensajes de texto. Cuando un usuario decide enviar un mensaje de texto lo primero que debe hacer es definir el usuario al cual le enviará el mensaje.

En SEGEUS implementamos tres métodos para seleccionar el usuario destino: el primero es seleccionar al usuario de la lista y elegir el servicio de mensajes de texto; el segundo método es elegir el servicio de mensajes de texto sin elegir algún usuario de la lista, se activa un cuadro de diálogo donde se pide al usuario proporcionar el ID del destinatario; finalmente el tercer método para elegir el usuario destino, es escribir directamente el mensaje en la ventana de chat que corresponda al usuario destino (esta ventana se abre una vez que se haya enviado o recibido un mensaje).

Cuando elegimos el usuario destino, el dispositivo genera una **petición virtual** que es enviada al servidor. El servidor verifica en primera instancia que el usuario destino sea correcto, si el usuario no existe, el servidor responde con un mensaje de error. Si el usuario destino existe, el servidor verifica la lista de preferencias de este último y busca el dispositivo con mayor preferencia que se encuentre en línea. Si el usuario destino se encuentra fuera de línea en todos sus dispositivos, el servidor responde la **petición virtual** con un mensaje de error. Si un dispositivo recibe una respuesta a la petición virtual que indique un error, éste lanza una notificación indicando que el usuario no esta en línea o que no existe. En caso de que el usuario si exista y se encuentre en línea, el servidor responde con la dirección IP a la cual se debe enviar el mensaje de texto (la que corresponde al dispositivo con mayor preferencia disponible).

Una vez que el usuario que origina el mensaje de texto recibe una respuesta correcta a la **petición virtual**, se procede a escribir el mensaje y se envía a la dirección recibida por parte del servidor. El formato del mensaje de texto es el siguiente:

```
IDUsuarioOrigen_:_\msgtxt_:_ContenidoDelMensaje
```

El primer elemento indica el usuario que envía el mensaje. El segundo elemento indica la acción que el usuario está realizando, que en este caso, se refiere al envío de un mensaje de texto. Finalmente el tercer elemento es el contenido del mensaje.

Figura 5.15: Ventana de conversación (Windows)

En la Figura 5.15 podemos ver un ejemplo del envío de mensajes de texto en un sistema *Windows*. El nombre de la ventana se conformará de la palabra **Conversación** seguida de un guión y posteriormente el **ID del usuario** con el que se está conversando. A cada mensaje de la conversación le antecede la hora en que se envió o recibió el mensaje y el alias actual de los usuarios, tanto del que envía como del que recibe los mensajes.

Figura 5.16: Ventana de conversación (Symbian)

La ventana es similar en los sistemas *Linux* (ya que es la misma aplicación que en *Windows*) y *Windows Mobile*, dado que los gráficos se crearon con la misma biblioteca (*Tkinter*). Sin embargo, la aplicación cambia en *Symbian*, un ejemplo de la ventana

de conversación lo podemos observar en la Figura 5.16. La pestaña tiene como título el ID del usuario con que se está conversando. El texto se conforma de igual manera que como aparece en *Windows*. Finalmente las opciones que aparecen en el menú *Opciones* (este menú aparece en cualquier aplicación típica de *Symbian S60* en la parte inferior izquierda) corresponden a los servicios que se pueden realizar con ese usuario.

Transferencia de archivos

Para el servicio de envío de archivos se selecciona al usuario destino de igual manera que en el servicio de mensajes de texto. Una vez que elegimos la opción de enviar un archivo, se envía la **petición virtual** al servidor, que es procesada de igual manera que en el servicio de mensajes de texto.

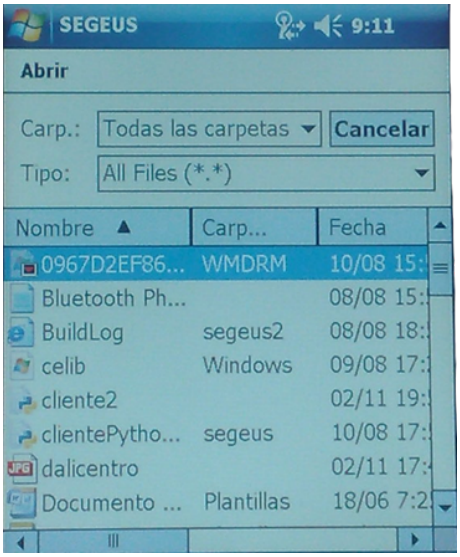

Figura 5.17: Envío de archivo (ventana de selección de archivo Windows Mobile)

Si el servidor envía una dirección IP como respuesta, el dispositivo que envió la **petición virtual** abre una ventana de *selección de archivos* (ver Figura 5.17). Esta ventana varía de acuerdo al sistema operativo en que se esté ejecutando SEGEUS.

Una vez que se selecciona un archivo, se envía una **petición real** a la dirección IP que pertenece al dispositivo del usuario destino. La **petición real** en este caso tiene el siguiente formato:

```
IDUsuarioOrigen_:_NombreArchivo_:_LongitudArchivo
```

Donde:

- *IDUsuarioOrigen*: indica el usuario que desea enviar el archivo,
- *NombreArchivo*: es el nombre del archivo y,

- *LongitudArchivo*: corresponde al tamaño en bytes del archivo.

Cuando un usuario recibe una petición de archivo, aparece un cuadro de diálogo donde le indica quién desea transferir el archivo, el nombre y el tamaño del mismo. Este cuadro pregunta al usuario si desea recibir el archivo. Si el usuario acepta el archivo aparece una ventana de transferencia del mismo (excepto en *clienteSegeusS60*, donde únicamente aparece un mensaje al inicio y otro al final de la transferencia).

Un ejemplo de la transferencia de archivo lo podemos observar en la Figura 5.18. La transferencia se indica por medio de una línea que va cambiando de color conforme se va transfiriendo el archivo y debajo de esta línea aparece el número de *bytes* (Kb o Mb según el tamaño del archivo) que se han transferido y el total de *bytes* (Kb o Mb) del archivo.

Figura 5.18: Transferencia de archivo (Windows)

Llamadas VoIP

El servicio de llamadas VoIP funciona similar a los otros servicios en cuanto a la **petición virtual**. Sin embargo, la petición real varía, dependiendo del sistema donde se está ejecutando la aplicación **Cliente**.

Cuando se envía la **petición real** de una llamada VoIP, siempre se envía en dos tiempos. En primer lugar (y en todos los casos) se envía un mensaje de texto al dispositivo con el cual se pretende establecer la llamada VoIP. El mensaje que se envía lleva el siguiente formato:

```
IDUsuarioOrigen_:_\llamada
```

Donde el primer elemento indica el ID del usuario que desea realizar la llamada y el segundo elemento indica que el mensaje no debe procesarse como un mensaje de texto sino como una petición de llamada VoIP. De esta manera cuando un dispositivo recibe un mensaje de este tipo, la aplicación abre un cuadro de diálogo preguntando al usuario si desea responder o no la llamada VoIP (ver Figura 5.19).

En segundo lugar debe enviarse la petición SIP para realizar la llamada VoIP. Esta petición es enviada por medio de *PJSUA*. La aplicación *clienteSegeus* envía la petición directamente de dispositivo a dispositivo, sin embargo, para las aplicaciones *clienteSegeusCE* y *clienteSegeusS60* difiere el modo en que se envía la petición SIP.

En *clienteSegeusCE* y *clienteSegeusS60*, la aplicación (creada con *Python*) envía un mensaje al servidor indicando la función que desea realizar con SIP, estas funciones

Figura 5.19: Mensaje que indica llamada VoIP (Windows)

se dividen en: llamar (enviando *call_make* y posteriormente *sip:urldestino:5060*), colgar (enviando *call_colgar*) y finalmente salir (enviando *salir*).

