

ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

“DISEÑO Y DESARROLLO DE UNA APLICACIÓN P2P DE MENSAJERÍA
PARA LA ESPOL USANDO TECNOLOGÍA JXTA”

TESIS DE GRADO

Previa a la obtención del Título de:

INGENIERO EN COMPUTACIÓN

ESPECIALIZACIÓN SISTEMAS DE INFORMACIÓN

Presentado por

Xavier Fernando Calle Peña

Vanessa Inés Cedeño Mieles

Guayaquil – Ecuador

2007

AGRADECIMIENTO

Agradecemos a la ING.
CRISTINA ABAD
Directora de tesis, por
su ayuda y colaboración
para la realización de
este trabajo.

DEDICATORIA

A Dios

A nuestros padres

TRIBUNAL DE GRADUACION

Ing. Holger Cevallos

Subdecano de la FIEC

Ing. Cristina Abad

Directora de Tesis

Ing. Xavier Ochoa

Miembro del Tribunal

Ing. Katherine Chiluiza

Miembro del Tribunal

DECLARACIÓN EXPRESA

“La responsabilidad del contenido de esta Tesis de Grado, nos corresponde exclusivamente; y el patrimonio intelectual de la misma, a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL”

(Reglamento de Graduación de la ESPOL).

Xavier Fernando Calle Peña

Vanessa Inés Cedeño Mieles

RESUMEN

Actualmente en los laboratorios de las distintas facultades de la ESPOL existe el reglamento que especifica que en el interior de las salas de los centro de cómputos se prohíbe el uso de cualquier programa o sitio de comunicación en línea. Esta regla fue la motivación para la realización de este proyecto. Queremos otorgar una aplicación de mensajería alternativa que les permita a los estudiantes poder comunicarse con sus compañeros politécnicos. Decidimos diseñar y desarrollar una aplicación p2p, que en inglés quiere decir peer to peer y en español significa par a par la cual no depende de un servidor central. Para esto utilizamos la tecnología JXTA, que viene de “yuxtapuesto”, de Java Sun que es un grupo de protocolos que permite a cualquier dispositivo conectado a la red comunicarse y colaborar de una manera p2p. JXTA es independiente de cualquier lenguaje de programación y de implementaciones.

La red p2p se basa en una serie de nodos que se comportan a la vez como clientes y como servidores de los demás nodos de la red. Cualquier nodo puede iniciar o completar una transacción compatible. Los nodos pueden diferir en configuración local, velocidad de proceso, ancho de banda de su conexión a la red y capacidad de almacenamiento. Lo que otorga las ventajas de tolerancia a fallos, escalabilidad, costo de implementación, mejor

rendimiento y disponibilidad de recursos. Aunque actualmente existen sistemas de mensajería p2p basados en tecnología JXTA disponibles para bajar de Internet como MyJXTA y JIM (JXTA Instant Messenger). Estos otorgan la funcionalidad de un chat p2p pero no cumplen los requerimientos para ser implementados en instituciones académicas ya que cualquiera puede acceder a ellos. Por lo que el uso de nuestra aplicación será restringido para estudiantes de la ESPOL únicamente, validando el ingreso con la información de las cuentas de usuarios, trabajadores y profesores de la base de datos del CSI, ya que sino el sistema podría degenerarse si algunos usuarios deciden hacerse pasar por otros.

El presente proyecto tiene como alcances:

- El uso de la tecnología JXTA
- Chat de 1 a 1
- Chat entre múltiples usuarios simultáneamente
- Creación de grupos
- Agrupación de usuarios
- Cartelera comunal
- Envío de archivos
- Estado de cada usuario conectado
- Búsqueda de usuarios conectados

ÍNDICE GENERAL

	Pág.
ÍNDICE GENERAL	VIII
ÍNDICE DE ABREVIATURAS	XIV
ÍNDICE DE FIGURAS	XV
ÍNDICE DE TABLAS	XVII
INTRODUCCIÓN	1
1. CAPÍTULO 1: PLANTEAMIENTO DEL PROBLEMA	
1.1 Definición del problema.....	2
1.2 Justificaciones de diseño.....	3
1.2.1 Sistemas de mensajería existentes.....	3
1.2.2 Plataforma .Net de Microsoft vs. JXTA.....	4
1.2.3 Ventajas de JXTA.....	5
2. CAPÍTULO 2: REDES PAR-A-PAR	
2.1 Definición de redes Par-a-Par.....	6
2.2 Tipos de redes Par-a-Par.....	8
2.2.1 Redes centralizadas.....	8
2.2.2 Redes descentralizadas.....	9
2.2.3 Redes P2P Híbridas.....	10
2.3 Funcionamiento de redes Par-a-Par.....	11
2.4 Características de las redes Par-a-Par.....	13

2.5	Principales problemas de las redes Par-a-Par.....	15
2.6	Utilización de las aplicaciones Par-a-Par.....	16
2.7	Historia de las aplicaciones Par-a-Par.....	19
2.7.1	Mensajería instantánea.....	19
2.7.2	Transferencia de archivos.....	20
2.7.3	Computación distribuida.....	22
2.8	Problemática actual de las redes Par-a-Par.....	23
3.	CAPÍTULO 3: PLATAFORMA JXTA	
3.1	Introducción a proyecto JXTA.....	25
3.2	JXTA.....	26
3.3	Arquitectura de JXTA™.....	28
3.4	Conceptos de JXTA.....	32
3.4.1	<i>Peers</i>	32
3.4.2	Grupos de <i>Peers</i>	33
3.4.3	Servicios de red.....	35
3.4.4	Módulos.....	37
3.4.5	<i>Pipes</i>	39
3.4.6	Mensajes.....	41
3.4.7	<i>Advertisements</i> (Publicaciones).....	42
3.4.8	Seguridad.....	44
3.4.9	<i>ID's</i>	46
3.5	Protocolos de JXTA.....	47

4. CAPÍTULO 4: ANÁLISIS Y DISEÑO DEL PROYECTO

4.1	Especificaciones.....	50
4.1.1	Requerimientos funcionales.....	50
4.1.2	Requerimientos no funcionales.....	50
4.2	Diseño conceptual.....	51
4.2.1	Visión del sistema.....	51
4.2.2	Objetivos.....	51
4.2.3	Alcances.....	52
4.2.4	Restricciones.....	55
4.3	Diseño Lógico.....	56
4.3.1	Casos de uso.....	56
4.3.2	Descripción de casos de uso.....	57
4.3.3	Lista de escenarios.....	64
4.3.4	Especificación de escenarios.....	65
4.3.5	Especificación de actores.....	65
4.3.6	Diagramas de casos de uso.....	66
4.3.7	Diagrama UML.....	68

5. CAPÍTULO 5: IMPLEMENTACIÓN DE PROYECTO

5.1	Uso de NetBeans IDE 5.0.....	69
5.2	Servidor Proxy.....	69
5.2.1	Configuración de JXTA para comunicarse vía HTTP Proxy Servers.....	72

5.3	Autenticación.....	72
5.3.1	Servicio Web.....	73
5.4	Unirse al Net Peer Group.....	75
5.5	Creación del grupo ESPOL.....	76
5.6	Unirse al grupo ESPOL.....	77
5.7	Servicio de membresías y credenciales.....	79
5.8	Descubrimiento de <i>Peers</i>	81
5.9	Creación de <i>Pipes</i>	83
5.10	Pipe de entrada de mensajes.....	85
5.11	Pipe de salida de mensajes.....	85
5.12	Pipe de entrada de archivos.....	85
5.13	Pipe de salida de archivos.....	86
5.14	Creando el servicio de presencia.....	86
5.14.1	El <i>Advertisement</i> de presencia.....	87
5.14.2	Publicar el <i>Advertisement</i> de presencia.....	88
5.14.3	Descubrimiento de <i>Advertisement</i> de presencia...88	
5.14.4	Actualizar el <i>Advertisement</i> de presencia.....	89
5.15	Creando el servicio de cartelera comunal.....	89
5.15.1	El <i>Advertisement</i> de cartelera.....	90
5.15.2	Implementación del <i>Advertisement</i> de cartelera...91	
5.15.3	Publicar el <i>Advertisement</i> de cartelera.....	91
5.15.4	Descubrimiento de <i>Advertisement</i> de Lista de	

Contactos.....	91
5.16 Lista de Contactos.....	91
5.17 Búsqueda de contactos por usuario.....	94
5.18 Comunicación de uno a muchos.....	94
6. CAPÍTULO 6: PRUEBAS DE ESCALABILIDAD	
6.1 Pruebas de carga/ pruebas de estrés.....	98
6.1.1 Prueba de carga <i>Discovery</i>	98
6.1.2 TCP <i>Relay</i>	101
6.2 Pruebas de Tiempos de Ida y Vuelta.....	103
6.2.1 Pipes <i>Unicast</i>	103
6.3 Pruebas independientes.....	106
6.3.1 Tiempo de <i>parsing</i> de un XML.....	106
6.3.2 Tiempo a conectarse a un peer <i>Rendezvous</i>	108
6.3.3 Tiempo a unirse a una red JXTA.....	109
7. CAPÍTULO 7: PRUEBAS DE USUARIO	
7.1 Usabilidad.....	111
7.2 Pruebas de usuario.....	112
7.3 Atributos de usabilidad.....	114
7.4 Evaluación de pruebas.....	120
7.5 Mejoras en el proyecto.....	124
CONCLUSIONES Y RECOMENDACIONES.....	128
APÉNDICES.....	133

APÉNDICE A: GLOSARIO.....	134
APÉNDICE B: FLUJO DE VENTANAS.....	231
BIBLIOGRAFÍA.....	250

ÍNDICE DE ABREVIATURAS FRECUENTEMENTE USADAS

API	Application Programming Interface
DNS	Domain Name System
HTTP	Hypertext Transmisión Protocol
ISP	Internet Service Provider
J2SE	Java 2 Platform, Standard Edition
JXTA	De “yuxtapuesto”, una plataforma peer-to-peer
MP3	MPEG-1 Audio Layer-3
NAT	Network Address Translation
P2P	Peer-to-Peer, Par-a-Par
SMTP	Simple Mail Transfer Protocol
SSL	Secure Socket Layer
TCP	Transmisión Control Protocol
TTL	Time to Live
URI	Uniform Resource Identifier
URN	Uniform Resource Name
XML	eXtensible Markup Language

ÍNDICE DE FIGURAS

		Pág.
Figura 2.1	Arquitectura Cliente Servidor.....	21
Figura 2.2	Arquitectura Par-a-Par.....	22
Figura 2.3	Redes Centralizadas.....	23
Figura 2.4	Redes Descentralizadas.....	24
Figura 2.5	Redes P2P Híbridas.....	25
Figura 2.6	Utilización de las Aplicaciones Par-a-Par.....	30
Figura 2.7	Gráfico de Utilización de las Aplicaciones Par-a-Par.....	31
Figura 2.8	Gráfico de Utilización de las Aplicaciones Par-a-Par.....	31
Figura 3.1	Peers en la Web Expandida.....	39
Figura 3.2	Pipes.....	49
Figura 3.3	Protocolos de JXTA.....	56
Figura 4.1	Diagrama de Casos de Uso (parte 1)	72
Figura 4.2	Diagrama de Casos de Uso (parte 2)	73
Figura 4.3	Diagrama UML.....	74
Figura 5.1	Comunicación de Peers a través de Proxies, Firewalls o NATS.....	77
Figura 5.2	Arquitectura Genérica de un Servicio Web.....	79
Figura 5.3	Creación del NetPeerGroup.....	81
Figura 5.4	Creación del Grupo ESPOL.....	82
Figura 5.5	Actualización del Grupo ESPOL.....	83

Figura 5.6	Descubrimiento de Peers.....	86
Figura 5.7	Creación de Pipes.....	88
Figura 5.8	El usuario 1 se conecta al usuario 2.....	89
Figura 5.9	Contactos agregados a una conversación.....	96
Figura 5.10	Envío de mensajes desde el nodo invocador.....	96
Figura 5.11	Envío de mensajes desde un nodo invocado.....	97
Figura 6.1	Prueba de carga Discovery a un peer Rendezvous.....	101
Figura 6.2	Prueba de carga Discovery a un peer Rendezvous por TCP relay.....	103
Figura 6.3	Pruebas de latencia de Pipes Unicast entre Peers.....	105
Figura 6.4	Pruebas de latencia de Pipes Unicast entre Rendezvous y Peers.....	106
Figura 6.5	Pruebas de Tiempo de Parsing de un XML.....	108
Figura 6.6	Pruebas de Tiempo a conectarse a un peer rendezvous.....	109
Figura 7.1	Análisis de Subtareas.....	127
Figura 7.2	Análisis de Encuesta.....	129

ÍNDICE DE TABLAS

		Pág.
Tabla I	Tabla de Prueba de carga Discovery a un peer Rendezvous...	100
Tabla II	Tabla de Prueba de carga Discovery a un peer Rendezvous por TCP relay.....	102
Tabla III	Tabla de Pruebas de latencia de Pipes Unicast entre Peers....	104
Tabla IV	Tabla de Pruebas de latencia de Pipes Unicast entre Rendezvous y Peers.....	106
Tabla V	Tabla de Pruebas de Tiempo de Parsing de un XML.....	107
Tabla VI	Tabla de Pruebas de Tiempo a conectarse a un peer Rendezvous.....	109
Tabla VII	Usuarios de prueba.....	112
Tabla VIII	Pruebas de Usabilidad 1.....	124
Tabla IX	Pruebas de Usabilidad 2.....	126
Tabla X	Mejoras en el Proyecto 1.....	130
Tabla XI	Mejoras en el Proyecto 2.....	132

INTRODUCCIÓN

Como la política de la ESPOL impide la utilización de aplicaciones de comunicación en las salas de cómputo, surge la necesidad de un servicio de mensajería cuyo acceso pueda ser restringido a los usuarios de la Institución para que puedan comunicarse entre ellos únicamente, mas no con usuarios externos. Por esta razón, hemos desarrollado una aplicación de mensajería instantánea, utilizando una arquitectura par-a-par (P2P), la cual es de fácil administración, escalable y flexible. Esta aplicación de mensajería basada en JXTA no depende de un servidor central (1). Los nodos de la red P2P se comportan a la vez como clientes y como servidores de los demás nodos. De esta manera, aún cuando uno o más nodos fallen, los otros pueden seguir comunicándose entre sí. Un esquema P2P proporciona varias ventajas con respecto a un enfoque cliente/servidor, como una muy buena tolerancia a fallos, gran escalabilidad, bajos costos de implementación, buen rendimiento y alta disponibilidad de recursos.

El documento está estructurado de la siguiente manera, en el capítulo 1 se plantea el problema, en el capítulo 2 se describen las redes par-a-par, en el capítulo 3 se describe la plataforma JXTA, el capítulo 4 presenta el análisis y diseño del proyecto, el capítulo 5 la implementación del proyecto, el capítulo 6 muestra pruebas de escalabilidad y el capítulo 7 las pruebas de usuario.

CAPÍTULO 1

PLANTEAMIENTO DEL PROBLEMA

1.1 Definición del problema

Las políticas en la ESPOL especifican que en el interior de las salas de los centros de cómputo se prohíbe el uso de cualquier programa o sitio de comunicación en línea, como por ejemplo MSN Messenger, Yahoo Messenger, etc. Este tipo de políticas buscan evitar que los estudiantes se distraigan u ocupen el tiempo de máquina en actividades no relacionadas con la institución.

Lastimosamente, este tipo de políticas también disminuyen las posibilidades de colaboración productiva entre los usuarios de la institución. Por esta razón, surge la necesidad de una herramienta de mensajería que pueda ser restringida a los usuarios de una institución. Actualmente existen alternativas libres al sistema MSN Messenger de Microsoft o al Yahoo Messenger. Un ejemplo de un tipo de aplicación de mensajería que puede ser utilizada únicamente entre los usuarios de una institución es **Jabber**(2), el cual es un protocolo libre gestionado por Jabber Software Foundation basado en el estándar **XML** para

mensajería instantánea y el cual podría adaptarse para desarrollar una aplicación estrictamente universitaria. El problema de Jabber es que utiliza una arquitectura cliente/servidor, lo cual otorga un punto único de administración y de fallo. Surge entonces la necesidad de un sistema de mensajería que pueda ser restringido a usuarios universitarios y que no tenga un punto único de fallo y que sea de fácil administración. Para esto presentamos un diseño de un sistema de mensajería universitaria P2P.