Cuando el servidor recibe un mensaje *call_make*, envía un mensaje SIP al dispositivo que desea hacer la llamada, este mensaje indica la dirección SIP a la cual debe marcar (*sip:urldestino:5060*). De esta manera cuando un dispositivo recibe un mensaje SIP (en la aplicación creada en *C*) que indica que la acción a realizar es una llamada VoIP, entonces el dispositivo manda la petición SIP de dispositivo a dispositivo.

De igual manera cuando el servidor recibe un mensaje *call_colgar*, envía un mensaje SIP al dispositivo indicando que debe colgar una llamada. De esta forma, cuando el dispositivo recibe un mensaje SIP (en la aplicación creada en *C*) que indica que la acción es colgar, debe interrumpir la llamada activa, o no contestar en caso de que alguien esté esperando por una petición de llamada VoIP.

Lo mismo sucede cuando el servidor recibe un mensaje *salir*, envía un mensaje SIP, indicando al dispositivo que cierre de manera automática la aplicación creada en *C*.

Con lo anterior, logramos completar el **intermediario** para comunicar la aplicación creada en *Python* y la aplicación creada en *C*.

5.3.7. Servicio de presencia

El servicio de presencia lo manejamos de manera independiente a los demás servicios, debido a que este servicio funciona de manera diferente. El servicio de presencia está basado en la **comunicación por eventos**, los cuales son mensajes que se generan a partir de otros eventos remotos y de los cuales el servidor notifica a los usuarios de SEGEUS.

Los eventos que originan estos mensajes son: *inicio de sesión*, *cambio de alias*, *cambio de estado* y *término de sesión*.

Cuando un usuario **inicia sesión** (con su ID y contraseña correctos), se generan tres eventos en el servidor: a) la respuesta del servidor con el alias y la lista de usuarios de SEGEUS, b) la modificación en la tabla de sesiones de la base de datos (donde se almacena la dirección IP del dispositivo que acaba de iniciar sesión, y se modifica el estado de la sesión que corresponde a dicho dispositivo), y c) el mensaje de notificación que envía a todos los dispositivos con sesión activa (aunque el estado de estos sea fuera

de línea), indicando el ID del usuario que inició sesión junto con su alias y dispositivo.

Cuando un dispositivo recibe una notificación de **inicio de sesión** por parte del servidor, puede generar dos eventos: a) si el usuario que inició sesión no tenía ningún dispositivo en línea, el evento que se genera es mostrar un aviso por medio de un cuadro de diálogo indicando la acción de inicio de sesión y/o b) el evento que modifica la lista de usuarios, agregando el dispositivo a la lista de dispositivos que corresponden al usuario que inició sesión.

Cuando un usuario **cambia de alias**, se generan dos eventos en el servidor: a) la modificación en la tabla de usuarios de la base de datos (sustituyendo el alias anterior por el nuevo) y b) el envío de mensajes de notificación a todos los dispositivos con sesión activa (incluyendo a los dispositivos del usuario que realizó el cambio de alias).

Si un dispositivo recibe una notificación de **cambio de alias** pueden activarse dos eventos: a) si el dispositivo que recibe el mensaje pertenece al usuario que realizó el cambio de alias, entonces se modifica el *distintivo de usuario* colocando el nuevo alias, y b) si el dispositivo que recibe el mensaje pertenece a un usuario distinto al que realizó el cambio de alias, entonces la modificación se realiza en la lista de usuarios, sustituyendo el alias del usuario que realizó el cambio.

Cuando un usuario **cambia de estado**, primero se genera un evento en el dispositivo que realizó el cambio, se modifica el estado en el *distintivo de usuario*. Posteriormente se envía el mensaje al servidor.

Si el servidor recibe un **cambio de estado** por parte de algún dispositivo: a) realiza el cambio en la tabla de sesiones que corresponde al dispositivo (modificando el estado de la sesión) y b) notifica al resto de los usuarios sobre el cambio de estado (enviando un mensaje con el nuevo estado, el dispositivo y el ID de quien realizó el cambio).

Cuando un dispositivo recibe una notificación de **cambio de estado** por parte del servidor pueden ejecutarse dos eventos: a) si el cambio de estado pertenece al único dispositivo en el cual se encontraba en línea el usuario, o bien, no tenía ningún dispositivo en línea, entonces se muestra un cuadro de diálogo que indica el ID, alias, dispositivo y estado del usuario que generó el cambio y b) si el usuario que generó el cambio de estado, tenía en línea un dispositivo diferente al que utilizó para realizar el cambio de estado, entonces el evento que se genera es únicamente modificar la lista de usuarios, ya sea agregando o quitando (según sea el caso) el dispositivo en el cual se realizó el cambio de estado.

Si un usuario **termina una sesión** ocurren los siguientes eventos en el servidor: a) modifica la tabla sesiones de la base de datos (poniendo en cero la dirección IP del

dispositivo que terminó la sesión y cambiando el estado de la sesión a *fuera de línea*), b) envía un mensaje notificando al resto de los usuarios a cerca del término de sesión (el mensaje contiene el ID del usuario y el dispositivo en el que terminó su sesión), y c) si el usuario que termina su sesión lo hace desde un dispositivo con *Windows Mobile* o *Symbian* estos envían un mensaje especial al servidor, este mensaje es procesado y produce un mensaje SIP como respuesta por parte del servidor, indicando al dispositivo que debe cerrar de manera automática la sesión SIP.

Cuando un dispositivo recibe un mensaje del servidor indicando el **término de sesión** de algún usuario, sucede lo mismo que cuando recibe una notificación del servidor indicando el cambio de estado (en este caso del estado fuera de línea).

Capítulo 6

Pruebas y análisis resultados

Una vez que explicamos el funcionamiento de SEGEUS, lo que procede es hablar de las pruebas que realizamos para demostrar la funcionalidad de nuestro proyecto.

6.1. Pruebas de la base de datos

Primero que nada es importante detallar que la base de datos es necesaria, debido a que SEGEUS es un servicio que gestiona sesiones de usuarios. La base de datos almacena a todos los usuarios del servicio, sirve además para saber en qué dispositivos está activa su sesión y las direcciones de los mismos, otra de las actividades de la base de datos es poder conocer las preferencias de los usuarios. Finalmente, si deseamos ampliar el servicio e implementarle nuevos módulos como más servicios, o por ejemplo implementar un módulo para la gestión de los dispositivos, la base de datos sería una buena opción para ello.

Con respecto a la base de datos, lo primero que probamos fue que las operaciones con la misma funcionaran de manera correcta. Primero las inserciones de nuevos elementos a la base de datos, posteriormente las consultas y finalmente las modificaciones. Para realizar estas pruebas creamos una interfaz en *Python* que se encarga de las operaciones para el manejo de la BD, esta interfaz la agregamos posteriormente a la aplicación servidor final.

Para realizar las funciones con *MySQL* en *Python* es necesario importar el módulo *MySQLdb* de la siguiente manera:

```
import MySQLdb
```

Posteriormente se coloca dentro de un bloque *try except* la siguiente instrucción:

```
db=MySQLdb.connect(host='nombre_host', user='nombre_usuario',
        passwd='password',db='nombre_bd')
cursor=db.cursor()
```

La cual iniciará una sesión en *MySQL*, una vez hecho lo anterior se pueden realizar las operaciones con la base de datos desde *Python*.

```
comando = ‘‘Comando de SQL’’
cursor.execute(comando)
```

Con la sintaxis anterior, se ejecuta la operación indicada por la variable *comando*, en el caso de realizar una selección, se utiliza la sintaxis `resultado=cursor.fetchall()` debajo de la instrucción *cursor.execute(comando)*, en este caso, la respuesta de la selección será almacenada en la variable *resultado*.

Las únicas operaciones de la BD en SEGEUS son la actualización y la selección, sin embargo, también creamos un servicio simple de administración la base de datos de forma remota. Esta aplicación permite realizar las operaciones básicas: consultar las sesiones activas, agregar y eliminar usuarios existentes. El funcionamiento con la base de datos utilizando *Python* fue satisfactorio, en la Figura 6.1 podemos observar un ejemplo del funcionamiento de la creación de usuarios, de manera remota desde un sistema *Windows*.

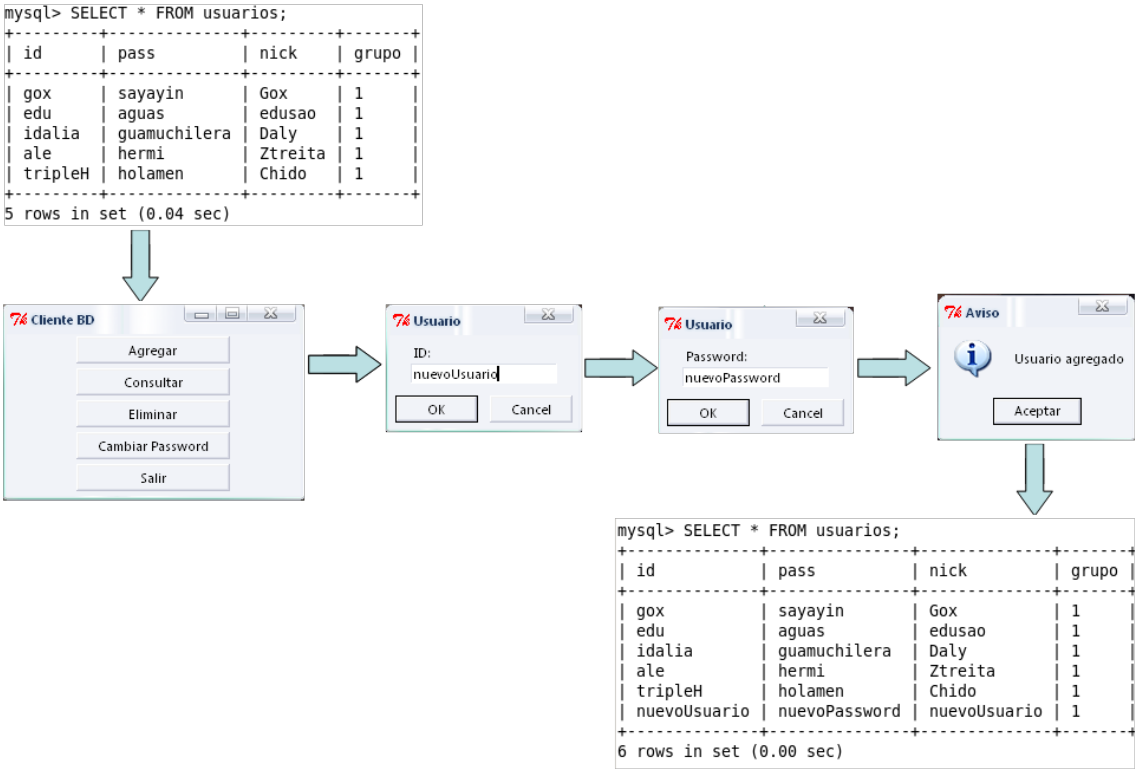

Figura 6.1: Creación de un nuevo usuario de manera remota desde un sistema *Windows*

6.2. Pruebas de PJSUA

Paralelo a las pruebas de las operaciones con la base de datos, probamos los servicios (mensajes, llamadas VoIP y presencia) de *PJSUA* del módulo de *Python*. Como

Cuadro 6.1: Pruebas de la biblioteca PJSUA

Servicio de PJSUA	**Sistema operativo**		
	Windows y Linux	**Symbian**	**Windows Mobile**
Presencia	Funciona de manera satisfactoria.	Funciona de manera satisfactoria, pero debe modificarse el código.	La función está definida, pero debe programarse.
Mensajes	Funciona de manera satisfactoria.	Los mensajes se reciben y despliegan, pero no existe una función para enviarlos.	Los mensajes se reciben pero no se despliegan, y no existe una función para enviarlos.
Llamadas	Funciona de manera satisfactoria.	Funciona de manera satisfactoria, aunque hay que modificar el código de la función marcar.	Funciona de manera satisfactoria.

mencionamos anteriormente, estos servicios únicamente funcionaron en *Windows* y *Linux*, sin embargo, cuando probamos *PJSUA* en su versión en *C* para los dispositivos móviles obtuvimos resultados satisfactorios en el recibo de mensajes SIP y en las llamadas VoIP.

Los resultados de las pruebas de PJSUA se observan en el cuadro 6.1.

Como vemos en el cuadro 6.1, se deben realizar ciertas modificaciones para que PJSUA funcione de manera satisfactoria:

- **Presencia**: para el servicio de presencia en *Symbian*, debe modificarse el código dado que hay una constante que se define desde el inicio (*SIP_DST_URI*), esta constante es la que se recibe como argumento en la función *Subscribe*. Esta función recibe como argumento una dirección SIP, la cual se dará de alta en el servicio de presencia, es decir, el usuario conocerá el estado de las direcciones que se suscriban con esta función. La modificación que debe hacerse es para que se le permita al usuario teclear las direcciones que desea dar de alta en el servicio de presencia, ya que el código por defecto recibe la dirección definida como *SIP_DST_URI*. En el caso de *Windows Mobile*, la función que corresponde al servicio de presencia sólo está definida, pensamos que en versiones futuras de *PJSUA*, sea implementada.

- **Mensajes**: para el servicio de mensajes en *Symbian* se debe crear la función para enviar mensajes, de igual modo esperamos que los desarrolladores de *PJSUA* la implementen en versiones futuras. En cuanto al servicio de mensajes en *Windows Mobile*, se reciben, pero la interfaz no está diseñada para que se desplieguen, y de igual modo que en *Symbian*, la función para enviar los mensajes, no existe.

- **Llamadas VoIP**: en el servicio de llamadas VoIP de *Symbian*, sucede lo mismo que en el de presencia, las llamadas se realizan a la dirección que esté definida en la variable *SIP_DST_URI*, por lo tanto, debe realizarse una modificación para que al momento de realizar la llamada, el usuario pueda teclear la dirección SIP destino a quien desea realizar la llamada.

6.3. Intermediario

Como mencionamos anteriormente, nosotros realizamos una modificación en el servicio de llamadas VoIP de *PJSUA*, la cual se encarga de poder ejecutar el servicio de llamadas VoIP de manera remota tanto en *Symbian* como en *Windows Mobile*.

Para lograr lo anterior, primero probamos la compatibilidad de los mensajes SIP entre distintos sistemas operativos, es decir, que los mensajes enviados por el módulo de *Python* desde *Linux* o desde *Windows*, fueran recibidos en *Symbian* y en *Windows Mobile*.

Una vez probado lo anterior, utilizamos el servicio de mensajes SIP (del módulo de *Python*) para enviar órdenes desde el **Servidor** hacia los dispositivos *Symbian* y *Windows Mobile* (que son recibidos en el módulo de *C*). Las órdenes que pueden enviarse por medio de los mensajes SIP son las siguientes:

- **Realizar llamada**: el **Servidor** envía un mensaje SIP con el siguiente formato:

  ```
  sip:URL:5060
  ```

  De esta manera, cuando un dispositivo recibe este mensaje SIP, realiza un petición de llamada VoIP a la URL indicada en el mensaje.

- **Colgar**: el **Servidor** envía un mensaje SIP con el caracter `h`. Cuando un dispositivo recibe este mensaje puede: a) terminar una sesión VoIP activa o b) responder con un *NO* a una petición de llamada VoIP entrante.

- **Salir**: el **Servidor** envía un mensaje con el caracter `w`. Cuando los dispositivos reciben este mensaje, terminan la ejecución de la aplicación PJSUA de manera automática.

Para que el **Servidor** envíe estas órdenes por medio de mensajes SIP, es necesario que los dispositivos envíen la orden que desean ejecutar desde la aplicación **Cliente** creada en *Python*. El resultado de lo anterior, es el **intermediario** que conecta a las aplicaciones de *Python* y *C* de los dispositivos *Symbian* y *Windows Mobile*.