1.2 Justificaciones de diseño

1.2.1 Sistemas de Mensajería P2P Existentes

Actualmente existen sistemas de mensajería basados en tecnología JXTA disponibles para bajar de Internet como **MyJXTA** (3), y **JIM** (JXTA Instant Messenger) (4). Estos otorgan la funcionalidad de un chat P2P pero no cumplen los requerimientos para ser implementados en instituciones académicas ya que cualquiera puede acceder a ellos y no buscan maximizar la colaboración académica. Por estas razones, hemos diseñado una aplicación de mensajería instantánea cuyo uso será restringido para que solamente permita conectarse a los estudiantes y demás miembros de una misma Universidad, validando el ingreso con la

información de las cuentas de usuarios de la base de datos de la Institución.

1.2.2 Plataforma .Net de Microsoft vs. JXTA

El **Framework** .NET de Microsoft así como JXTA provee una rica plataforma para la creación de aplicaciones P2P. Fundamentalmente JXTA y .NET tienen propósitos completamente diferentes, .Net se enfoca más en la arquitectura tradicional de cliente/servidor de entrega de servicios. Aunque la tecnología .NET puede formar las bases para una aplicación P2P, crear una completa solución P2P usando .NET requeriría que el desarrollador especifique todas las interacciones P2P del núcleo como por ejemplo descubrimientos de **peers**. Esta solución demandaría recrear todos los mecanismos que ya están definidos por los protocolos de JXTA (5). Por esto escogemos la tecnología JXTA, porque nos permite enfocarnos en los aspectos de diseño propios de la aplicación sin perder tiempo en detalles de mecanismos de comunicación P2P.

Otra razón muy importante, es que de esta manera trabajamos con una tecnología de código abierto que busca publicitar la

tecnología P2P y que ya posee usuarios en su red. Al trabajar con JXTA colaboramos con la red y la red así mismo con nosotros, mientras que si creamos una red propia no será compatible con ninguna otra más que con sí misma.

1.2.3 Ventajas de JXTA

- Proyecto JXTA de Sun (Juxtapose significa contraponer)
- **Interoperabilidad** entre diferentes sistemas P2P y comunidades
- Creación de aplicaciones P2P sólidas, sobre una base bien definida y común
- **Ubicuidad**, JXTA opera en cualquier dispositivo, teléfonos celulares, PDAs, sensores electrónicos, PCs, servidores

CAPÍTULO 2

REDES PAR-A-PAR

2.1 Definición de redes Par-a-Par

Una red par-a-par (P2P) es un ambiente de red donde un cliente puede comportarse como un servidor para otros clientes de la red. Esta concepción difiere radicalmente de la arquitectura cliente-servidor ya que no existen servidores u otros equipos de control del tráfico entre los miembros. Las Figuras 2.1 y 2.2 permiten apreciar las diferencias entre estos enfoques. Cabe recalcar la ausencia de un punto único de fallo en la red P2P de la Figura 2.2.

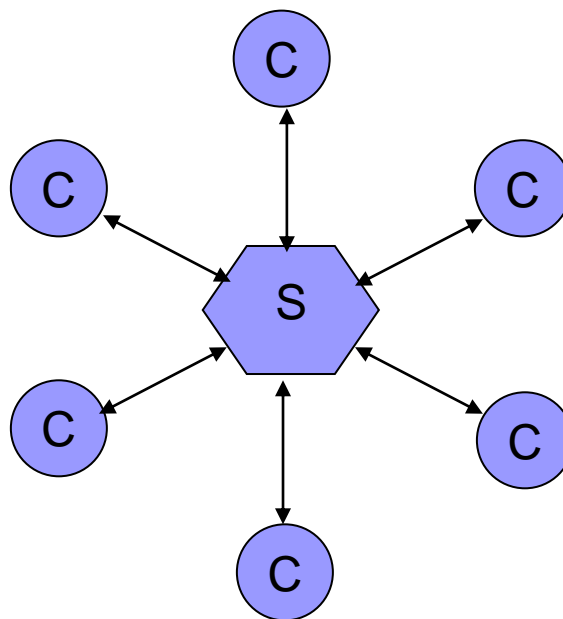


Fig. 2.1

Arquitectura Cliente-Servidor

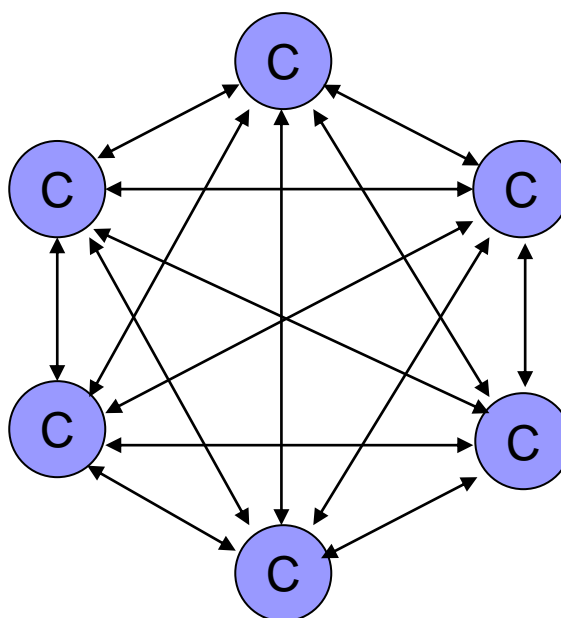


Fig. 2.2

Arquitectura Par-a-Par

2.2 Tipos de redes Par-a-Par

Según Santamaría Cabrera, Pablo y Santos López, Miguel (6), las redes P2P se clasifican en centralizadas, descentralizadas e híbridas.

2.2.1 Redes Centralizadas

Este tipo de redes utilizan un servidor central que gestiona todas las operaciones de intercambios, es decir, utilizan un computador central para localizar información y luego la transferencia de la información se realiza par-a-par. Este tipo de red tiene la ventaja de ser muy eficiente en cuanto a la localización de información, pero tiene un grado de vulnerabilidad alto, ya que cualquier ataque al computador central significa la anulación del servicio. Ejemplo de este tipo es **Napster** (7). La Figura 2.3 muestra una red de tipo centralizada.

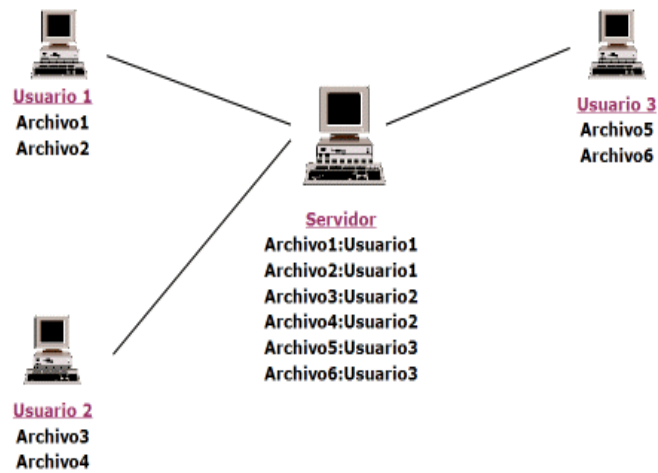


Fig. 2.3 (REF. 6)

Redes Centralizadas

2.2.2 Redes Descentralizadas

Nacieron para solventar la gran vulnerabilidad que tenía el modelo anterior, ya que no poseen un servidor central, pero esto trajo problemas para la gestión de búsqueda y transferencia, haciendo menos eficiente en este aspecto a este tipo de red. Ejemplo: **Gnutella** (8), y **Freenet** (9). La Figura 2.4 muestra una red de tipo descentralizada.

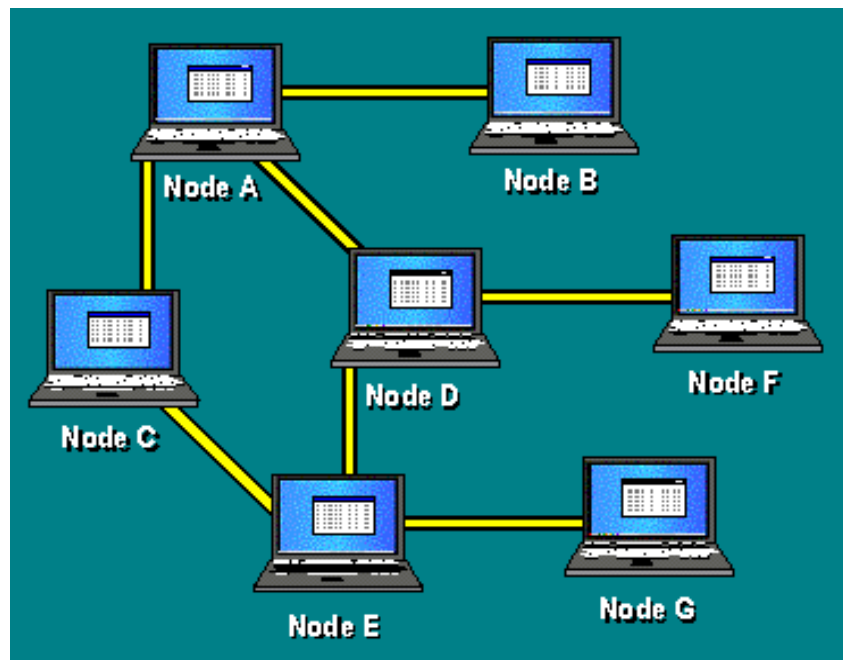


Fig. 2.4 (REF. 6)

Redes Descentralizadas

2.2.3 Redes P2P Híbridas

Se crearon para mejorar la localización de información y la transferencia de archivos y para hacer redes menos vulnerables. Este tipo de redes son una mezcla de los dos tipos anteriores, aprovechando sus ventajas y corrigiendo los errores. Consiste en transacciones a través de súper clientes (súper-peers) que actúan como nodos activos agilizando el funcionamiento de la red. Ejemplo: **Kazaa** (10), y **eDonkey**. La Figura 2.5 muestra una red de tipo híbrida.

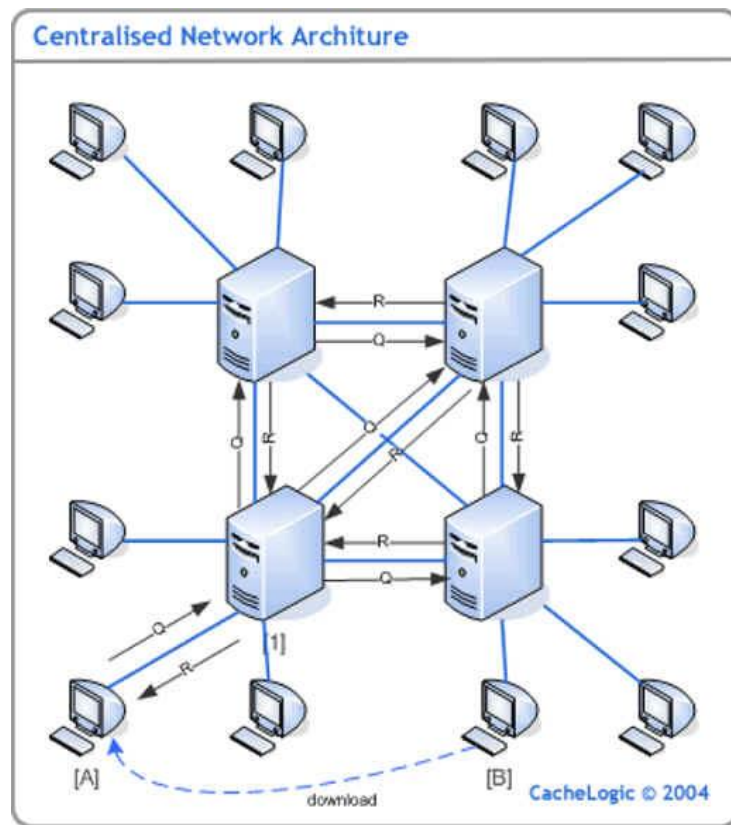


Fig. 2.5 (REF. 6)

Redes P2P Híbridas

2.3 Funcionamiento de redes Par-a-Par

El P2P se basa principalmente en la filosofía de que todos los usuarios deben compartir, conocida como filosofía P2P (11).

Debido a que la mayoría de los ordenadores domésticos no tienen una IP fija, sino que le es asignada por el proveedor, **ISP**, en el momento de conectarse a Internet, no pueden conectarse entre sí porque no saben

las direcciones que han de usar de antemano. La solución habitual es realizar una conexión a un servidor o servidores con dirección conocida o IP fija, que se encarga de mantener la relación de direcciones IP de los clientes de la red, de los demás servidores y habitualmente información adicional, como un índice de la información de que disponen los clientes.

Con las p2p los clientes obtienen esta información sobre el resto de la red por Inundación de Pedidos o también llamado **Multicast** (12), y pueden intercambiar información entre sí, ya sin intervención de los servidores. Lo que hacen es que al conectarse a la red, el peer que busca un archivo envía una consulta a todos sus vecinos en la red P2P, estos a su vez propagan el pedido a sus vecinos y así sucesivamente. Esto termina cuando se encuentra una respuesta o el mensaje excede un límite de propagaciones (**TTL**). El TTL es el tiempo de vida de los registros DNS cacheados en el cliente e influye en cuánto tiempo el cliente contacta a una IP. Es sencillo de implementar, muy efectivo y no hay punto único de fallo ni de censura. Pero la desventaja es que genera mucho tráfico innecesario y por lo tanto consume mucho ancho de banda, lo que limita la escalabilidad.

Pero JXTA aplica una solución la cual es utilizar una versión modificada de inundación de pedidos, llamada **rendezvous**. La red tiene dos tipos

de nodos, los normales y los súper nodos, también llamados súper *peers* o hubs. Los súper nodos tienen conexiones a muchos otros nodos e indexan los archivos de estos. Un cliente inunda a los súper nodos que conoce hasta que recibe una respuesta al pedido. Es decir, este esquema combina la inundación con un directorio centralizado, logrando lo mejor de ambas soluciones.

2.4 Características de las redes Par-a-Par

Según Abad, Cristina (13), las siguientes son las características más importantes de las redes P2P.

- **Descentralización:** Los sistemas descentralizados no tienen cuellos de botella ni puntos únicos de fallo. Los recursos son compartidos entre los miembros. El problema es que al no haber nadie a cargo, puede ser difícil administrar la red, o encontrar un recurso en la misma.
- **Escalabilidad masiva:** Significa que los cambios en las dimensiones de la red no afecten significativamente a su rendimiento. Las redes P2P necesitan ser grandes, contar cada vez con más usuarios. El problema es que mientras más grande es la red P2P, más complejos deben ser los algoritmos que rigen

su funcionamiento, ya que si no es así, perderíamos eficiencia y tendríamos un rendimiento no aceptable. Si se prolifera mucho este tipo de redes podrían llegar a su fin debido a una diversificación de usuarios y a cada red se conectarán muy pocos.

- **Auto-organización:** Al no existir una entidad central que rijan el comportamiento del sistema, los nodos deben auto-organizarse para su administración y manejo del contenido. Los algoritmos deben preocuparse de lograr escalabilidad, operación confiable, rendimiento satisfactorio y tolerancia a fallos, a partir de una colección de nodos no confiables con conexiones intermitentes. Algunos de los impactos de las comunicaciones P2P son debidos a la auto-organización.
- **Conexiones ad-hoc:** Los nodos se unen y salen de la red P2P en cualquier momento. Esto dificulta el poder proporcionar garantías de servicio.
- **Rendimiento:** La escala masiva potencial de estos sistemas puede tanto mejorar como empeorar el rendimiento, todo depende de los algoritmos que rigen a la red P2P.
- **Seguridad:** Es un problema, ya que no hay manera de confiar en un 100% en los otros participantes.

- **Tolerancia a fallos:** No hay un punto único de fallo. Sin embargo, las desconexiones de los nodos, congestiones en la red, o nodos con fallas pueden afectar la confiabilidad del sistema. La solución es usar una gran redundancia de archivos, rutas en la red y otros recursos esenciales.
- **Interoperabilidad:** Para que dos nodos se comuniquen, deben hablar el mismo lenguaje. Un nodo de Napster no puede comunicarse con uno de Gnutella.

En ciertos casos se desea, anonimidad, expresión libre y negación de responsabilidad. Estas características de libertad de expresión y privacidad son más fáciles de obtener en un sistema P2P que en uno centralizado.

2.5 Principales problemas de las redes Par-a-Par

El principal problema de las de las redes p2p es la ubicación eficiente de los nodos. Para solucionar esto Napster, utiliza un servidor central donde todas las peticiones son direccionadas a éste. El servidor central informa al usuario de donde puede bajar los datos que busca. El problema es el servidor central deja un único punto de fallo, por lo que cualquier ataque al servidor central anulará el sistema.

Gnutella utiliza inundación de mensajes para localizar los datos. De esta forma un mensaje es redirigido a todos los nodos del sistema mientras el nodo que contiene la información solicitada es encontrado. El problema es que el número de mensajes aumenta con el número de nodos.

Aplicaciones más nuevas como Kazaa y eDonkey utilizan súper-peers, los cuales almacenan información indexada sobre los nodos del sistema y de ahí trabajan de la misma forma que un servidor central, salvo que al distribuir la información indexada a través de muchos súper-peers, la vulnerabilidad del sistema es limitada.

2.6 Utilización de las aplicaciones Par-a-Par

La Asociación para la Investigación de los Medios de Comunicación o **AIMC**, es una entidad sin fines de lucro cuyo fin social es aumentar el conocimiento sobre el uso de los diferentes medios de comunicación. AIMC ha llevado a cabo diversos estudios en torno a Internet cuyos resultados se pueden consultar en su página Web (14).

Según los datos aportados por la AIMC:

- Uno de cada cinco ínter nautas (20,1%) utiliza estas redes de intercambio varias veces al día.

- Un 26,9%, sin embargo, asegura no hacerlo nunca.

La Figura 2.6 muestra en detalle la clasificación de la frecuencia de uso de las redes P2P por parte de los encuestados y los resultados absolutos y en porcentaje.

<i>P. ¿Cómo clasificaría su frecuencia de uso de...</i>		
	Absolutos	%
BASE	57.310	100,0
Redes de intercambio de archivos P2P (eMule, Kazaa,...)		
Varias veces al día	11.506	20,1
Todos o casi todos los días	8.855	15,5
Varias veces a la semana	6.377	11,1
Una vez por semana	2.395	4,2
Un par de veces al mes	3.032	5,3
Una vez por mes	1.420	2,5
Menos de una vez al mes	3.031	5,3
Nunca ó prácticamente nunca	15.391	26,9
NS/NC	5.303	9,3

Fig. 2.6 (REF. 14)

Utilización de las aplicaciones Par-a-Par

La Figura 2.7 muestra un grafico de barras de los datos mostrados en la Figura 2.6.

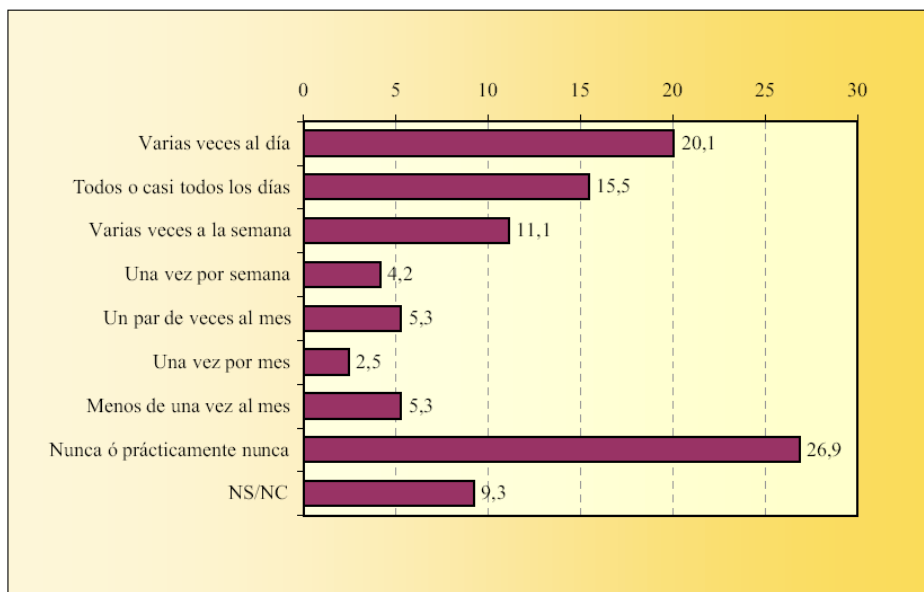


Fig. 2.7 (REF. 14)

Gráfico de utilización de las aplicaciones Par-a-Par

Basándonos en la clasificación de servicios utilizados en la red, La Figura 2.8 muestra que el intercambio de archivos mediante redes P2P ocupa el cuarto lugar, tras el correo electrónico, la navegación por la red, y la mensajería instantánea.

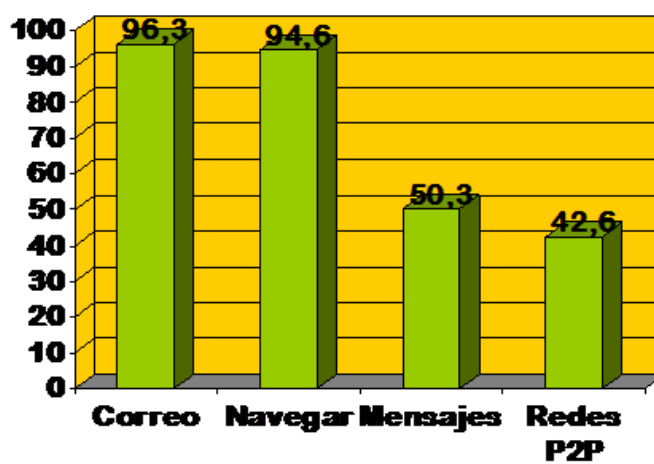


Fig. 2.8 (REF. 14)

Gráfico de Utilización de las aplicaciones Par-a-Par

2.7 Historia de las aplicaciones Par-a-Par

Según Wikipedia La Enciclopedia Libre (15), las redes P2P siempre han existido pero nunca han sido reconocidas como ellas. Los servidores con direcciones IP fijas siempre han tenido la capacidad de comunicarse con otros servidores para acceder servicios. Un número de preaplicaciones P2P como el email y el sistema de nombres de dominio (**DNS**) se crearon sobre estas para proveer redes distribuidas. Pero una aplicación se destaca de las otras, Usenet.

Usenet permite que dos computadoras intercambien información. No posee una unidad administrativa central, la distribución del contenido es administrado por cada nodo y el contenido de la red Usenet es replicado a través de sus nodos.

Desde Usenet, las aplicaciones P2P más populares han sido las de mensajería instantánea, transferencia de archivos y computación distribuida.

2.7.1 Mensajería instantánea

ICQ, fue creado por Mirabais en noviembre de 1996, y le dio a los usuarios una manera de comunicarse rápido con sus amigos. Notificaba cuando sus amigos se conectaban y les permitía enviar mensajes instantáneos e intercambiar archivos. La aplicación se clasifica como P2P centralizada, es decir utiliza la arquitectura cliente servidor para proveer su servicio. Pero todas las otras comunicaciones entre usuarios se hacen de forma P2P con mensajes dirigiéndose de la máquina de un usuario a otra sin tener un servidor intermediario. ICQ tuvo muchos imitadores como **MSN Messenger, AOL y Yahoo! Messenger**.

2.7.2 Transferencia de archivos

Napster fue creada en otoño de 1999, utilizaba servidores centrales para almacenar la lista de equipos y los archivos que proporcionaba cada equipo, con lo que no era una aplicación perfectamente P2P. Aunque ya existían aplicaciones que permitían el intercambio de archivos entre los usuarios, como **IRC** (16), y **Usenet**, Napster fue el primero en especializarse en los archivos de música **MP3**.

El hecho de que Napster fuera un servicio centralizado resultó en su perdición. En diciembre de 1999, varias discográficas estadounidenses demandaron a Napster. Durante un tiempo, el intercambio de archivos fue a la deriva. Existían bastantes alternativas de aplicaciones P2P de las cuales se estableció como líder P2P **Audiogalaxy** (17). Pero ésta acabó también por orden judicial ya que al igual que Napster era otra aplicación centralizada de intercambio de música. La **RIAA** (18), la asociación estadounidense de discográficas, tomó estas resoluciones judiciales como victorias importantes encaminadas a acabar con la llamada "piratería".

Cientes nuevos y la aparición de la red Gnutella, fueron sustituyendo a Napster y Audiogalaxy, entre otros. Luego en el 2002, se dio un éxodo masivo de usuarios hacia las redes descentralizadas, como Kazaa, **Grokster** (19), y **Morpheus** (20).

Luego apareció **eDonkey 2000**, aplicación que se mantuvo junto a Kazaa como líder del movimiento P2P.

Otro paso importante lo marcó el protocolo **Bittorrent** (21), que pese a tener muchas similitudes con eDonkey 2000 proporciona,

según los desarrolladores, una mayor velocidad de descarga, pero a costa de una menor variedad de archivos, y una menor longevidad de éstos en la red. La unión de ambos protocolos se ha logrado en el programa **Lphant** (22).

2.7.3 Computación distribuida

La computación distribuida es una manera de resolver problemas difíciles al dividir el problema en subproblemas que pueden ser resueltos independientemente por un gran número de computadores.

En 1996 **SETI@HOME** (23), empezó a distribuir una aplicación de protector de pantalla basada en servidor a los usuarios, para permitirles procesar datos de un radio telescopio y contribuir con la búsqueda de vida extraterrestre.

En un proyecto similar en 1997, Distributed.net usó el poder computacional de sus usuarios para descifrar mensajes encriptados previamente imposibles de quebrar.

En ambos casos, el software del cliente contacta un servidor para descargar su porción del problema que está siendo resuelto. Hasta que el problema es resuelto no se requiere comunicación con el servidor.

En el futuro se espera que la computación distribuida evolucione para que tome ventaja de la tecnología P2P.

2.8 Problemática actual de las redes Par-a-Par

Según Abad, Cristina (12) copiar archivos entre PCs es legal, infringir leyes de copyright es ilegal. Varios creadores de sistemas P2P han sido demandados por las disqueras por infringir en leyes de copyright al incentivar el intercambio ilegal de canciones. De hecho, también se ha demandado a universidades (por las acciones de sus estudiantes) y usuarios comunes. Por ahora, no se puede demandar a los ISPs (proveedores de servicios de Internet). Los ISPs al utilizar cachés, que son conjuntos de datos duplicados de otros originales para mejorar tiempo de acceso, generalmente almacenan archivos MP3 ilegales. Esta acción de almacenar datos la llamaremos “cachear” en el resto del documento. No se puede demandar a los ISPs ya que se ha determinado que es la caché y no el ISP quien decide qué cachear o no,

y por esto no se puede hacer responsable al ISP por el contenido que cachea, según regulación de 1998 de la **DMCA**, Digital Millenium Copyright Act, Acta de copyright del milenio digital de EE.UU. En muchos países no hay leyes al respecto, así que en ellos no queda claro si es legal o ilegal.

CAPÍTULO 3

PLATAFORMA JXTA

3.1 Introducción a Proyecto JXTA

Las aplicaciones P2P actuales tienden a usar protocolos propietarios y de naturaleza incompatible, reduciendo la ventaja ofrecida de reunir dispositivos en redes P2P. Cada red forma una comunidad cerrada, completamente independiente de las otras redes e incapaz de proveer sus servicios.

Hasta ahora la necesidad de explorar las posibilidades de la tecnología P2P ha superado la importancia de la interoperabilidad y reutilización de software. Para evolucionar al P2P en una solución de plataforma madura los desarrolladores necesitan un lenguaje común para permitir a los peers comunicarse y realizar lo fundamental en las redes P2P.

Tomando en cuenta la necesidad de un lenguaje P2P común, **Sun Microsystems** (24), formó el proyecto JXTA (25) bajo la guía de Hill Joy y Mike Clark para diseñar una solución que sirva a todas las aplicaciones

P2P. En su núcleo, JXTA es simplemente un grupo de especificaciones de protocolos, lo que lo hace poderoso.

El nombre JXTA se deriva de la palabra *juxtapose* (yuxtaponer) que significa colocar dos entidades lado a lado o en una proximidad. Al escoger este nombre el equipo desarrollador de Sun reconoció que las soluciones P2P siempre existirán al lado de las soluciones cliente servidor en vez de reemplazarlas completamente.

3.2 JXTA

JXTA es un grupo de protocolos abiertos p2p que permiten a cualquier dispositivo en la red, desde un teléfono celular a PDA, desde una PC a un servidor, comunicarse y colaborar como peers. Estos protocolos son independientes de cualquier lenguaje de programación y múltiples implementaciones, llamadas **bindings**, existen para diferentes ambientes. Nosotros utilizamos el JXTA binding de la plataforma **J2SE**.

JXTA provee una plataforma con las funciones básicas necesarias para una red P2P:

- **Interoperabilidad:** Permite a los peers que proveen varios servicios P2P, localizarse entre sí y comunicarse entre ellos.

- **Independencia de plataforma:** Es independiente de lenguajes de programación, protocolos de transporte y plataformas de despliegue.

Una característica común de los peers en una red P2P es que a menudo existen en el límite de una red regular. Al ser sujetos de conectividad no predecible con direcciones de redes potencialmente variables, se encuentran fuera del alcance del DNS. JXTA acomoda a los peers en el límite de la red para proveer un sistema para direccionar únicamente a los peers, que es independiente del servicio tradicional de nombres. A través del uso de ID's de JXTA un peer puede moverse en las redes, cambiando transportes y direcciones de red, aún puede estar temporalmente desconectado, y seguir siendo direccionable por otros peers.

JXTA provee un grupo de protocolos abiertos y una implementación de referencia de código abierto para el desarrollo de aplicaciones peer-to-peer. Los protocolos de JXTA estandarizan la manera en que los peers:

- Se descubren entre sí (Discover)
- Se auto organizan en grupos de peers
- Publican y descubren servicios de red (**Advertise** and **Discovery**)
- Se comunican entre ellos

- Se monitorean entre ellos

Los protocolos pueden ser implementados en el lenguaje de programación Java, C/C++, Perl y otros lenguajes. Pueden ser implementados sobre **TCP/IP**, **HTTP**, **Bluetooth**, **HomePNA** u otro protocolo de transporte.

Los protocolos de JXTA permiten a los desarrolladores construir y desplegar servicios y aplicaciones P2P ínter operables. Dispositivos heterogéneos con software completamente diferente pueden operar entre sí. Con la tecnología JXTA los desarrolladores pueden escribir aplicaciones de redes ínter operables que puedan:

- Buscar otros peers en la red con descubrimiento dinámico a través de **firewalls**
- Fácilmente compartir documentos con cualquiera a través de la red
- Encontrar hasta el contenido más minucioso en sitios de red
- Crear un grupo de peers que provean un servicio
- Monitorear actividades de peers remotamente
- Comunicarse de manera segura con otros peers en la red

3.3 Arquitectura de JXTA™

La arquitectura de software JXTA se encuentra dividida en tres capas, como se muestra en la Figura 3.1 a continuación.