6.4. Pruebas de las funciones de cambios

Una vez probado el funcionamiento de *PJSUA* en modo consola, decidimos crear una interfaz gráfica que volviera amigables los servicios. Como la biblioteca de *Tkinter*,

contiene las mismas funciones tanto para *Linux* como para *Windows* y *Windows Mobile*, decidimos realizar las primeras pruebas en *Ubuntu*.

La primera aplicación cliente que desarrollamos fue creada y probada en una *Laptop* con *Ubuntu* 8.04, las pruebas que se realizaron en esta aplicación fueron las funciones administrativas (cambio de alias, de estado, de preferencias y de contraseña) y posteriormente probamos que la comunicación cliente-servidor se llevara a cabo de manera satisfactoria. Una vez que se probaron estas funciones, portamos la aplicación a una PC con *Ubuntu* 8.10.

Ya instalada la aplicación en la *Laptop* y la PC, probamos en primer lugar, que las funciones de cambios (alias, estado y preferencias) se vieran reflejadas en ambos dispositivos. Las pruebas de las funciones administrativas se realizaron de la siguiente manera.

- **Cambio de alias**.- Realizamos las pruebas de cambio alias verificando que:
  1. Se viera reflejado el cambio en el dispositivo donde se hizo el cambio.
  2. Se viera reflejado en otro dispositivo con la misma sesión virtual activa.
  3. Finalmente que se viera reflejado en un dispositivo de un usuario diferente al que realizó el cambio.
- **Cambio de estado**.- Realizamos las pruebas de cambio de estado verificando que:
  1. Se viera reflejado el cambio en el dispositivo donde se hizo el cambio.
  2. Se viera reflejado en un dispositivo de un usuario diferente al que realizó el cambio, de la siguiente manera: a) al hacer el cambio de estado, el usuario que lo hizo tuviera otro dispositivo en estado *en línea* y que b) al hacer el cambio de estado, el usuario que lo realizó no tuviera otro dispositivo con sesión activa[1].
- **Cambio de preferencias**.- Realizamos las pruebas de cambio de preferencias verificando que:
  1. Se viera reflejado el cambio en el dispositivo donde se hizo el cambio.
  2. Que el usuario que realizó el cambio revisara sus preferencias utilizando otro dispositivo (observando los cambios realizados).
  3. Se viera realizado con la ejecución siguiente de un servicio, e.g., hacer el cambio de preferencias referente al mensaje de texto y que el siguiente mensaje que recibiera el usuario, llegara de acuerdo a la modificación de sus preferencias.
- **Cambio de contraseña**.- Revisamos que al hacer un cambio de contraseña el usuario iniciara sesión en otro dispositivo y tuviese que introducir su nueva contraseña.

[1]Tener una sesión activa no necesariamente implica que el estado del usuario sea *en línea*

6.5. Pruebas de los servicios

Para la parte del servicio de **Presencia**, probamos que cuando un usuario realizara cambios de estado y de alias, éste se viera reflejado en sus propios dispositivos y en los dispositivos de los otros usuarios. Posteriormente agregamos mensajes que indican cambios a los otros usuarios, añadiendo de esta manera el concepto básico de conciencia de grupo.

En cuanto al servicio de **Mensajes de texto** y **Transferencia de archivos**, utilizamos el servicio que brinda *PJSUA*, sin embargo con *Windows Mobile* y *Symbian* no podíamos enviar mensajes SIP, únicamente recibir. Entonces optamos por utilizar *sockets* para implementar ambos servicios.

Finalmente para el servicio de **llamadas VoIP**, decidimos combinar nuestra aplicación con el servicio de llamadas VoIP de *PJSUA*. Inicialmente capturábamos el evento de recepción de peticiones de llamada VoIP de *PJSUA*, sin embargo, de esta manera era imposible identificar el ID del usuario que estaba realizando la petición de la llamada. Por lo anterior, optamos por enviar un mensaje de texto indicando una petición especial que se refiere a una llamada VoIP (ver sección 5.3.6). Una vez que se envía este mensaje, se manda la petición de llamada VoIP desde SIP (en el caso especial de *Symbian* y *Windows Mobile* la petición SIP se envía utilizando el **intermediario**). De esta manera se probó el funcionamiento del servicio de llamadas VoIP entre las aplicaciones cliente instaladas en *Ubuntu*.

6.6. Pruebas de interfaz gráfica

Una vez probadas todas las funciones, se portó la aplicación al sistema operativo *Windows XP*. A pesar de que la comunicación cliente-servidor funcionaba de manera satisfactoria, al iniciar sesión con un dispositivo *Ubuntu*, la aplicación en *Windows* se bloqueaba.

El bloqueo se debía a que en ***Windows***, las operaciones con la biblioteca de gráficos *Tkinter* no pueden ejecutarse usando hilos (*threads*) independientes, es decir, todas las operaciones con los gráficos deben realizarse en el hilo de ejecución donde se haya creado la ventana principal. Para corregir lo anterior, creamos una función llamada `graficos`. Esta función, se encarga de los eventos como avisos, peticiones, y de la creación y modificación de ventanas de chat y de transferencia de archivos. Para poder llamar esta función, utilizamos la siguiente instrucción:

```
root.after(100, graficos, "tipoDeOperacion_:parametros")
```

Donde `root` es el nombre de la ventana principal, *after* es el método que indica, que la función gráficos se ejecutará en el hilo de ejecución de *root* después de 100 milisegundos, *tipoDeOperacion* se refiere al evento que se está generando, y los parámetros se envían de distinta manera dependiendo de la operación que se vaya a realizar.

Con esta modificación, se corrigieron los errores de *Windows* y la aplicación se volvió compatible para ambos sistemas (*Ubuntu* y *Windows XP*).

La siguiente aplicación cliente que desarrollamos, fue para ***Symbian S60***. Las pruebas las realizamos en un *Nokia N95* (8gb) con *Symbian S60 3ra edición Feature Pack 1*. Lo primero que hicimos fue portar el nuevo *Symbian_ua* que desarrollamos con *PJSUA*. Una vez que probamos que se le pudieran dar órdenes de manera remota al dispositivo por medio de mensajes SIP, decidimos portar el cliente en el dispositivo.

Para probar el cliente en *Symbian* tuvimos que crear una interfaz gráfica con *PyS60*, esta interfaz presentaba los mismos problemas que aparecieron en *Windows* (los eventos gráficos no pueden modificarse en hilos de ejecución independientes). Para corregir este problema creamos una función llamada *actualizar*, la cual se llama a sí misma cada 3 centésimas de segundo y se ejecuta en el hilo de ejecución principal. La función *actualizar*, verifica el estado de las variables globales (*peticiones*, *avisos*, etc.) que se modifican en los hilos de ejecución independientes, realizar alguna operación en caso de que alguna variable global cambie y finalmente restablecer su valor inicial.

Para concluir la aplicación, bastó con crear el **intermediario** (ver sección 5.3.6) entre la aplicación *Symbian_ua* y *clienteSegeusS60*, para poder realizar las llamadas VoIP. Una vez terminada la aplicación, realizamos las pruebas de todos los servicios entre los distintos clientes, obteniendo resultados satisfactorios.

Posteriormente, portamos el cliente de *Windows* y *Linux* en un dispositivo con *Windows Mobile*. Las pruebas las realizamos en un PDA *HP IPAQ 216* con sistema operativo *Windows Mobile* 6.0. Para esta aplicación cambiamos la manera en que se desplegaban los gráficos, debido a que la pantalla en un dispositivo PDA es de menor tamaño que una pantalla de *Laptop* o PC. Finalmente la función de llamada VoIP, la tomamos de la aplicación *clienteSymbianS60*, ya que para *Windows Mobile* también se requería del medio entre *clienteSegeusCE* y la aplicación de *PJSUA* que modificamos para el sistema *WindowsCE*.