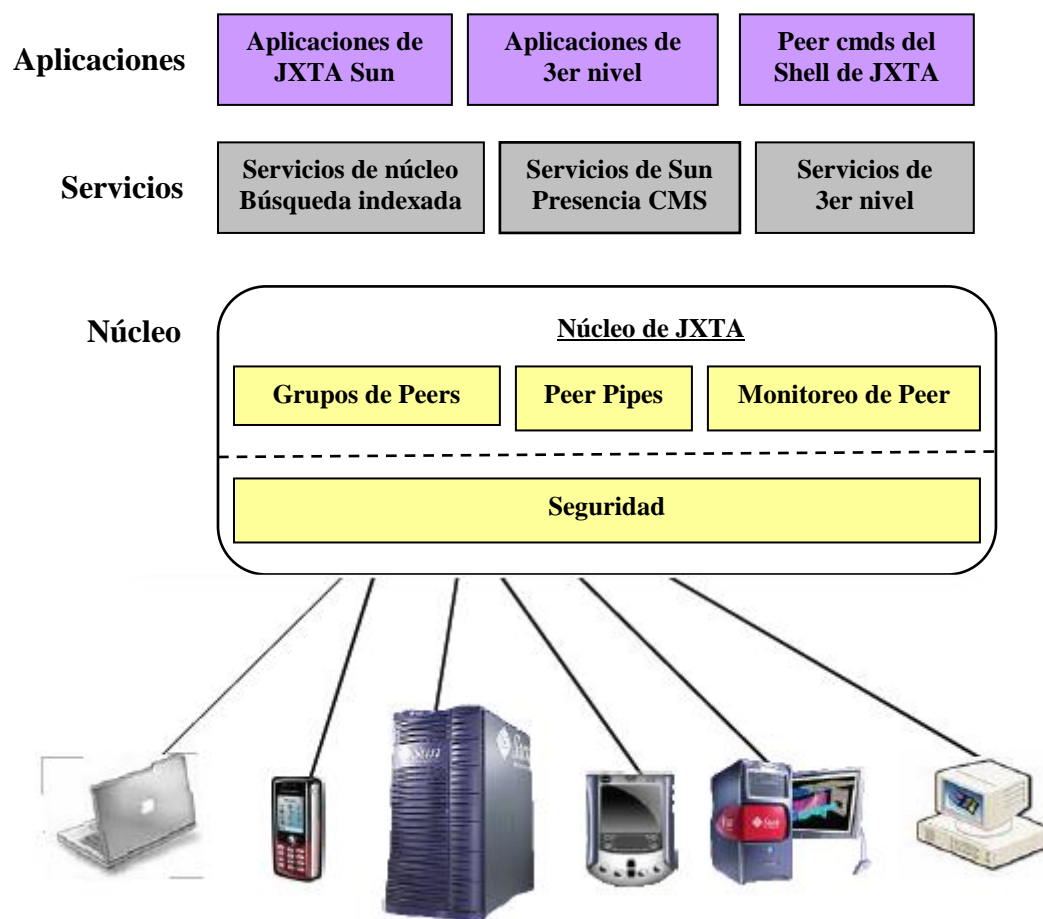


Fig. 3.1 (REF. 26)

Peers en la Web expandida

- **Capa de plataforma (Núcleo de JXTA):** Encapsula propiedades comunes a la red p2p. Incluye bloques para permitir mecanismos

de clave para aplicaciones p2p, incluye discovery, transporte (incluye manejo de firewall), la creación de peers y grupos de peers.

- **Capa de Servicios:** Incluye servicios de red que pueden no ser absolutamente necesarios para que una red P2P pueda operar, pero son comunes o deseados en el ambiente p2p. Algunos ejemplos de servicios de red incluyen: buscar e **indexar**, directorio, sistemas de almacenamiento, compartir archivos, sistemas de archivos distribuidos, agregación de recursos y renta, traslado de protocolos, autenticación y servicios de **PKI** (Infraestructura de clave pública).
- **Capa de Aplicaciones:** Incluye implementación de aplicaciones integradas, como mensajes instantáneos P2P, compartir documentos y recursos, manejar y entregar contenido de entretenimiento, sistemas de mail P2P y muchos otros.

La frontera entre servicios y aplicaciones no es rígida. Una aplicación para un cliente puede ser vista como un servicio para otro cliente. Todo el sistema es diseñado para ser modular, permitiendo a los desarrolladores escoger entre una colección de servicios y aplicaciones que coincidan con sus necesidades.

La red JXTA consiste en una serie de nodos interconectados, también llamados peers. Los peers se pueden auto-organizar en grupos de peers, que proveen servicios comunes. Ejemplos de servicios que pueden ser proveídos por un grupo de peers incluyen compartir documentos o aplicaciones de Chat.

Los peers de JXTA publican (*advertise*) sus servicios en documentos XML llamados publicaciones (*advertisements*). Las publicaciones permiten que otros peers en la red aprendan a cómo conectarse e interactuar con los servicios de los *peers*.

Los peers de JXTA usan tuberías (*pipes*) para enviar mensajes entre ellos. Los *pipes* son mecanismos de transferencia de mensajes asincrónicos y unidireccionales usados para el servicio de comunicación. Los mensajes son simplemente documentos XML cuyo sobre contiene credenciales e información de enrutamiento.

JXTA se distingue de otros modelos de redes distribuidos por:

- El uso de documentos XML (publicaciones) para describir recursos de red.
- Abstracción de *pipes* para *peers*, sin depender de un nombramiento central de autoridad de direccionamiento como un DNS.

- Un esquema uniforme de direccionamiento de peers (ID's de Peers).

3.4 Conceptos de JXTA

Esta sección describe los conceptos más importantes de la plataforma JXTA.

3.4.1 Peers

Es cualquier dispositivo de red que implementa uno o más de los protocolos de JXTA. *Peers* pueden incluir sensores, teléfonos y PDA's, así mismo como PC's, servidores y supercomputadores. Cada peer opera independientemente y de manera asíncrona desde los otros peers y es identificado por un **Peer ID**.

Los peers publican una o más interfaces de redes para usar con los protocolos de JXTA. Cada interfaz es publicada como un **endpoint** de peer que identifica como única a la interfaz de red. Los endpoints de peer son usados por los peers para establecer conexiones directas punto a punto entre dos peers.

Los peers no requieren tener conexiones de red directas punto a punto entre ellos. Peers intermediarios pueden ser usados para enrutar mensajes a peers que se encuentran separados por conexiones de red física o configuraciones de red, por ejemplo **NATs**, firewalls, **proxies**, etc.

Los peers son configurados espontáneamente para descubrirse unos a otros en la red para formar relaciones transitorias o persistentes llamadas peer groups (grupos de peers).

3.4.2 Grupos de *Peers*

Es una colección de Peers que se han puesto de acuerdo sobre un grupo de servicios. Los peers se auto-organizan entre grupos de peers, cada uno identificado por un único ID del grupo. Cada peer group puede establecer su propia política de membresía, desde abierta hasta alta seguridad para asegurar y proteger.

Los peers pueden pertenecer a más de un grupo simultáneamente. Por defecto el primer grupo que es creado es el **Net peer Group**. Todos los peers pertenecen a él. Los peers pueden elegir unirse a grupos adicionales.

Los protocolos de JXTA describen cómo los peers pueden publicar, descubrir (discover), unirse (join) y monitorear grupos, ellos no dictan cuándo o por qué los grupos son creados.

Un grupo de peer provee un grupo de servicios llamados servicios de grupo de peer. JXTA define un núcleo de servicios. Para que dos peers interactúen vía servicio deben ser parte del mismo grupo de peers.

Los servicios son los siguientes:

- **Servicio de Descubrimiento (Discovery Service)** – Es usado por miembros de peers para buscar recursos de los grupos de peers, como peers, grupos, pipes y servicios.
- **Servicio de membresía (Membership Service)** – Es usado por miembros actuales para rechazar o aceptar una aplicación de membresía de nuevo grupo. Los peers que deseen unirse a un grupo deben primero localizar un miembro actual y luego requerir una unión o “join”. La aplicación a join es rechazada o aceptada por los miembros actuales. El servicio de membresía puede incentivar una votación de peers o elegir un grupo

designado representativo para aceptar o rechazar las aplicaciones de membresía.

- **Servicio de Acceso (Access Service)** – es usado para validar pedidos hechos por un peer para otro. Los peers que reciben el requerimiento proveen a los peers que realizan el pedido de credenciales e información sobre los pedidos hechos para determinar si el acceso es permitido.
- **Servicio de Tubería (Pipe Service)** – Es usado para crear y administrar conexiones de pipe entre los miembros de un grupo de peers.
- **Servicio de Resoluciones (Resolver Service)** – Es usado para enviar pedidos genéricos a otros peers. Los peers pueden definir e intercambiar pedidos para encontrar cualquier información que pueda hacer falta.
- **Servicio de monitoreo (Monitoring Service)** – Es usado para permitir que un peer monitoree otros miembros del mismo grupo de peers.

3.4.3 Servicios de Red

Los peers cooperan y se comunican para publicar (**publish**), descubrir (discover) e invocar (invoke) servicios de red. Los peers pueden publicar múltiples servicios. Descubren servicios de red a través del Peer Discovery Protocol.

Los protocolos de JXTA reconocen dos niveles de servicios de redes

- **Servicios de Peer** – Un servicio de peer es accesible sólo en el peer que publica ese servicio. Si ese peer falla, el servicio también falla. Múltiples instancias del servicio pueden correr en peers diferentes, pero cada instancia publica su propio advertisement.
- **Servicios de Peer Group** – Un servicio de peer group está compuesto de una colección de instancias de los servicios corriendo en múltiples miembros del grupo. Si algún peer falla el grupo de servicios colectivos de peers no es afectado. Estos servicios son publicados como parte del advertisement del peer group.

Los servicios pueden preinstalarse en un peer o ser cargados de la red. Para correr un servicio un peer debe localizar una implementación de acuerdo al ambiente del peer. El proceso de encontrar, bajar e instalar un servicio de la red es similar a realizar

una búsqueda en el Internet para una página Web, obtener la página y luego instalar el plug-in requerido.

3.4.4 Módulos

Los módulos de JXTA son una abstracción usados para representar cualquier pedazo de código usado para implementar un comportamiento en el mundo de JXTA. Los servicios de red son el ejemplo más común de comportamiento que puede ser instanciado en un peer. No se especifica lo que el código es, puede ser una clase de Java, un Java jar, una librería dinámica **DLL**, un grupo de mensajes XML o un **script**.

Los módulos proveen una abstracción genérica para permitir a un peer instanciar un nuevo comportamiento. A medida que los peer buscan o se unen a un nuevo grupo pueden encontrar nuevos comportamientos que quieran instanciar. Por ejemplo, cuando un peer se une a un grupo debe aprender un nuevo servicio de búsqueda que sólo es usado por este grupo. Para poder unirse a este grupo el peer debe instanciar este nuevo servicio de búsqueda. El framework del módulo permite la representación y publicación de comportamientos de plataformas independientes y

permite a los peers describir e instanciar cualquier tipo de implementación de un comportamiento. Por ejemplo, un peer tiene la habilidad de instanciar un comportamiento de la implementación en Java o en C.

La habilidad de describir y publicar comportamiento de plataforma independiente es esencial para soportar grupos compuestos por peers heterogéneos. El módulo de advertisement permite a los peers de JXTA describir un comportamiento de manera independiente de la plataforma.

La abstracción de un módulo incluye una clase módulo, una especificación de módulo y una implementación de módulo. A continuación se muestra una descripción de cada uno:

- **Clase de Módulo:** Es usado para publicar la existencia de un comportamiento. Es identificado por un único ID y la clase de módulo ID.
- **Especificación de Módulo:** Es usado para acceder un módulo. Contiene toda la información necesaria para acceder o invocar al módulo.

- **Implementación de Módulo:** Es la implementación de una especificación de módulo dada. Puede haber múltiples implementaciones de módulos para una especificación de módulo dada. Cada uno contiene el **ModuleSpecID** de la especificación asociada que implementa.

Los módulos son usados por los servicios de grupos y pueden también ser usados para servicios de stand-alone.

3.4.5 Pipes

Los peers de JXTA usan pipes para enviar mensajes entre ellos. Son asíncronos y unidireccionales, transportistas de datos y mecanismos de transferencia de mensajes no tan confiables, con la excepción de los pipes seguros unicast. Soportan la transferencia de cualquier objeto, incluyendo código binario, data strings y objetos basados en la tecnología Java.

Los puntos finales (endpoint) son referidos como input pipe, el punto que recibe, y el output pipe, el pipe que envía. Los pipes de JXTA pueden tener endpoints conectados a diferentes peers en diferentes tiempos o no estar conectados.

Los pipes son canales de comunicación virtual y pueden conectar a peers que no tienen un enlace físico directo. En este caso uno o más endpoints de peers intermediarios son usados para pasar mensajes entre los dos endpoints del pipe.

Los pipes ofrecen dos maneras de comunicación, punto a punto y propagación (Ver Figura 3.2). También existe pipe de unicast seguros.

- **Pipes punto-a-punto:** Conecta exactamente dos endpoints de un pipe. Un input pipe en un peer recibe mensajes enviados del output pipe de otro peer. También es posible para múltiples peers unirse a un único input pipe.
- **Pipes de propagación:** Conecta un output pipe a múltiples input pipes. Los mensajes fluyen desde un output pipe al de input. Toda la propagación es hecha en el alcance de un grupo. El output pipe y todos los input pipes deben pertenecer al mismo grupo.
- **Pipes unicast seguros:** Es un tipo de pipe punto a punto que provee un canal de comunicación seguro y confiable.

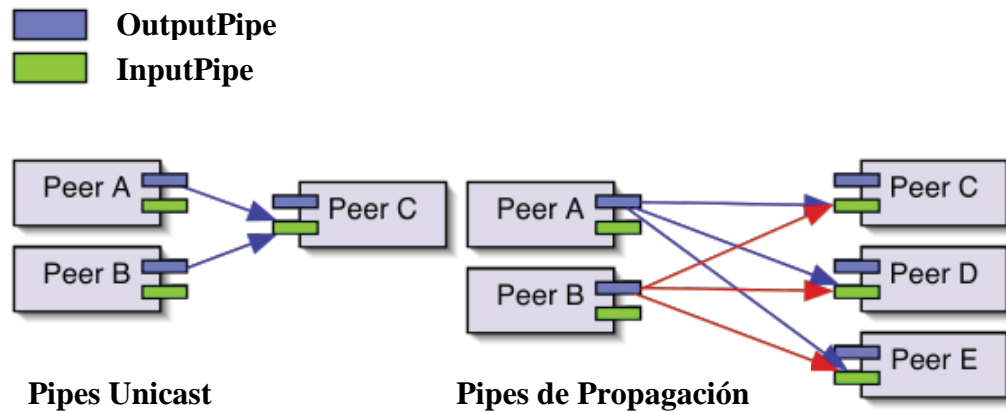


Fig. 3.2 (REF. 27)

Pipes

3.4.6 Mensajes

Un mensaje es un objeto que es enviado entre peers de JXTA. Es la unidad básica de intercambio de datos entre peers. Los mensajes son enviados y recibidos por el servicio de Pipe y por el servicio Endpoint.

Un mensaje es una secuencia ordenada de contenidos nombrados y con tipos llamados elementos de mensaje. El contenido puede ser arbitrario. Hay dos representaciones de mensajes, XML y binario. El uso de mensajes XML para definir los protocolos permite que diferentes clases de peers participen en un

protocolo. Como los datos están en etiquetas cada peer es libre de implementar el protocolo de la mejor manera, según sus habilidades y roles.

3.4.7 Advertisements (Publicaciones)

Todos los recursos de la red de JXTA como peers, grupos, pipes y servicios, son representados por un advertisement. Es un lenguaje neutral de estructura meta data representado por documentos XML. Los protocolos de JXTA usan advertisements para describir y publicar la existencia de los recursos de los peers. Los peers descubren recursos buscando por su correspondiente advertisement y pueden cachear cualquier descubrimiento local.

Cada advertisement es publicado con un tiempo de vida que especifica la disponibilidad de su recurso asociado. Esto permite el borrado de recursos obsoletos sin requerir un control centralizado. Un advertisement puede ser vuelto a publicar para extender su tiempo de vida como recurso.

Los protocolos de JXTA definen los siguientes tipos de advertisement:

- **Advertisement de Peer** – Describe el recurso del peer. Mantiene información específica del peer, como su nombre, ID, endpoints disponibles y cualquier atributo de tiempo de ejecución que los servicios de grupos individuales quieren publicar.
- **Advertisement de Peer Group** - Describe los recursos específicos de un grupo, como nombre, ID de grupo, descripción, especificación y parámetros de servicio.
- **Pipe Advertisement** – Describe un canal de comunicación de pipe y es usado por el servicio de pipe para crear el input y output asociado del endpoint del pipe. Cada pipe advertisement contiene un simbólico ID opcional, un tipo de pipe y un pipe ID único.
- **Clase de Módulo Advertisement** – Describe la clase de un módulo. Su principal propósito es documentar formalmente la existencia de la clase de un módulo. Incluye nombre, descripción y un único ID.
- **Módulo de Especificación de Advertisement**– Define la especificación de un módulo. Su principal propósito es proveer

referencias a la documentación necesitada para crear implementaciones conformes a la especificación.

- **Módulo de Advertisement Impl** – Define la implementación de un módulo de especificación dado. Incluye nombre, ModuleSpecID asociado, así como código, paquete y campos de parámetros que permite a los peers obtener datos necesarios para ejecutar la implementación.
- **Advertisement Rendezvous** – Describe un peer que actúa como un peer rendezvous para un grupo de peers dado.
- **Advertisement con Información de Peer** – Describe la información del recurso del peer. Mantiene información específica sobre el estado actual de un peer, como el tiempo de subida, número de mensajes recibidos y enviados, el tiempo en que se recibió el último mensaje y el que se envió.

3.4.8 Seguridad

Los peers de JXTA operan con un modelo de confianza basado en roles, en el cual un peer actúa bajo la autoridad concedida por otro peer confiable para realizar una tarea específica. Cinco requerimientos de seguridad básica deben ser provistos:

- Confidencialidad: Garantiza que el contenido de un mensaje no sea entregado a individuos no autorizados.
- Autenticación: Garantiza que el emisor sea quien dice ser.
- Autorización: Que el emisor del mensaje se encuentre autorizado.
- Integridad de datos: Garantizar que el mensaje no sea modificado accidentalmente o deliberadamente en el tránsito.
- Refutabilidad: Garantizar que el mensaje sea transmitido por un emisor propiamente identificado y que no sea retransmitido de un mensaje previamente enviado.

Los mensajes XML proveen la habilidad de añadir meta-data como credenciales, certificados, resúmenes (hash criptográficos) y claves públicas a los mensajes de JXTA, permitiendo que los requerimientos de seguridad básica sean obtenidos. Los resúmenes de los mensajes garantizan la integridad de mensajes de los datos. Los mensajes también pueden ser encriptados usando claves públicas y firmados usando certificados para la confidencialidad y refutabilidad. Las credenciales pueden ser usadas para proveer mensaje de autenticación y autorización.

JXTA es compatible con los mecanismos de seguridad de capa de transporte como Secure Sockets Layer (**SSL**) e Internet Protocol Security (IPSec). Sin embargo, estos protocolos sólo proveen la integridad y confiabilidad de la transferencia de mensaje entre dos peers que se comunican. Para proveer transferencia segura en una red multi-salto como JXTA, una asociación confiable debe establecerse entre todos los peers intermediarios. La seguridad se compromete si uno de los enlaces de comunicación no es seguro.

3.4.9 IDs

Peers, grupos, pipes y otros recursos de JXTA necesitan ser identificados como únicos. Un ID identifica una entidad y sirve de manera canónica para referirse a esa entidad. Actualmente hay seis tipos de entidades de JXTA que tienen tipos de JXTA ID definidos: los peers, grupo de peers, pipes, contenidos, clases de módulos y especificaciones de módulos.

Los **URNs** son usados para expresar los ID's de JXTA. Los URNs son una forma de **URI** que deben servir como identificadores de recursos persistentes, de locación independiente. Como otras formas de URI, los ID's de JXTA están presentados como texto.

3.5 Protocolos de JXTA

JXTA define una serie de mensajes con formato XML, o protocolos, para la comunicación entre peers. Los peers usan estos protocolos para descubrirse entre ellos, publicar y descubrir recursos de red, comunicaciones y mensajes de ruta.

JXTA define 6 protocolos:

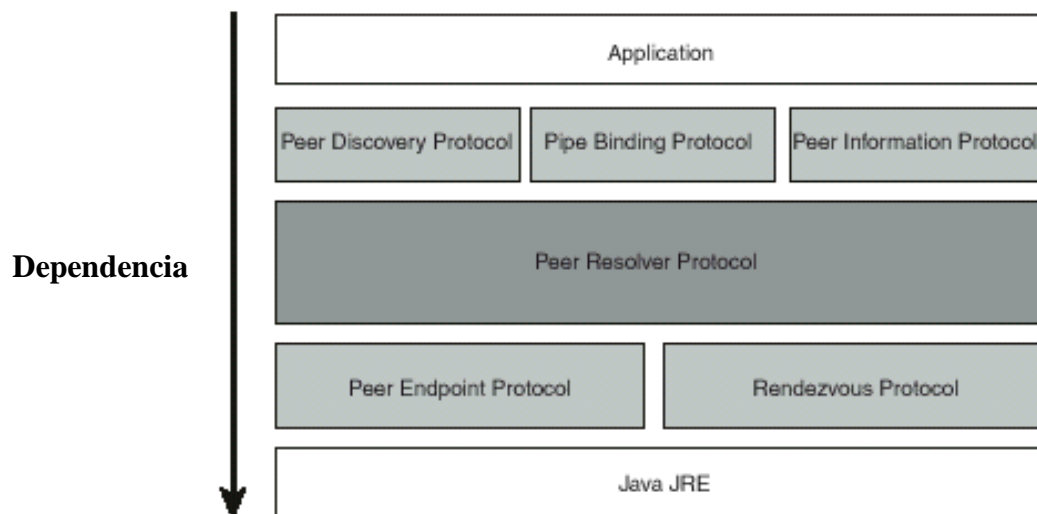
- **Peer Discovery Protocol (PDP)** - Usado por peers para publicar sus propios recursos y descubrir recursos de otros peers. Cada recurso de peer es descrito y publicado usando una publicación (advertisement).
- **Peer Information Protocol (PIP)** - Usado por peers para obtener información del estado de otros peers.
- **Peer Resolver Protocol (PRP)** – Permite a los peers enviar un pedido genérico a cualquier número de peers y recibir una respuesta o múltiples de ellas. Los pedidos pueden ser direccionados a todos los peers en un peer group o a peers específicos en el grupo. A diferencia del PDP y el PIP este protocolo permite a servicios de

peers definir e intercambiar cualquier información arbitraria que necesite.

- **Pipe Binding Protocol (PBP)** - Usado por peers para establecer un canal de comunicación virtual o pipe, entre uno o más peers. El PBP es usado por un peer para unir dos o mas puntos de una conexión (pipe endpoint).
- **Endpoint Routing Protocol (ERP)** - Usado por peers para encontrar rutas a puertos destinatarios en otros peers. La información de ruta incluye una secuencia ordenada de ID's de peers que pueden ser usados para mandar un mensaje al destino.
- **Rendezvous Protocol (RVP)** – Mecanismo con el cual los peers pueden suscribirse o ser un suscriptor de un servicio de propagación. En un peer group, peers pueden rendezvous peers o peers que están escuchando a peers rendezvous. El RVP permite a un peer mandar mensajes a todas las instancias que escuchan del servicio. El RVP es usado por el Peer Resolver Protocol y el Pipe Binding Protocol para propagar mensajes.

Todos los protocolos de JXTA son asíncronos y están basados en un modelo petición/respuesta. Un peer de JXTA usa uno de los protocolos para mandar una petición a uno o más peers en el peer group. Puede recibir cero, una o más respuestas a su petición. Por ejemplo, un peer

puede usar PDP para enviar una petición de discovery preguntando por todos los peers conocidos en el Net peer group por defecto. En este caso, múltiples peers van a querer responder con respuestas de discovery. En la Figura 3.3 se puede observar la jerarquía de los protocolos anteriormente descritos, cabe resaltar que los peers de JXTA no necesitan implementar todos los protocolos, sólo los que van a usar.



Jerarquía de los protocolos

Fig. 3.3 (REF. 28)

Protocolos de JXTA

CAPÍTULO 4

ANÁLISIS Y DISEÑO DEL PROYECTO

4.1 ESPECIFICACIONES

4.1.1 Requerimientos funcionales

- Sólo los usuarios que tienen una cuenta de usuario y contraseña válida de la ESPOL pueden ingresar a la red del chat de la ESPOL.
- Validar el ingreso a nuestro chat con el servicio de autenticación del **CSI** (Centro de Servicios Informáticos).
- Se debe poder unir a la red de mensajería cualquier PC conectada a la red de la ESPOL, con el Java Virtual machine 1.5 o mayor instalado (siempre y cuando un firewall no bloquee su acceso).

4.1.2 Requerimientos no funcionales

- Resolver los problemas de los estudiantes que no poseen una aplicación que les brinde una conexión sincrónica con sus

compañeros de Universidad.

- Dar facilidad de acceso a un chat oficial de la ESPOL con fines académicos ya que las aplicaciones de mensajería, tipo MSN Messenger, se encuentran prohibidas en la universidad.
- Sencillo y de fácil uso, es un sistema de uso “intuitivo” donde no se requiere mayor conocimiento del sistema por parte del usuario para su utilización.

4.2 Diseño Conceptual

4.2.1 Visión del Sistema

- Otorgar a los estudiantes una herramienta de colaboración P2P de mensajería para ser usada en un ambiente en el cual no se desea depender de un servidor central, la cual les permita mantener un chat con múltiples usuarios de todos los puntos de la ESPOL como son rectorados, facultades, laboratorios.

4.2.2 Objetivos

- Dar facilidad de acceso a un chat oficial de la ESPOL con fines académicos ya que las aplicaciones de mensajería tipo MSN

Messenger se encuentran prohibidas en la universidad.

- Resolver los problemas de los estudiantes que no poseen una aplicación que les brinde una conexión sincrónica con sus compañeros de Universidad.
- Otorgar a los estudiantes una herramienta de colaboración la cual les permita mantener un chat con múltiples usuarios de todos los puntos de la ESPOL como son rectorados, facultades, laboratorios.
- Diseñar y desarrollar una aplicación peer-to-peer de mensajería para ser usado en un ambiente en el cual no se desea depender de un servidor central.
- Incursionar en el diseño y desarrollo de aplicaciones basadas en la tecnología JXTA.
- Restringir el ingreso a la aplicación validando el usuario y la contraseña de la cuenta principal de la ESPOL, otorgado a alumnos, profesores y trabajadores de la institución usando la base de datos del CSI (Centro de Servicios Informáticos).
- Mediante evaluaciones de escalabilidad demostrar que la aplicación tiene la capacidad de mantener su rendimiento medio, conforme aumenta el número de usuarios.

4.2.3 Alcances

- **Uso de la tecnología JXTA:** Usar la tecnología JXTA (descrita en el capítulo 3) como plataforma P2P para diseñar un sistema de mensajería universitaria que sea escalable y no dependa de un servidor central.
- **Visualización de usuarios conectados:** En la ventana de usuarios conectados se mostrará a los contactos agregados anteriormente, indicando si se encuentran conectados o no.
- **Permite chat de 1 a 1:** Los usuarios pueden mantener conversaciones sincrónicas con otro usuario conectado a la red en una ventana individual.
- **Permite chat entre múltiples usuarios simultáneamente:** Los usuarios pueden mantener un chat con múltiples usuarios en una misma ventana. Es decir, más de un usuario forma parte de la conversación.
- **Agrupar usuarios en grupos, por facultad y unidad administrativa:** En la ventana que muestra los usuarios conectados al sistema se puede agrupar los usuarios según al grupo o la facultad a la que pertenecen.
- **Creación de grupos:** En la ventana donde muestra a los usuarios conectados se podrá agrupar a los usuarios en los diferentes grupos que el usuario cree.

- **Cartelera comunal:** Cualquier usuario pueda leer o “pegar” anuncios en una cartelera, los cuales perdurarán por un tiempo, aún cuando, el usuario que los coloca se desconecte. De esta manera usuarios que se conecten después podrán leer estos mensajes.
- **Envío de archivos:** El usuario puede enviar archivos a los usuarios conectados.
- **Estado de cada usuario conectado:** El usuario puede ser capaz de cambiar su estado el cual puede ser uno predeterminado como Conectado, No Disponible, Vuelvo Enseguida, Ausente, Al teléfono, Salí a Comer o un estado personalizado como “Trabajando en: [lugar]” o “Trabajando sobre: [materia]”.
- **Búsqueda de usuarios conectados:** El usuario puede realizar búsquedas de usuarios conectados a la red por su nombre de usuario, sobrenombre y/o facultad.
- **Comunicaciones seguras:** Las comunicaciones en el chat se harán de forma segura, siguiendo las propiedades de:
 - Confidencialidad:* Garantizar que los mensajes no sean leídos por otros.
 - Autenticación:* Garantiza que el emisor sea quien dice ser.
 - Autorización:* Que el emisor del mensaje sea autorizado.

Integridad de datos: Garantizar que el mensaje no sea modificado accidentalmente o deliberadamente.

Refutabilidad: Garantizar que el mensaje sea transmitido enviando una certificación, que no sea retransmitido.

4.2.4 Restricciones

- El sistema de chat permite enviar y recibir mensajes únicamente en formato texto.
- No se implementarán recursos multimedia como cámaras, micrófonos, etc.
- Si el usuario no tiene cuenta activa de la ESPOl no puede ingresar al sistema de chat.
- Dado la naturaleza de las redes peer-to-peer la administración del sistema se vuelve compleja, las facultades disponibles a elección de los usuarios son estáticas.
- Una vez publicado un anuncio en la cartelera no podrá ser modificado o borrado; se tendrá que esperar a que se termine su tiempo de vida el cual es de dos semanas.

4.3 Diseño Lógico

4.3.1 Casos de Uso

1. Conectarse a la red (autenticación)
2. Agregar contacto
3. Eliminar contacto
4. Mover contacto
5. Crear grupo
6. Eliminar grupo
7. Iniciar conversación con un usuario
8. Agregar otros usuarios a la conversación
9. Enviar mensajes
10. Recibir mensajes
11. Enviar archivo
12. Recibir archivo
13. Guardar conversación
14. Cambiar estado
15. Leer anuncios de la cartelera comunal
16. Pegar anuncios a la cartelera comunal
17. Buscar usuarios conectados
18. Guardar lista de contactos

19. Leer lista de contactos
20. Cambiar opciones de configuración
21. Cerrar sesión

4.3.2 Descripción de Caso de Uso

- **Caso de uso 1:**
 - **Nombre:** Conectarse a la red
 - **Descripción:** Un usuario desea ingresar a la red para comunicarse con sus contactos, para esto debe conectarse a la red P2P. El usuario debe ingresar el usuario y contraseña de su cuenta ESPOL, la facultad a la que pertenece y su sobrenombre (nickname) para conectarse a la red.
 - **Nota:** Los datos del usuario y la contraseña son obligatorios, la facultad y el sobrenombre son opcionales.
- **Caso de uso 2:**
 - **Nombre:** Agregar contacto
 - **Descripción:** Cualquier usuario que se ha conectado a la red P2P tiene la opción de agregar un usuario a su lista de contactos. Debe hacerlo insertando el mail de la ESPOL del contacto a agregar.
 - **Nota:** El usuario recién agregado va a ser colocado dentro del

grupo de No determinados.

- **Caso de uso 3:**

- **Nombre:** Eliminar contacto

- **Descripción:** Cualquier usuario que se ha conectado a la red P2P tiene la opción de eliminar un contacto de su lista.

- **Nota:** Debe hacer clic derecho sobre él y escoger eliminar.