Una vez concluida la aplicación se realizaron las pruebas entre los distintos clientes de todo el sistema en conjunto, los resultados obtenidos fueron satisfactorios, aunque el *clienteSegeusCE* muestra una velocidad inferior a la de los otros clientes, en una red con muchos equipos, debido a la potencia de la tarjeta de red del PDA.

6.7. Análisis de resultados

Finalmente de acuerdo con las pruebas del sistema podemos realizar un análisis de resultados y comparar nuestro trabajo con algunos de los trabajos relacionados. Para ello podemos observar el cuadro 6.2, donde encontramos las principales características

de nuestro trabajo.

Cuadro 6.2: Características de SEGEUS

Característica	**Descripción**
Servicios	SEGEUS proporciona los servicios de Mensajes de texto, transferencia de archivos, llamadas de VoIP y presencia.
Conciencia de grupo	Gracias al servicio de presencia, SEGEUS muestra un nivel básico de conciencia de grupo, ya que los usuarios conocen el estado, alias y los dispositivos con que otros usuarios se encuentran disponibles.
Interoperabilidad	Presenta interoperabilidad entre plataformas (Windows, Linux, Windows Mobile y Symbian).
Heterogeneidad	Presenta heterogeneidad entre sistemas operativos, dispositivos (PC, *Laptop*, PDA y *Smartphone*) y métodos de conexión (alámbrico e inalámbrico).
Movilidad	Los usuarios pueden conectarse a SEGEUS sin necesidad de utilizar cables.
Multisesión	Los usuarios pueden conectarse en más de un dispositivo y tener más de una sesión activa simultáneamente.
Flexibilidad	Los usuarios pueden dar preferencia al tipo de tráfico que desean recibir en sus dispositivos.
Portabilidad	SEGEUS está desarrollado para que pueda utilizarse en dispositivos con diferentes características y sistemas operativos, que van desde PC y *Laptops* (con sistemas *Linux* y *Windows*), hasta dispositivos PDA y *Smartphones* (con sistemas *Windows Mobile* y *Symbian*).
Escalabilidad	SEGEUS está pensado para que se puedan agregar nuevos servicios

Sin embargo, existen otras características que no hemos desarrollado, de entre las más importantes encontramos la siguientes:

- *Multiusuarios*: que implica que la aplicación pueda tener a más de un usuario en el mismo dispositivo de manera simultánea.
- *Multiprotocolo*: que soporte diferentes protocolos de aplicaciones ya existentes (*Skype*, *MSN*, *Google Talk*, etc.).
- *Emoticón*: que en los mensajes de texto se puedan agregar códigos que se representen como imágenes.
- *Conciencia visual*: que los usuarios puedan agregar imágenes a sus sesiones y que otros usuarios puedan observarlas.

De acuerdo con estas características y las mencionadas en el cuadro 6.2, podemos realizar una comparativa con diferentes sistemas de mensajería instantánea y revisar las características que nos proporcionan estos últimos, de esta manera podemos marcar las ventajas y desventajas que presenta nuestro trabajo con respecto a los ya existentes, en el cuadro 6.3.

Cuadro 6.3: Ventajas y desventajas de SEGEUS

Características	**Aplicación**				
	Segeus	**Messenger**	**Skype**	**Nimbuzz**	**Net-Meeting**
Servicios	✓	Agrega videollamada.	Agrega videollamada.	No recibe archivos en dispositivos móviles.	✓
Conciencia de grupo	✓	✓	✓	✓	×
Interoperabilidad	✓	Para otras plataformas debe usarse una aplicación compatible.	✓	✓	Compatible con *Ekiga*.
Heterogeneidad	✓	*Win*, *WinCE* Para otras plataformas deben usarse aplicaciones compatibles.	✓	✓	×
Multisesión	✓	Sólo en su última versión para PC y Laptops.	×	×	×
Movilidad	✓	✓	✓	✓	Sólo *Laptops*.
Flexibilidad	✓	×	×	×	×
Portabilidad	✓	Diferentes versiones pero todas para dispositivos con *Windows*.	✓	✓	×
Escalabilidad	**Hay en todos los casos.**				
Multiusuarios	×	Descargando Messenger Plus!.	×	×	×
Multiprotocolo	×	×	×	✓	×
Emoticón	×	✓	✓	✓	×
Conciencia visual	×	✓	✓	✓	×

El cuadro 6.3 podemos observar las ventajas y desventajas que tiene SEGEUS con respecto a las aplicaciones más populares, el lector puede observar que:

- De acuerdo a lo **servicios**, SEGEUS se muestra competitivo con el resto de las aplicaciones, ya que soporta los principales servicios (mensajería, transferencia de archivos y llamadas VoIP), y aunque supera a *Nimbuzz* (que no permite recibir archivos en dispositivos móviles), se ve superado por *Messenger* (que no solo implementa videollamada, sino que soporta juegos y sincornización con correo electrónico, entre otros servicios), *Skype* (soporta videollamada y conferencia de voz con más de dos usuarios), *Nimbuzz* (soporta conferencia de voz de aplicaciones distintas, e.g., un usuario *Nimbuzz* con un usuario *Skype* y otro de *Messenger* simultáneamente) y *Netmeeting* [76] (que implementa videollamada y un pizarrón compartido).

- SEGEUS implementa un nivel básico de **conciencia de grupo**, y la mayoría de las aplicaciones que proporcionan los servicios ya descritos, también cuentan con esta caraterística, aunque hay excepciones, e.g., en *Netmeeting* [59] sólo conoces la dirección IP del usuario.

- En cuanto a **interoperabilidad**, observamos que SEGEUS cuenta con aplicaciones compatibles para diferentes plataformas al igual que *Skype* y *Nimbuzz*. *Messenger* por su parte, únicamente desarrolla aplicaciones para usuarios *Microsoft*, por lo tanto la interoperabilidad que presente será con aplicaciones *Windows* y *Windows Mobile*, sin embargo, existen aplicaciones compatibles con *Messenger* (como *aMSN*, *Kopete* e inclusive *Nimbuzz* entre otras) aunque son desarrolladas por otros grupos de trabajo, finalmente *NetMeeting* sólo es desarrollado para versiones no embebidas de *Windows*, por lo tanto no presenta interoperabilidad, aunque si es compatible con *Ekiga* para *Linux*.

- Observando la característica de *heterogeneidad*, mostramos que SEGEUS puede ser ejecutado en dispositivos con diferentes características (tanto de *software* como de *hardware*), lo mismo sucede con *Skype* y *Nimbuzz*, ya que cuentan con versiones diferentes para distintos tipos de dispositivos. *Messenger* también cuenta con versiones para dispositivos móviles, aunque solo para aquellos dispositivos que tienen ambientes *Windows* por lo tanto la heterogeneidad que presenta es a medias. *NetMeeting* por su parte, únicamente es soportado por PC y *Laptop*.

- La característica de **movilidad** va ligada a las anteriores, por lo tanto, las aplicaciones comparadas presentan un nivel de movilidad, al poder ser instaladas en dispositivos móviles. *Netmeeting* por su parte, presenta movilidad limitada al sólo poder ser instalado en dispositivos móviles de tipo *Laptop*.

- En cuanto a *multisesión*, únicamente SEGEUS la soporta en su totalidad (ya que permite a los usuarios iniciar sesión en distintas plataformas de manera

simultánea), podemos decir, que esta característica es una innovación que ofrecemos. *Messenger* aunque presenta esta característica en su última versión, únicamente puede ser utilizada en *Laptop* y PC.

- La **flexibilidad** es otra de las inovaciones que presenta SEGEUS y que ninguna otra aplicación tiene, ya que permite dar preferencia a los usuarios de acuerdo a sus dispositivos y servicios, requeriría forzosamente que tuvieran la característica de multisesión.