- **Caso de uso 4:**

- **Nombre:** Mover contacto

- **Descripción:** Cualquier usuario que se ha conectado a la red P2P tiene la opción de mover a los usuarios de su lista de contactos a los diferentes grupos de su lista.

- **Nota:** Puede hacerlo haciendo clic en el usuario, escoger Mover a y seleccionar el grupo al cual desea mover el contacto.

- **Caso de uso 5:**

- **Nombre:** Crear grupo

- **Descripción:** El usuario que se ha conectado a la red, tiene opción a crear un grupo al cual puede agregar amigos o personas con características similares.

- **Caso de uso 6:**

- **Nombre:** Eliminar grupo

- **Descripción:** El usuario que se ha conectado a la red, tiene opción a eliminar un grupo que ha creado anteriormente.

- **Nota:** Para eliminar un grupo debe borrar primero a todos los contactos que hay en él.
- **Caso de uso 7:**
- **Nombre:** Iniciar conversación con un usuario
- **Descripción:** El usuario que se ha conectado a la red, puede iniciar la conversación con un usuario haciendo doble clic sobre él, clic derecho o escogiendo la opción en el menú.
- **Nota:** Se abre una ventana individual para iniciar la conversación, la cual mostrará el texto de la conversación en un área de texto y un campo de texto para escribir el mensaje y un botón para enviarlo.
- **Caso de uso 8:**
- **Nombre:** Agregar otros usuarios a la conversación
- **Descripción:** El usuario tiene la opción en la ventana de conversación de oprimir el botón agregar para obtener una ventana con todos los usuarios conectados y escoger al contacto que desea agregar.
- **Nota:** Puede agregar solamente a usuarios conectados.
- **Caso de uso 9:**
- **Nombre:** Enviar mensajes
- **Descripción:** El usuario que se ha conectado a la red, puede escribir mensajes para que sean vistos por sus contactos en

una conversación específica.

- **Nota:** Debe escribir el mensaje dentro de la caja de texto y oprimir el botón para enviarlo.
- **Caso de uso 10:**
- **Nombre:** Recibir mensajes
- **Descripción:** Un usuario del sistema puede recibir un mensaje de parte de cualquier otra persona que haya iniciado una conversación.
- **Nota:** Los mensajes son leídos del textarea en la ventana de conversación.
- **Caso de uso 11:**
- **Nombre:** Enviar archivo
- **Descripción:** Cualquier usuario en el sistema puede enviar un archivo a un contacto conectado. En la ventana de conversación el usuario podrá elegir la opción de enviar archivo, luego de oprimirla aparecerá la pantalla para que elija el archivo y enviarlo.
- **Nota:** En el menú de la ventana principal del chat el usuario tendrá la opción de ir a Acciones y escoger enviar archivos.
- **Caso de uso 12:**
- **Nombre:** Recibir archivo
- **Descripción:** Cualquier usuario en el sistema puede recibir un

archivo de un contacto. En la ventana de conversación el usuario podrá aceptar la opción de recibir archivo.

- **Nota:** Luego de recibirlo el archivo podrá ser visto en una carpeta específica.
- **Caso de Uso 13:**
- **Nombre:** Guardar conversación
- **Descripción:** Cualquier usuario conectado al sistema tiene la opción de guardar una conversación específica que haya tenido con un usuario.
- **Nota:** Puede hacerlo desde la ventana de conversación con un usuario, eligiendo archivo, guardar conversación y la ruta en la cual será guardada.
- **Caso de Uso 14:**
- **Nombre:** Cambiar estado
- **Descripción:** Cualquier usuario conectado al sistema tiene la opción de cambiar su estado. Puede escoger entre Conectado, No Disponible, Vuelvo Enseguida, Ausente, Al teléfono, Salí a Comer o un mensaje personalizado.
- **Nota:** Puede hacerlo desde el menú Archivo, Estado. Si escoge el estado personalizado una ventana aparecerá en la cual se debe ingresar el texto.
- **Caso de uso 15:**

- **Nombre:** Leer anuncios de la cartelera comunal
- **Descripción:** Cualquier usuario conectado al sistema tiene la opción de leer mensajes previamente colocados por otros usuarios en una ventana común que simula a una cartelera comunal.
- **Nota:** Puede hacerlo mediante el menú de Acciones y escogiendo la opción de Leer/Publicar anuncio en cartelera. También oprimiendo el dibujo de una cartelera en la parte superior derecha de la ventana principal.
- **Caso de uso 16:**
- **Nombre:** Pegar anuncios de la cartelera comunal
- **Descripción:** Cualquier usuario conectado al sistema tiene la opción de colocar mensajes en una ventana que puede ser leída por cualquier usuario conectado a la red. Lo hace escribiendo en la caja de texto el mensaje y oprimiendo publicar.
- **Nota:** Puede hacerlo mediante el menú de Acciones y escogiendo la opción de Leer/Publicar anuncio en cartelera. También oprimiendo el dibujo de una cartelera en la parte superior derecha de la ventana principal. Los mensajes perdurarán por un tiempo determinado aun cuando el usuario que los coloca se desconecte. De esta manera usuarios que se

conecten después podrán leer estos mensajes.

▪ **Caso de uso 17:**

▪ **Nombre:** Buscar usuarios conectados

▪ **Descripción:** Cualquier usuario conectado al sistema tiene la opción de buscar a algún usuario específico que también se encuentre conectado.

▪ **Nota:** Puede hacerlo por usuario, nickname o facultad a la que pertenece. Puede detener la búsqueda si lo desea.

▪ **Caso de uso 18:**

▪ **Nombre:** Guardar lista de contactos

▪ **Descripción:** Cualquier usuario conectado al sistema tiene la opción de guardar en un archivo XML su lista de contactos actual que se compone de los grupos y contactos que posee.

▪ **Nota:** La lista es guardada en la dirección que el usuario especifique.

▪ **Caso de uso 19:**

▪ **Nombre:** Leer lista de contactos

▪ **Descripción:** Cualquier usuario conectado al sistema tiene la opción de leer un archivo XML que contenga una lista de contactos que se compone de grupos y los contactos que hay en cada uno de ellos.

▪ **Nota:** La lista es leída de la dirección que el usuario

especifique.

- **Caso de uso 20:**
- **Nombre:** Cambiar opciones de configuración
- **Descripción:** Un usuario tiene la opción de cambiar las opciones de configuración de la aplicación como son las opciones de servidor Proxy y de desempeño de JXTA.
- **Nota:** En la opción de servidor Proxy el usuario debe ingresar el servidor Proxy y el puerto. En la opción de JXTA el usuario debe indicar si se desea actuar como un peer rendezvous o no.
- **Caso de uso 21:**
- **Nombre:** Cerrar sesión
- **Descripción:** Un usuario conectado al sistema tiene la opción de cerrar sesión para así terminar la ejecución de la aplicación.
- **Nota:** Si al cerrar sesión el usuario tiene ventanas de conversación abiertas, obtendrá un mensaje el cual le advertirá que las ventanas abiertas se cerrarán si acepta continuar con el cierre de sesión.

4.3.3 Lista de Escenarios

La lista de escenarios correspondiente para cada uno de los casos de uso se encuentra en el Apéndice B.1.

4.3.4 Especificación de escenarios

La especificación de los escenarios listados en el subíndice 4.3.3 se encuentra en el Apéndice B.2.

4.3.5 Especificación de Actores

Nombre: Usuario

Descripción: El usuario es capaz de ingresar a la red p2p y comunicarse con los demás usuarios en el chat mediante una comunicación privada con un usuario.

Nota: El usuario debe poseer la aplicación para acceder a la red.

Nombre: Sistema de chat p2p

Descripción: Es el encargado de conectar a toda la red compuesta de usuarios que poseen la aplicación p2p. Conjunto de tecnologías que habilitan el intercambio directo de servicios o datos entre computadores

Nota: Corre sin ningún control centralizado u organización jerárquica.

4.3.6 Diagrama de Casos de Uso

La Figura 4.2 muestra los casos de uso del 1 al 10.



Fig. 4.1

Diagrama de Casos de Uso (parte 1)

La Figura 4.3 muestra los casos de uso del 11 al 21.

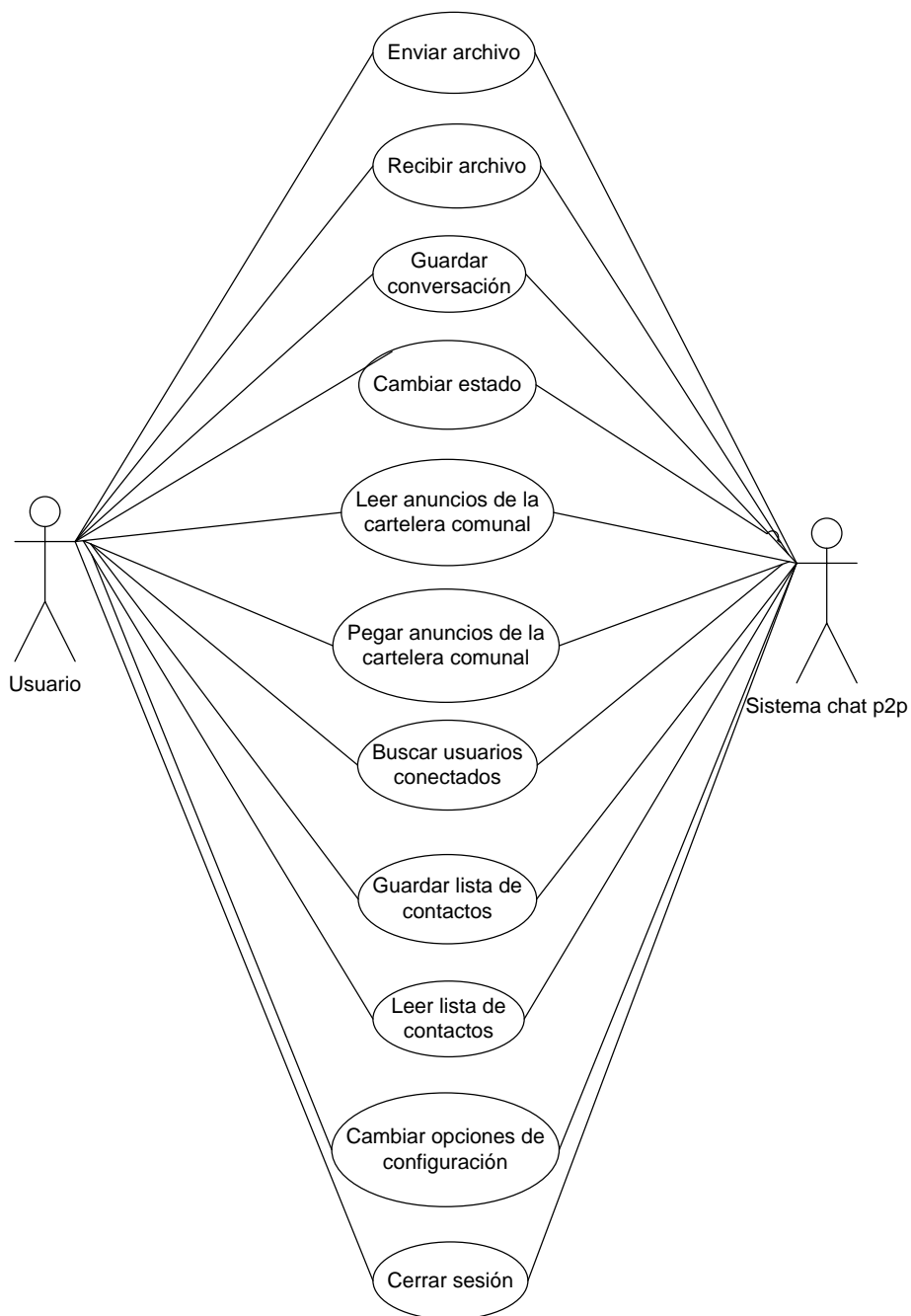


Fig. 4.2

Diagrama de Casos de Uso (parte 2)

4.3.7 Diagrama UML

La Figura 4.3 muestra el diagrama UML de la aplicación.

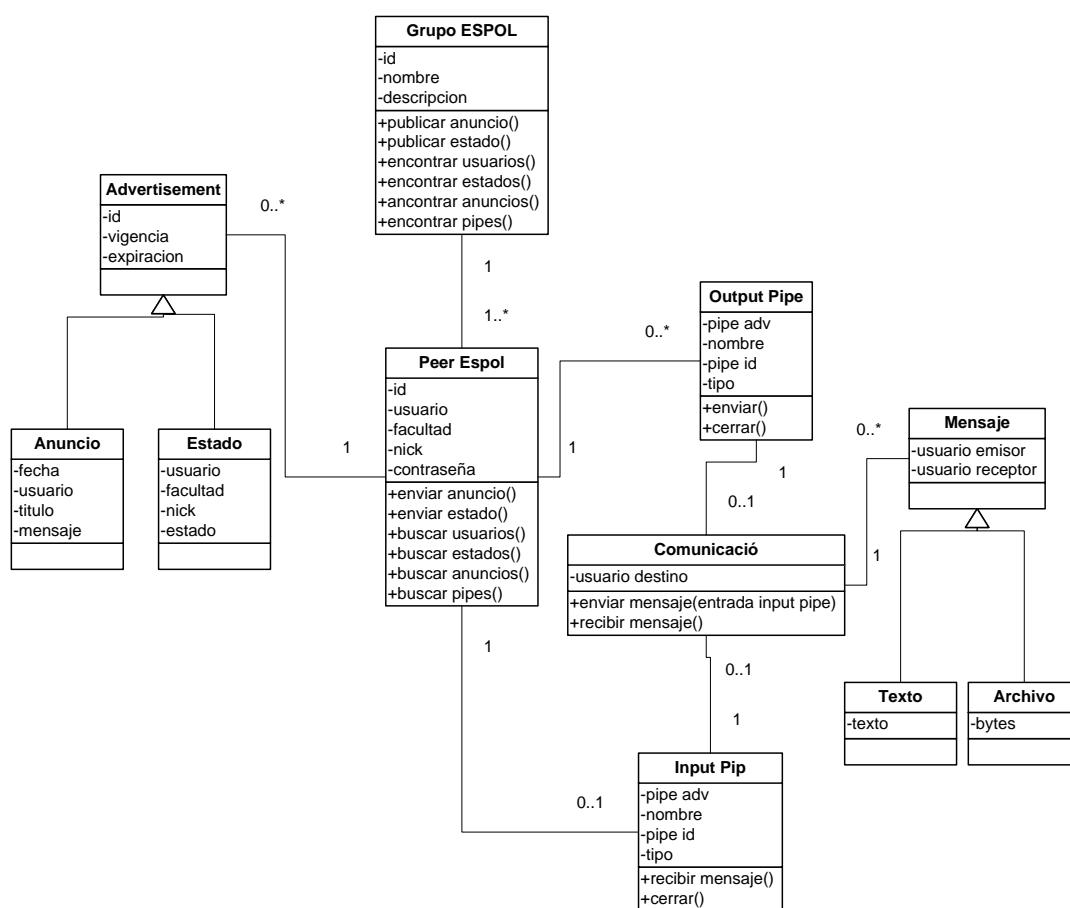


Fig. 4.3

Diagrama UML

CAPÍTULO 5

IMPLEMENTACIÓN DE PROYECTO

5.1 Uso de NetBeans IDE 5.0

El **IDE de NetBeans**, “Integrated Development Environment” o Entorno Integrado de Desarrollo es un IDE de Java de código abierto robusto y gratis que provee al desarrollador con todo lo necesario para crear aplicaciones de escritorio, Web o móviles. La plataforma de NetBeans es 100% Java y corre en cualquier Java Virtual Machine compatible. Las aplicaciones basadas en NetBeans son escritas una vez y corren en cualquier lugar (29).

5.2 Servidor Proxy

Un Servidor Proxy/Firewall es un servidor que se encuentra entre la aplicación de un cliente, como puede ser un Web browser, y un verdadero servidor. Intercepta todos los pedidos al verdadero servidor para ver si puede completar los pedidos él mismo. Si no, envía el pedido al verdadero servidor.