- De acuerdo a la **portabilidad**, podemos observar que esta característica depende mucho de la heterogeneidad e interoperabilidad, *SEGEUS*, *Skype* y *Nimbuzz* tienen sus versiones para distintos dispositivos, lo que los vuelve portables. *Messenger* presenta esta característica pero únicamente para ambientes *Windows* y *Netmeeting* por su parte no es portable, ya que sólo puede ejecutarse en *Windows* para PC y *Laptop*.

- Han aparecido nuevas versiones con nuevas características de las aplicaciones comparadas, la **escalabilidad** es importante para seguir innovando, SEGEUS por su parte, está pensado para aplicaciones futuras, por lo tanto, se dejan algunos campos libres en la BD que pudieran ocuparse a futuro.

- La característica de **multiusuarios**, se presenta únicamente en *Messenger*, ya que permite iniciar sesión con más de una cuenta del mismo servicio, aunque para ello, se debe descargar *Messenger Plus!* y únicamente funciona en PC y *Laptop*. *Pidgin* [77] (que puede descargarse del sitio [78]) y *Amsn* entre otras aplicaciones, aunque no fueron escritas en la tabla, también soportan la característica de *multiusuarios* y sin tener que descargar otras aplicaciones.

- En cuanto a **multiprotocolo**, únicamente *Nimbuzz* lo presenta, existen otras aplicaciones como *Kopete* para *Linux* que también soportan esta característica.

- El **emoticón** es de lo más utilizado en aplicaciones de este tipo, sin embargo, por cuestiones de tiempo no hemos desarrollado esta característica en SEGEUS, ya que es una característica visual y no funcional, y preferimos dedicarle más trabajo a la funcionalidad.

- Finalmente la **conciencia visual**, es un nivel más elevado de conciencia de grupo y la mayoría de las aplicaciones del género cuentan con esta característica. En la última versión de la BD de SEGEUS, se agregó un campo al usuario que pertenece a la imagen para compartir, sin embargo, esta característica se deja para trabajo futuro.

Capítulo 7

Conclusiones y trabajo futuro

Conclusiones

Obtuvimos un servicio de calidad que puede compararse con las aplicaciones de vanguardia que brindan servicios similares (*MSN*, *Skype*, etc.). Este servicio combina muchas de las tecnologías utilizadas en la actualidad (comunicación inalámbrica, SIP, sistemas distribuidos, cómputo ubicuo, cómputo móvil, etc.). La heterogeneidad que alcanza nuestro trabajo no sólo cubre la parte de dispositivos (PC convencionales, *Laptops*, PDA y *Smartphones*) con sistemas embebidos (*Symbian* y *Windows Mobile*) y no enbebidos (*Windows* y *Linux*), sino también la parte del tipo de conexión (ya que soporta medios alámbricos e inalámbricos). Además alcanzamos la portabilidad deseada, gracias a que el sistema podría portarse en otros dispositivos (*Mac*, *PSP*, entre otros) puesto que utilizamos las herramientas estándar de desarrollo tanto de *Python* como de *PJSUA*.

La instalación de *PJSUA* no es sencilla y para realizar las modificaciones tanto para *Symbian* como para *Windows Mobile* y para crear el **intermediario** que comunicara a *PJSUA* con un servidor creado en *Python*, se requiere de amplios conocimientos tanto del lenguaje *Python* como del funcionamiento de la biblioteca *PJSUA*, ya que el hecho de combinar dos lenguajes de programación, resulta complicado y más cuando las funciones no fueron creadas por nosotros (como es el caso de las de la biblioteca), sino que tuvimos que modificar las ya diseñadas.

Comprender la biblioteca *PJSUA* fue un trabajo difícil, debido a que hubo que probar el funcionamiento de sus servicios y la interoperabilidad entre plataformas soportadas. Finalmente, tuvimos que comprender las funciones a nivel código, para así poder combinarlas con las funciones que desarrollamos en el lenguaje *Python* y lograr un mayor nivel de portabilidad para nuestra aplicación.

Mediante nuestro trabajo es posible enviar mensajes de texto, intercambiar archivos y realizar llamadas VoIP entre PC, *Laptop*, PDA y *Smartphones*, además es posible que el usuario tenga su sesión activa en más de un dispositivo, otorgándole así la capacidad de dar preferencias a la ejecución de sus servicios en los dispositivos que desee, esto brinda un grado más alto de comodidad al usuario puesto que vuelve

más flexible nuestra aplicación.

Para gestionar el mecanismo de selección de preferencias, tuvimos que incorporar el uso de una base de datos, y el hecho de manipular el cambio de preferencias en tiempo de ejecución, no fue tarea fácil, ya que el usuario tiene la capacidad de realizar varios cambios de preferencias simultáneamente debido a que puede realizarlos desde sus diferentes dispositivos de manera concurrente.

Otro de los puntos importantes fue la creación de un mecanismo de comunicación por eventos, ya que los eventos se manejan de forma distinta en los diferentes sistemas en los que trabajabamos, por lo tanto, tuvimos que crear un mecanismo para que los mensajes fueran compatibles entre las diferentes plataformas (*Windows*, *Linux*, *Symbian S60* y *Windows Mobile*), además, del mecanismo para generar los eventos en los dispositivos.

Tuvimos que aprender el funcionamiento del protocolo SIP, ya que es el que tomamos por medio de la biblioteca *PJSUA* y nos permite realizar las llamadas VoIP. Además de comprender la manera en que se manejan los mensajes de texto en SIP, puesto que éstos son los que utilizamos para crear el **intermediario** que comunica las aplicaciones de *C* con las de *Python*.

A lo largo del desarrollo de la interfaz tuvimos que enfrentarnos con el problema de generar interfaces gráficas que pudieran ejecutarse en hilos diferentes, y este problema fue resuelto de manera satisfactoria, ya que al final permitimos realizar varias operaciones simultáneamente (e.g., tener varias ventanas de *chat* abiertas, o enviar varios archivos simultáneamente).

Como pudimos observar en los resultados nuestro trabajo tiene las características básicas de las aplicaciones más populares que proporcionan servicios similares y aunque no cumple con el cien por ciento de las características, tiene otras innovadoras y de gran funcionalidad. Podemos concluir en cuanto a SEGEUS, que su funcionalidad puede resultar atractiva en dos ámbitos, a nivel comercial, ya que resultaría muy rentable que los usuarios pudieran tener acceso a la interoperabilidad brindada por SEGEUS, y a nivel investigación puesto que el trabajo que puede surgir a raíz de SEGEUS puede ser muy amplio

Trabajo futuro

La biblioteca *PJSUA* está en constante actualización y ha añadido un módulo de **llamada múltiple** (varios usuarios en una misma llamada) o conferencia de voz en sus últimas versiones. Sin embargo lograr esto implicaría conocer más a fondo la biblioteca y su implementación en *Windows Mobile* y *Symbian* sería mucho más complicada, es por eso que el servicio de conferencia de voz se ha considerado como trabajo futuro.

Durante el posgrado desarrollamos envío de imágenes en tiempo real de un *Smartphone Symbian* a una computadora utilizando *Python*. Con la biblioteca *PJSUA* hemos desarrollado el medio para enviar sonido y comunicar esa aplicación con *Python*. Si juntáramos ambos aplicaciones, podríamos crear un servicio de envío de video en

tiempo real. Sin embargo el envío de imágenes desde un PDA o de una PC y *Laptop* es más complicado, y debe crearse una interfaz compatible para las diferentes plataformas, dicha interfaz permitiría crear el servicio de **videollamada**, pero el tiempo que se requiere para hacer la interfaz compatible es demasiado, por ello se consideró este servicio como trabajo futuro también.

Supongamos que *UsuarioX* (con su sesión activa en su PC y en su *Smartphone*) tiene una ventana de *chat* abierta con *UsuarioY*, entonces *UsuarioX* decide cambiar las preferencias para que los mensajes de texto sean recibidos en su *Smartphone*, el siguiente mensaje que escriba *UsuarioY* en su ventana de *chat* será enviado al *Smartphone* de *UsuarioX*, esto brinda un nivel de **transparencia** al servicio de mensajes de texto. Un trabajo futuro sería realizar lo mismo con una llamada VoIP, sin embargo, este intercambio de dispositivo en una llamada activa, implicaría un gran trabajo de investigación acerca de la biblioteca *PJSUA* y la redirección de una llamada en tiempo de ejecución. Por lo tanto, también proponemos esta función de **transparencia de llamadas VoIP** como trabajo futuro [60].

Cuando un usuario inicia sesión en más de un dispositivo, es evidente que no puede estar presencialmente frente a todos sus dispositivos de manera simultánea. Por lo tanto, se propone también como trabajo futuro un **servicio no presencial**, que permita a los dispositivos redireccionar las peticiones de servicio en un determinado tiempo a otro dispositivo. O bien un **servicio de recepción de archivos basado en contexto** [61], que permita al usuario elegir el tipo de archivos que desee recibir (por ejemplo imágenes), y sean recibidos de manera automática sin que el usuario se encuentre presente.

Y finalmente el servicio de **almacenamiento temporal** que permitiría al usuario recibir mensajes y archivos mientras no se encuentre activa ninguna sesión y una vez que inicie sesión, el servidor pueda notificarle los mensajes recibidos y realizar la transferencia de los archivos almacenados.

Estas ideas que aportamos como trabajo futuro, requieren de un gran trabajo de investigación, sin embargo, de acuerdo a nuestra experiencia, podemos decir que las ideas son factibles en su totalidad para ser incorporadas a SEGEUS.

Bibliografía

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley and E. Schooler, *SIP: Session Initiation Protocol (RFC 3261)*, The Internet Society, June 2002.

[2] A. S. Tanenbaum and M. Van Steen, *Distributed Systems: Principles and Paradigms*, Prentice-Hall, 1995. Edición en castellano: Sistemas Operativos Distribuidos. Prentice-Hall, 1996.

[3] George Coulouris, *Distributed Systems: Concepts and Design*, Addison-Wesley, 3rd Edition, August 2000.

[4] Yongwan Park 1 and Fumiyuki Adachi 2, *Overview of Mobile Communication*, 1 Department of Information and Communication Engineering, Yeungnam University, Korea and 2 Graduate School of Engineering, Tohoku University, Japan, Springer, 2007.

[5] D. Chaparro González, *Computación ubicua*, Licenciatura, Universidad Rey Juan Carlos, Escuela Superior de Ciencias Experimentales y Tecnología, Junio 2003.

[6] J. Felix Basterretche, *Dispositivos Móviles*, Trabajo de Adscripción, Universidad Nacional del Nordeste Facultad de Ciencias Exactas, Naturales y Agrimensura, Argentina, 2007

[7] Klaus Rindtorff and Martin Welsch, *Portales Internet para dispositivos de Computación Móvil y Pervasiva*, Novatica, Vol. 153, No. 153. Computación ubicua, pp. 26-30, ATI, Septiembre-Octubre 2001.

[8] Baijian Yang, Pei Zheng and Lionel M. Ni, *Professional Microsoft Smartphone Programming*, Wiley Publishing, Inc., 2007.

[9] Gwenaël Le Bodic, *Mobile Messaging Technologies and Services SMS, EMS and MMS*, John Wiley and Sons Ltd, 2003.

[10] Henry Sinnreich and Alan B. Johnston, *Internet Communications Using SIP. Delivering VoIP and Multimedia Services with Session Initiation Protocol*, Wiley Publishing, Inc., Second Edition, 2006.

[11] Robert Sparks, *SIP Basics and Beyond*, ACM QUEUE, March 2007.

[12] Gonzalo Camarillo, *SIP Demystified*, McGraw-Hill, 2002.

[13] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications (RFC 1889)*, January 1996.

[14] Edward Oguejiofor, Philippe Bazot, Bruno Georges, Rebecca Huber, Callum Jackson, Jochen Kappel, Cameron Martin, Bala S. Subramanian and Abhijit Sur, *Developing SIP and IP Multimedia Subsystem (IMS) Applications*, Redbooks, February 2007.

[15] John F. Buford, Heather Yu, Eng Keong Lua, *P2P Networking and Applications*, Elsevier Inc., 2009.

[16] Tomasz Imielinsky, *Mobile computing: DataMan project perspective*, Computer Science Departament, Rutgers University, USA, 1996.

[17] Wei-Meng Lee, *Windows XP Unwired*, O'Reilly, August 2003.

[18] Carl Eklund, Roger B. Marks, Kenneth L. Stanwood and Stanley Wang, *IEEE Standard 802.16: A Technical Overview of the WirelessMAN Air Interface for Broadband Wireless Access*, IEEE, 2002

[19] A. Ganz, Z. Ganz, K. Wongthavarawat, *Multimedia Wireless Networks, Technologies, Standards and QoS*, Prentice Hall, 2004.

[20] M. Cardei, *Overview over the Bluetooth technology*, University of Minnesota, 2002.

[21] Antonio Cuevas, Carlos García, José Ignacio Moreno, Ignacio Soto, *Los pilares de las redes 4G: QoS, AAA y Movilidad*, Departamento de Ingeniería Telemática, Universidad Carlos III de Madrid, España.

[22] David Coleman, et al., *Groupware*, Prentice-Hall, Junio 1997.

[23] C.A. ElliS, S.J. GibbS and G.L. Rein, *Groupware Some Issues and Experiences*, Communications of the ACM, Vo1.34, No. 1, January 1991.

[24] Celeste Campo Vázquez, *Agentes móviles en Computación Ubicua*, Departamento de Ingeniería Telemática, Universidad Carlos III de Madrid, España,

[25] Mark Weiser, *The computer for the 21st century*. Scientific American, 9 1991.

[26] David L. Mills, *Network Time Protocol Version 4 Reference and Implementation Guide*, Technical Report, University of Delaware, 2006.

[27] Siu Man Lui and Sai Ho Kwok, *Interoperability of Peer-To-Peer File Sharing Protocols*, ACM SIGecom Exchanges, Vol. 3, No. 3, pp. 25-33, August 2001.

[28] Paul Dourish and Victoria Bellotti, *Awareness and Coordination in Shared Workspaces*, ACM, Cambridge, 1992.

[29] Vijay Gehlot and Anush Hayrapetyan, *A Formalized and Validated Executable Model of the SIP-Based Presence Protocol for Mobile Applications*, Center of Excellence in Enterprise Technology, Department of Computing Sciences, Villanova University, Villanova, USA, ACM, 2007.

[30] Rodrigo Vega García, *Sistema de comunicación con niveles de servicio basado en SIP para ambientes heterogéneos*, Maestría, Departamento de Computación, Centro de Investigación y Estudios Avanzados del Instituto Politécnico Nacional, México, Diciembre 2008.

[31] M. Agust Skulason, *Mobile devices as Web service providers*, Master Tesis, Technical University of Denmark, Informatics and Mathematical Modelling, September 2008.

[32] Robert W. Sebesta, *Concepts of Programming Lenguages*, Prentice Hall / Pearson, 2003.

[33] Daryl Wilding-McBridem, textitJava Development on PDAs: Building Applications for PocketPC and Palm Devices, Addison Wesley, June 2003.

[34] Li Gong, Scot t Oaks and Bernard Traversat, *JXTATM in a Nut shell*, O'Reilly and Associates, Inc., 2002.

[35] T. Socolofsky and C. Kale, *A TCP/IP Tutorial (RFC 1180)*, January 1991.

[36] Fredrik Lundh and Matthew Ellis, *Python Imaging Library Overview*, Tutorial, March 2002

[37] Fredrik Lundh, *An introduction to Tkinter*, Tutorial, 1999.

[38] John W. Shipman, *Tkinter reference: a GUI for Python*, New Mexico Tech, Computer Center, 2008.

[39] Nokia, *PyS60 Library Reference Release 1.4.1 final*, Nokia Corporation, July 2007.