La ESPOL posee un servidor Proxy el cual se encarga de mejorar el desempeño de pedidos para sus grupos de usuarios y filtrar los pedidos para tener un mejor control.

El uso combinado de Proxies, NAT y firewalls resulta en una serie de difíciles circunstancias para la comunicación entre peers. Los peers no se pueden conectar a las máquinas detrás de NAT a menos que un peer interno inicie la comunicación. La única herramienta que un peer posee para resolver este problema es su capacidad de crear conexiones salientes.

En muchas redes corporativas, HTTP es el protocolo más propenso a ser habilitado por un firewall para conexiones salientes. Desafortunadamente, HTTP es un protocolo de pedido respuesta. Cada conexión HTTP envía un pedido y luego espera una respuesta. La conexión debe permanecer abierta luego del pedido inicial para recibir la respuesta. Pero, no provee la capacidad para peers externos de cruzar espontáneamente el límite del firewall y conectarse a los peers dentro de la red interna; de este modo, los peer que se encuentran fuera del firewall no podrían ni siquiera descubrir a los peers que se encuentran dentro del firewall por lo que, la red p2p no se expandiría.

Para solucionar este problema, un peer dentro del firewall utiliza un Peer Router localizado fuera del firewall o visible fuera del firewall para atravesar el firewall. Los peers que tratan de contactar otro peer detrás de un firewall se conectan al peer router, y el peer detrás del firewall periódicamente se conecta al peer router. Cuando el peer interno se conecta al router, cualquier mensaje entrante es enviado al peer en respuesta http. La Figura 5.1 explica este proceso. El peer router es llamado también “Peer Relay”.

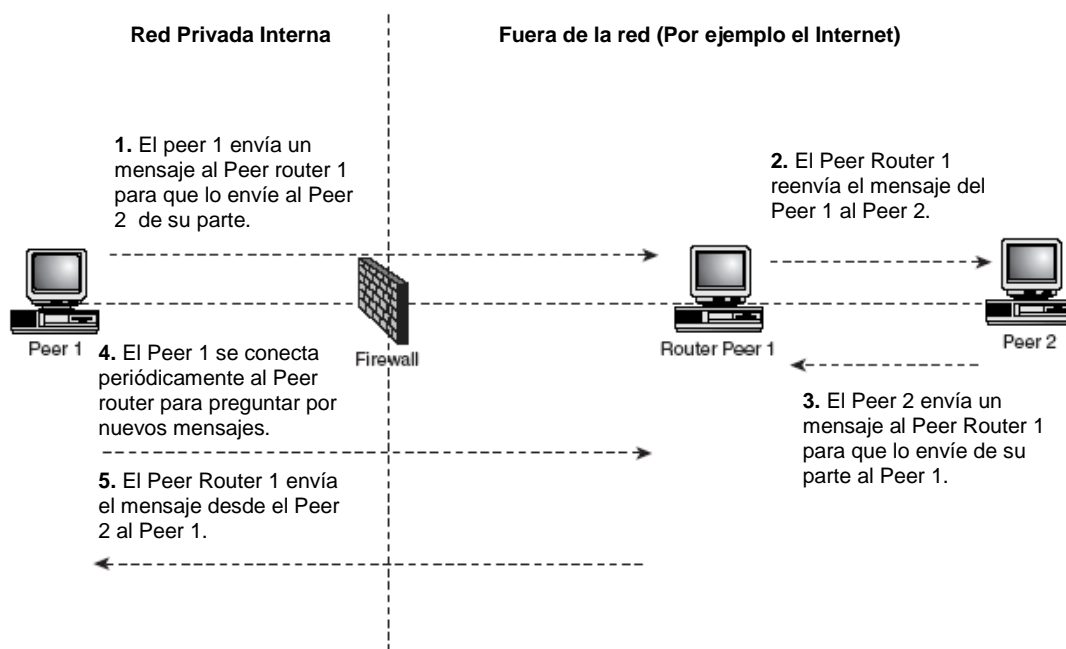


Fig. 5.1

Comunicación de Peers a través de Proxies, Firewalls o NATS

Para poder usar el Messenger basado en JXTA en la ESPOL y poder ver a los usuarios que se encuentren detrás del Proxy Server hay que

configurar JXTA para comunicarse vía http Proxy Servers y configurar los proxies que puedan requerir autenticación.

5.2.1 Configuración de JXTA para comunicarse vía HTTP Proxy Servers

Para habilitar que los peers de JXTA se comuniquen a través de firewalls, JXTA soporta el uso de HTTP proxies, los mismos proxies usados por los Web Browsers. Algunas configuraciones de Proxy Server pueden requerir también autenticación.

Existe una opción en la aplicación la cual le permite al usuario su configuración ingresando el servidor Proxy y el puerto.

El código de autenticación para trabajar con servidores proxies a través de JXTA se encuentra en el Apéndice C.1.

5.3 Autenticación

El uso de nuestra aplicación será restringido para miembros de la comunidad ESPOL únicamente, validando el ingreso con la información

de las cuentas de usuarios, trabajadores y profesores de la base de datos del CSI. La validación la haremos por medio de servicios Web.

5.3.1 Servicio Web

Los servicios Web permiten que diferentes aplicaciones de diferentes fuentes se comuniquen entre ellas sin código que consuma tiempo y debido a que toda la comunicación es en XML, los servicios Web no se encuentran atados a ningún sistema operativo o lenguaje de programación. Por ejemplo, Java puede comunicarse con Perl, las aplicaciones de Windows pueden comunicarse con las de **UNIX**.

Los servicios Web son básicamente funciones que se ejecutan en un servidor y son accedidas por los clientes mediante Internet (30). La Figura 5.2 explica cómo funciona un servicio Web en el que se ejecutan funciones en un servidor y son accedidos por los clientes a través del Internet.

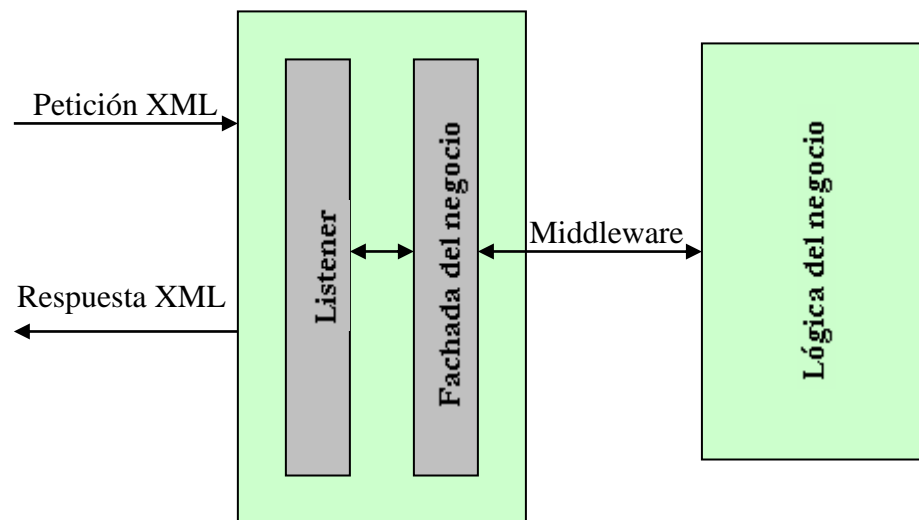


Fig. 5.2 (REF. 31)

Arquitectura genérica de un Servicio Web

Utilizando Netbeans 5.0 y con ayuda del asistente de cliente de servicio Web especificamos el archivo WSDL que queremos consumir. En este caso el servicio de autenticación de la ESPOL. En el campo URL de WSDL colocamos el siguiente URL:
<https://www.academico.espol.edu.ec/services/directorioEspol.asmx>
 Luego recuperamos el WSDL para descargarlo. Procedemos a trabajar con el código que se muestra en el Apéndice C.2.

Encontramos un problema al tratar de autenticar al usuario de nuestra aplicación a través de una conexión al servicio Web detrás de un firewall o servidor proxies. Para solucionar esto se debe

utilizar la autenticación que se muestra en el código del Apéndice D.3.

5.4 Unirse al Net Peer Group

Al iniciar, por defecto todos los peers son miembros del Net Peer Group, lo que permite a todos los peers en la red verse y comunicarse entre ellos. Los servicios que provee se llaman servicios de peer group para distinguirlos de los servicios de peer. Los servicios de peer group pueden ser implementados por varios miembros de un peer group, habilitando la redundancia. A diferencia de un servicio de peer, un servicios de peer group se mantiene disponible mientras uno de los miembros del peer group esté conectado a la red P2P y provea el servicio. La Figura 5.3 muestra esta característica. El código de esta implementación se encuentra en el Apéndice C.4.

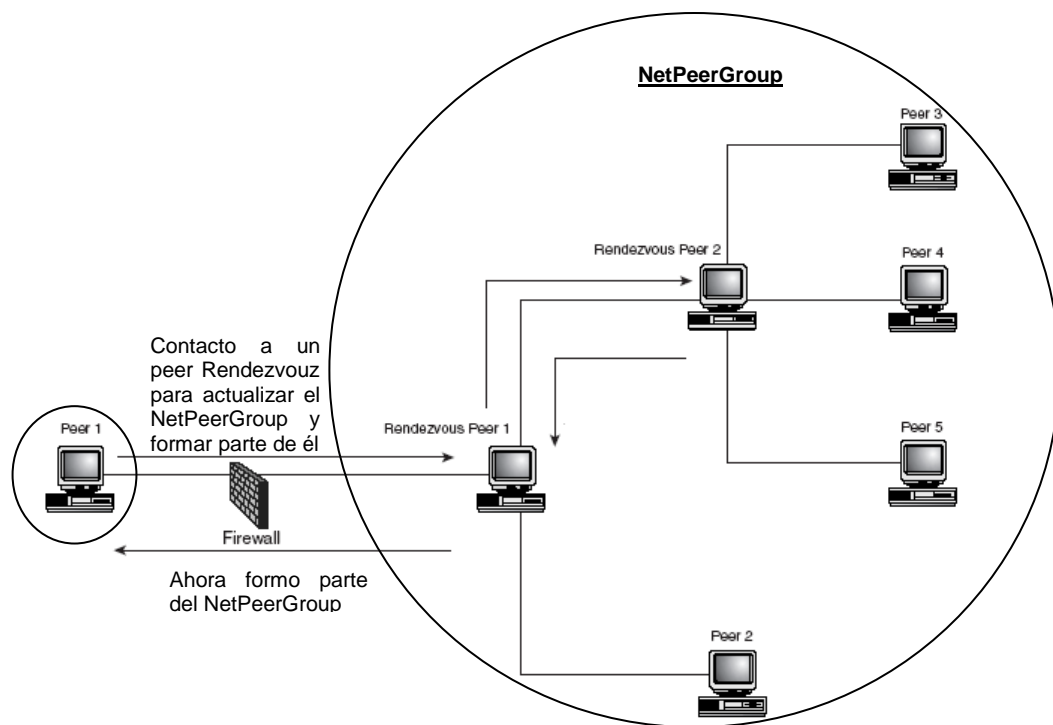


Fig. 5.3

Creación del NetPeerGroup

5.5 Creación del grupo ESPOL

Se debe crear un grupo ESPOL cada vez que se quiera formar parte de él. Todas las aplicaciones de los usuarios contienen un número que identifica al grupo ESPOL en las líneas de código del programa. Así se evita tener que crear un grupo inicialmente que luego debe ser encontrado por los usuarios. Puede ocurrir que el grupo haya sido cacheado en un peer advertisement que no se encuentre conectado y no

lo pueda encontrar al buscarlo. La figura 5.4 muestra esta característica.

El código se muestra en el Apéndice C.5.

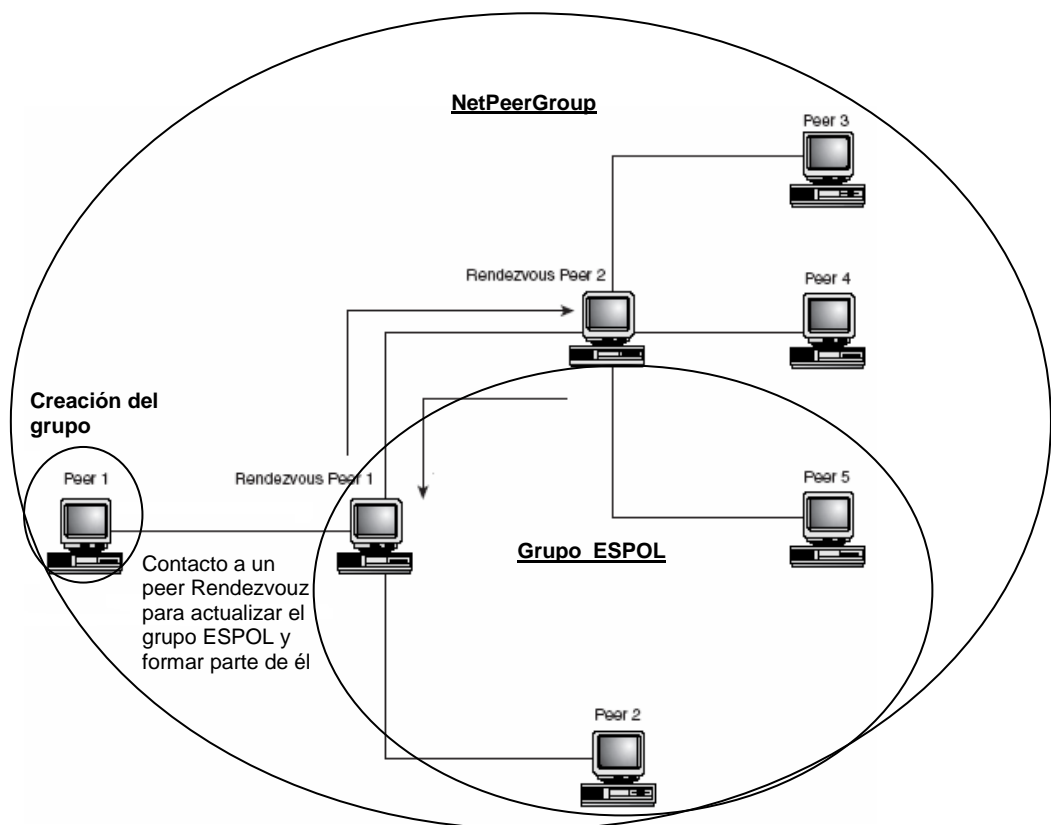


Fig 5.4

Creación del Grupo ESPOL

5.6 Unirse al grupo ESPOL

Para unirse al grupo de peers de la ESPOL es necesario descubrir y manipular grupos de peers. Trabajar con un grupo de peers involucra

trabajar con un Peer Group Advertisement el cual describe al grupo y sus servicios que mantiene disponibles para sus miembros.

JXTA es parte de un mundo heterogéneo en el cual los dispositivos corren en diferentes plataformas o modelos de servicios de invocación pueden descubrir e interactuar entre ellos. Por lo tanto, el comportamiento de las políticas de JXTA necesita ser descrito de manera independiente. No se requiere descargar las políticas, algunas pueden estar ya en los peers. Todas las políticas de comportamiento del NetPeerGroup se encuentran precargadas en la plataforma JXTA. Los módulos son usados para representar partes de código que pueden ser dinámicamente cargados e instanciados en un peer para implementar un nuevo comportamiento. Aplicaciones, servicios de red y políticas de peer group son ejemplo de comportamientos que pueden ser instanciados en un peer representado por módulos. La Figura 5.5 muestra la actualización del grupo ESPOL.