[40] Jurgen Scheible and Ville Tuulos, *Mobile Python Rapid Prototyping of Applications on the Mobile Platform*, John Wiley and Sons Ltd, 2007.

[41] Daniel Ashbrook, *Python for Nokia S60*, Tutorial, 2008

[42] Hun Jeong Kang and Zhi-Li Zhang, *SIP-based VoIP Traffic Behavior Profiling and Its Applications*, University of Minnesota, ACM, 2007.

[43] M. Handley and V. Jacobson, *SDP: Session Description Protocol (RFC 2327)*, The Internet Society, April 1998.

[44] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications (RFC 3550)*, The Internet Society, July 2003.

[45] T. Friedman, R. Caceres and A. Clark, *RTP Control Protocol Extended Reports (RTCP XR) (RFC 3611)*, The Internet Society, November 2003.

[46] A. B. Roach, *Session Initiation Protocol (SIP)-Specific Event Notification*, The Internet Society, 2002.

[47] Erich M. Nahum, John Tracey, and Charles P. Wright, *Evaluating SIP Server Performance*, IBM T.J. Watson Research Center, ACM, June 2007.

[48] Richard Gayraud and Olivier Jacques, *SIPp reference documentation*, 2004.

[49] *PJLIB Reference Manual*, generated by Doxygen, 2005.

[50] *Transmission Control Protocol (RFC 793)*, Information Sciences Institute, University of Southern California, September 1981.

[51] J. Postel, *User Datagram Protocol (RFC 768)*, August 1980

[52] Henning Schulzrinne 1 and Elin Wedlund 2, *Application-Layer Mobility Using SIP*, 1 Dept, of Computer Science, Columbia University and 2 Netinsight, Mobile Computing and Communications Review, Volume 4, Number 3.

[53] Andy Wigley, Daniel Moth and Peter Foot, *Microsoft Mobile Development Handbook*, Microsoft Press, 2007.

[54] Samir Saklikar and Subir Saha, *Identity Federation for VoIP-based Services*, Motorola Labs, India, ACM, 2007.

[55] Matthew MacDonald, *Peer-to-Peer with VB .NET*, Apress, 2003.

[56] Accenture, *Arquitectura Orientada a Servicios (SOA)*, Centro de Alto Rendimiento de Accenture (CAR), 2008

[57] Microsoft, *La Arquitectura Orientada a Servicios (SOA) de Microsoft aplicada al mundo real*, Microsoft Corporation, Diciembre 2006.

[58] Arpit Mehta, Andrea G. Forte and Henning Schulzrinne, *Using Conference Servers for SIP-based Vertical Handoff between IP and Cellular Networks*, Columbia University, ACM, 2008.

[59] Thomas A. Finholt, Elena Rocco, Daniel Bree and Nishant Jain, *NotMeeting: A field trial of NetMeeting in a geographically distributed organization*, SIGGROUP Bulletin, Vol 20, No. 1, April 1999.

[60] Pablo Vidales, Niklas Kirschnick, Blazej Lewcio, Frank Steuer, Marcel Wältermann and Sebastian Möller, *Mobisense Testbed: Merging User Perception and Network Performance*, ACM, March 2008.

[61] Jinwoo Park 1, Jong-Kwon Lee 1, YoungSang Paik 1, MyungChul Lee 1 and Chandra Narayanaswami 2, *Device Context Discovery System for Context-aware Services in Ubiquitous Device Collaboration Environment*, 1 Ubiquitous Computing Lab, IBM Korea, Seoul, Republic of Korea and 2 IBM T. J. Watson Research Center, NY, U.S.A., IEEE, 2006.

Referencias electrónicas

[62] *aMSN Features* [en línea], August 2006 [citado Septiembre 2009], http://amsn.sourceforge.net/wiki/tiki-index.php?page=aMSN+Features

[63] Skype Limited, *Skype features* [en línea], 2009 [citado Febrero 2009], http://www.skype.com/help/guides/skypeformac/skypefeatures/

[64] *Nimbuzz blog* [en línea], 2009 [citado Febrero 2009], http://blog.nimbuzz.com/category/features/

[65] Sergio Galicia, *Sergio Galicia* [en línea], 2009 [citado Agosto 2009], http://sergiogalicia.com/telcel-noticias-2222

[66] WordReference, *WordReference.com Diccionario de la lengua española. Heterogeneidad* [en línea], 2005 [citado Febrero 2009], http://www.wordreference.com/definicion/heterogeneidad

[67] Livemessenger.net, *Windows Live Messenger features* [en línea], 2006 [citado Abril 2009], http://www.livemessenger.net/features/

[68] Python Software Foundation, *Python Programming Language Official Website* [en línea], 1990-2009 [citado Noviembre 2008],

[69] Richard Gayraud, Olivier Jacques, *SIPp* [en línea], 2004 [citado Mayo 2009], http://sipp.sourceforge.net/

[70] *Features* [en línea], August 2009 [citado Septiembre 2009], http://wiki.ekiga.org/index.php/Features

[71] Benny Prijono, *pjsua Manual Page*, 2007 [citado en Diciembre 2008], http://www.pjsip.org/pjsua.htm http://www.python.org/

[72] MySQL AB and Sun Microsystems, Inc., *MySQL 5.1 Reference Manual* [en línea], 2009 [citado Marzo 2009], http://dev.mysql.com/doc/refman/5.1/en/

[73] MySQL AB and Sun Microsystems, Inc., *MySQL Downloads* [en línea], 2009 [citado Marzo 2009],
http://dev.mysql.com/downloads/

[74] SourceForge, Inc., *Python Windows CE port* [en línea], 2004 [citado Julio 2009],
http://sourceforge.net/projects/pythonce/files/

[75] *How to install Tkinter* [en línea], 2007 [citado Junio 2009],
http://tkinter.unpythonic.net/wiki/How_to_install_Tkinter

[76] Microsoft Corporation, *NetMeeting* [en línea], 2009 [citado 21 de Septiembre 2009],
http://technet.microsoft.com/en-us/library/cc743213.aspx,

[77] Ziff Davis Publishing Holdings Inc., *Pidgin Review* [en línea], 2001-2009 [citado Agosto 2009],
http://www.extremetech.com/article2/0,2845,2142654,00.asp

[78] Pidgin Contributors., *Pidgin* [en línea], [citado Agosto 2009],
http://pidgin.im/

Apéndice A

Lista de acrónimos

0G Generación cero de la tecnología celular

1G Primera generación de la tecnología celular

2G Segunda generación de la tecnología celular

2.5G Generación intermedia entre la 2G y la 3G

3G Tercera generación de la tecnología celular

3.5G Generación actual de la tecnología celular

4G Cuarta generación de la tecnología celular

API *Application Programming Interface*

BD Base de Datos

C/S Cliente - Servidor

CPN Red de Petri con Color

G Generación

GPL Licencia Pública General

GPRS Servicio General de Radio por Paquetes

GPS Sistema de Posición Global

GSM Sistema Global para comunicaciones Móviles

GUI	Interfaz Gráfica de Usuario
IEEE	Instituto de Ingenieros Eléctricos y Electrónicos
IETF	Fuerza de Trabajo de Ingenieros de Internet
IM	Mensajería Instantánea
IP	Protocolo de Internet
IrDA	Infrared Data Association
LP	Lenguaje de Programación
Mbps	Megabytes por segundo
MSN	Messenger
OSI	interconexión de sistemas abiertos
P2P	Punto a punto
PC	Computadora Personal
PDA	Asistente Personal Digital
PyS60	Python para Symbian S60
RTCP	Protocolo de Control de Transporte en Tiempo-Real
RTP	Protocolo de Transporte en Tiempo Real
SCIP	Protocolo Simple de Invitación a Conferencia
SDP	Protocolo de Descripción de Sesiones
SIP	Protocolo de Inicio de Sesión
SR	Sesión Real
UA	Agente Usuario
URI	Identificador de Recurso Uniforme
VoIP	Voz sobre IP
WIMAX	Worldwide Interoperability for Microwave Access