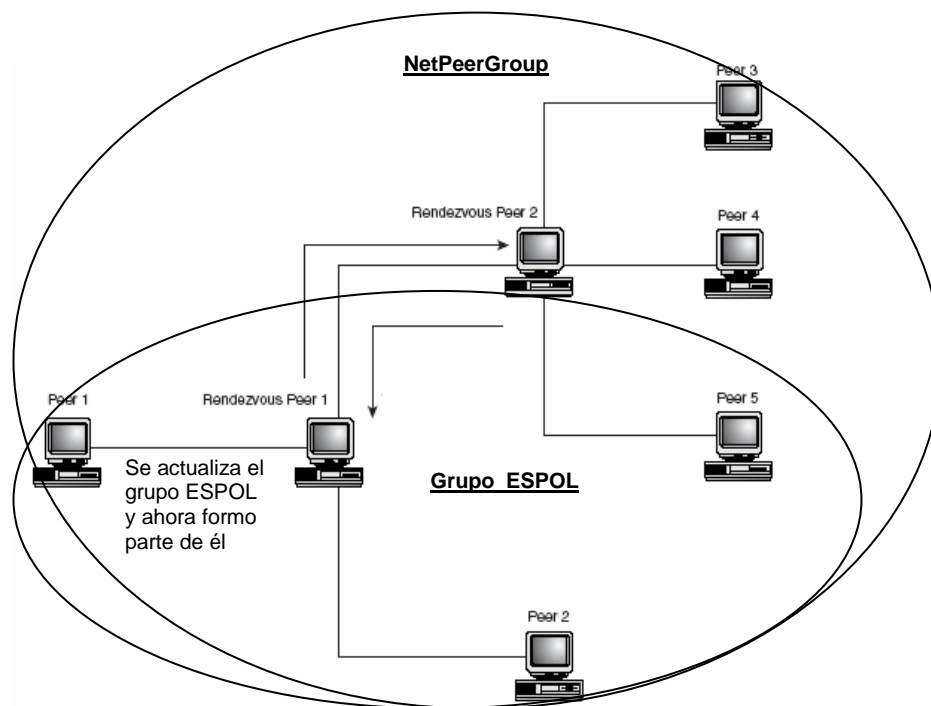


Fig. 5.5

Actualización del Grupo ESPOL

5.7 Servicio de membresías y credenciales

En contraste de la administración de un sistema centralizado, la administración en un sistema descentralizado puede ser muy compleja. Esto se debe a la dinámica y naturaleza de la red. Sin embargo el concepto de peer group puede reducir estas restricciones, ya que los servicios corriendo en cualquier peer son dependientes del peer group al que pertenecen.

Para manejar el ingreso al grupo de la ESPOL lo haremos a través de membresías. Todos los peers de JXTA interesados en el peer group se unen a él usando el servicio de membresía expuesto por el grupo. El trabajo del servicio de membresía es ayudar a autenticar un peer que se quiere unir al grupo. El protocolo de membresía impone y verifica requerimientos específicos para que un peer se una. Luego de que un peer se une exitosamente a un grupo puede tratar de ignorar algunas de las políticas de grupo y estas acciones no pueden ser controladas por un servicio de membresía. El primer mecanismo para evitar esto es validar a través de credenciales.

La credencial es un documento XML que es usado como una prueba de que un peer es miembro del grupo. El protocolo de membresía no lleva un registro de los usuarios que han ingresado. En un ambiente P2P es impráctico usar un servidor para validar la membresía. La credencial sirve como una manera de permitir a los peers reconocerse unos a otros como miembros válidos del mismo grupo de peers. Las credenciales de autenticación son creadas en el contexto de un peer group. Sin embargo son generalmente independientes de los grupos de peers. La intención es que la autenticación de credenciales sea pasada al servicio de membresías del mismo grupo de peers. El mecanismo para completar el objeto de autenticación es único para cada método de autenticación. Las

identidades asignan credenciales a los usuarios para acceder a los recursos de peer, mientras que la membresía de un grupo de peer es responsable de definir identidades aceptadas y autenticar peers que se quieren unir a un grupo.

Luego de aplicar para una membresía, el siguiente paso es unirse al grupo. Se pregunta si el objeto autenticador está completo y listo para ser enviado al servicio de membresía para unirse al grupo. Si todo está bien el método de `membership.join` es llamado. Este método retorna un objeto credencial. Cuando un peer se une a un peer group automáticamente busca un rendezvous para ese peer group. Si no encuentra un peer rendezvous, dinámicamente se vuelve él mismo un rendezvous para este peer group. El código de esta implementación es mostrado en el Apéndice C.6.

Cuando el peer está conciente de un peer group, ya sea por haberlo creado él mismo o realizando descubrimiento de peer group, en este caso es creado, se debe unir al grupo antes de cualquier comunicación que como parte de ese grupo pueda ocurrir.

5.8 Descubrimiento de Peers

El protocolo de descubrimiento de peers consiste de dos mensajes, un formato de petición para descubrir advertisements y un formato de respuesta para responder a una petición de descubrimiento. Estos dos formatos de mensajes, el mensaje de petición de descubrimiento y el mensaje de descubrimiento de respuesta definen todos los elementos requeridos para llevar a cabo una transacción de descubrimiento entre dos peers. La Figura 5.6 describe este proceso. El código de esta implementación se encuentra en el Apéndice C.7.

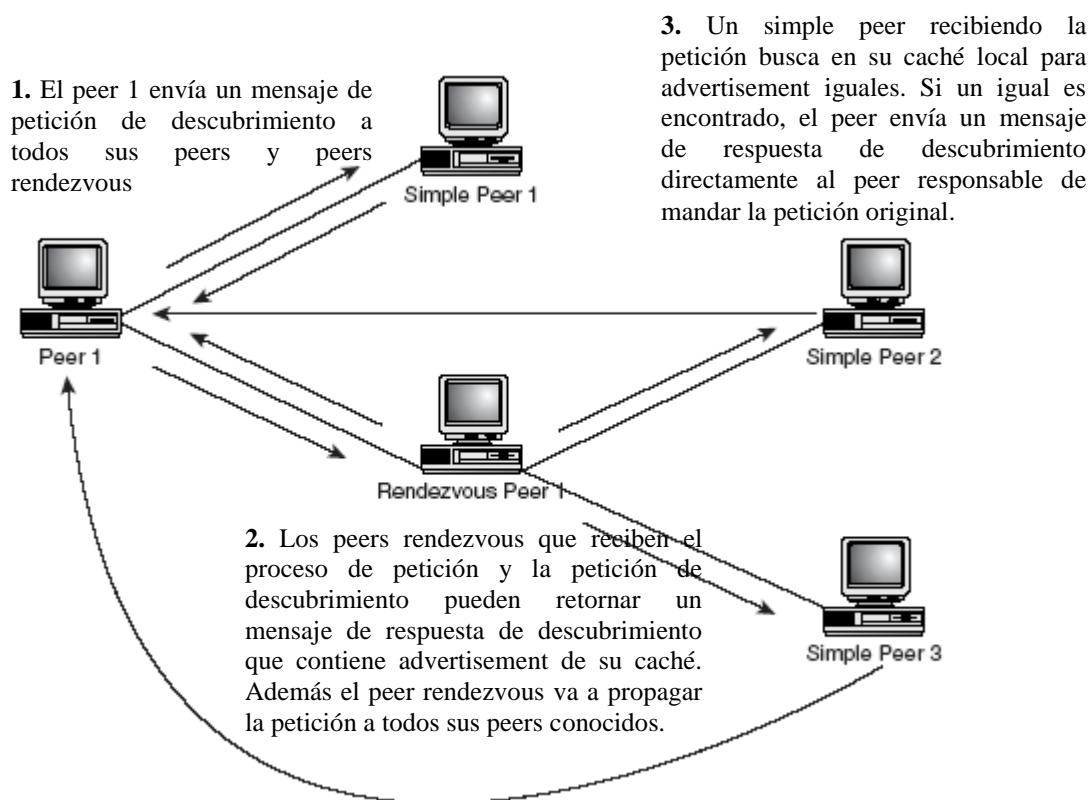


Fig. 5.6

Descubrimiento de Peers

5.9 Creación de Pipes

Para intercambiar datos, los peers deben emplear algún tipo de mecanismo para manejar la transmisión de datos a través de la red. Esta capa se llama la red de transporte. Es la responsable de todos los aspectos de transmisión de datos, incluyendo separar los datos en paquetes manejables, añadiendo cabeceras apropiadas a los paquetes para controlar su destino y en muchos casos se asegura que el paquete llegue a su destino. El transporte de red puede ser un transporte de nivel bajo, como UDP o TCP o uno de alto nivel como HTTP o **SMTP**. El concepto de un transporte de red en P2P puede ser dividido en tres partes:

Endpoints: La fuente inicial o destino final de cualquier dato transmitido a través de una red. Un endpoint corresponde la interfaz de red usada para enviar y recibir datos.

Pipes: Canales de comunicación virtuales, unidireccionales, asíncronos, conectando dos o más endpoints.

Mensajes: Contenedores de datos que se transmiten a través de un pipe desde un endpoint a otro.

Para comunicarse usando un pipe, un peer necesita encontrar los endpoints, uno para la fuente del mensaje y uno para cada destino del mensaje y conectarse conectando un pipe a cada uno de los endpoints. Cuando se une de esta manera, el endpoint actuando como una fuente de datos es llamado el pipe de salida y el endpoint actuando como repositorio de datos se llama pipe de ingreso. El pipe no es responsable de transmitir los datos entre los endpoints, es una abstracción usada para representar el hecho de que dos endpoints están conectados. Los endpoints proveen el acceso a la interfaz de red usada para transmitir y recibir datos.

Para enviar datos de un peer a otro, un peer empaqueta los datos a ser transmitidos en un mensaje y envía el mensaje usando un pipe de salida, en el otro extremo un peer recibe un mensaje de un pipe de ingreso y extrae los datos transmitidos.

La Figura 5.7 describe este proceso.

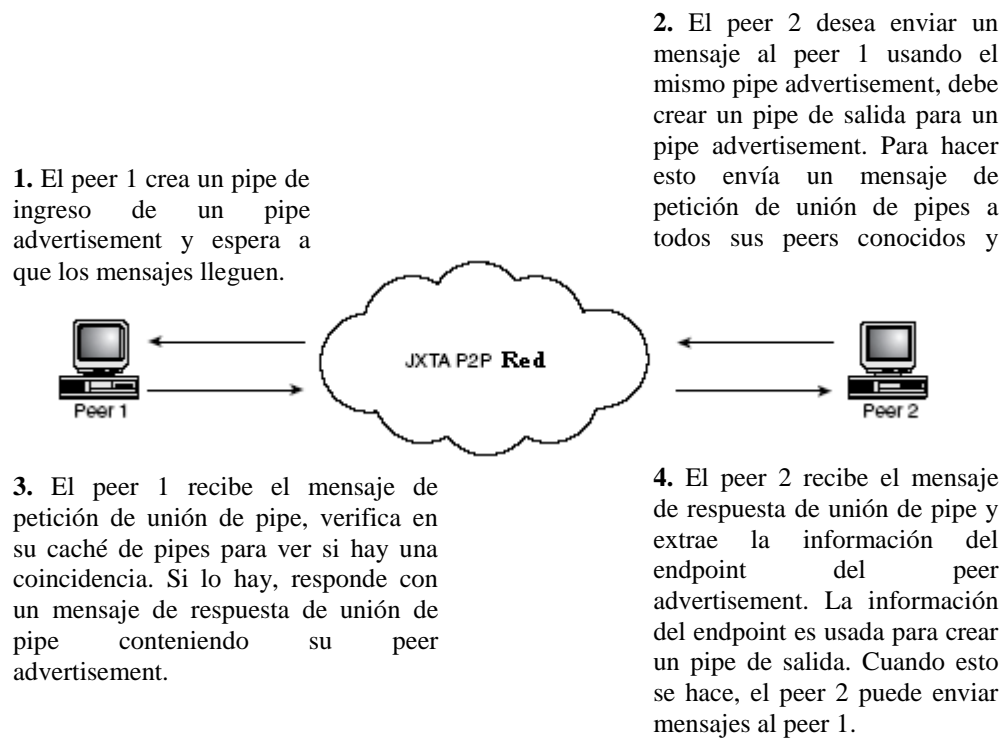


Fig. 5.7

Creación de Pipes

5.10 Pipe de Entrada de Mensajes

El código de esta implementación se encuentra en el Apéndice C.8.

5.11 Pipe de Salida de Mensajes

El código de esta implementación se encuentra en el Apéndice C.9.

5.12 Pipe de Entrada de Archivos

JxtaSocket es un pipe bidireccional que expone la interfaz de java.net.socket. JxtaSocket se comporta como un socket regular de java, con las siguientes diferencias (32):

JxtaSockets no implementan el algoritmo de Nagel (33), por lo que las aplicaciones deben enviar los datos al final de la transmisión o cuando las aplicaciones lo necesiten. La Figura 5.8 muestra esta característica.

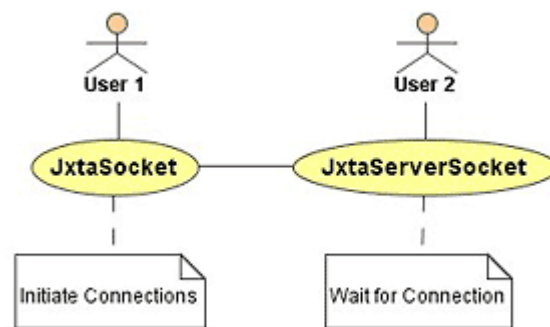


Fig. 5.8 (REF. 34)

El usuario 1 se conecta al usuario 2

El código de esta implementación se encuentra en el Apéndice C.10.

5.13 Pipe de Salida de Archivos

El código de esta implementación se encuentra en el Apéndice C.11.

5.14 Creando el servicio de Presencia

Para monitorear el estado de presencia de usuarios remotos crearemos el servicio de presencia. El estado permite a un usuario determinar si uno de sus contactos se encuentra en línea, desconectado, ocupado o temporalmente alejado de la computadora. La información de presencia de un peer va a ser publicada usando el servicio de Discovery, permitiendo que la información de presencia sea cacheada por otros peers. Se corre el riesgo de que la información de presencia cacheada esté desactualizada, pero esto puede ser resuelto publicando el advertisement con un tiempo de vida corto. Otra desventaja es que este método no es escalable a un gran número de peers ya que al existir muchos peers en la red todos deben actualizar los advertisement que hacen referencia al estado de sus contactos en periodos cortos. Esto generaría mucho tráfico en la red, pero es aceptable para una aplicación simple con una cantidad de usuarios no muy grande.

Para implementar el servicio de presencia es necesario:

Un advertisement: El servicio de presencia necesita un formato para que la información de presencia sea intercambiada con otros peers usando el servicio de Discovery.

5.14.1 El advertisement de presencia

Es responsable de describir el actual estado de presencia de un usuario en particular. Para identificar únicamente ambos peer y usuario, el advertisement de presencia necesita el peer ID más otro tipo de identificación que es única para el usuario.

Para implementar el advertisement de presencia se define una clase abstracta derivada de la clase `net.jxta.document.Advertisement`. El código de esta implementación se encuentra en el Apéndice C.12.

5.14.2 Publicar el Advertisement de presencia

Es responsable de publicar el advertisement del actual estado de presencia de un usuario en particular. El código de esta implementación se encuentra en el Apéndice C.13.

5.14.3 Descubrimiento de Advertisement de presencia

Es responsable de descubrir el actual estado de presencia de un usuario en particular. Lo utiliza el usuario que desea actualizar los estados de sus contactos.

Para el descubrimiento de advertisement de presencia se define una implementación de la clase abstracta PresenceAdvertisement. El código de esta implementación se encuentra en el Apéndice C.14.

5.14.4 Actualizar el Advertisement de presencia

Es responsable de actualizar el actual estado de presencia de un usuario en particular. Lo utiliza el usuario que desea actualizar su propio estado en un advertisement. El código de esta implementación se encuentra en el Apéndice C.15.

5.15 Creando el servicio de Cartelera Comunal

Para incentivar la colaboración entre los usuarios de la red par-a-par de la ESPOl desarrollamos la cartelera comunal. Ésta es un espacio en el cual los usuarios pueden publicar o consultar mensajes escritos por otros usuarios. Para esto creamos el servicio de cartelera. Los datos del anuncio, como lo son, usuario, asunto, fecha/hora y mensaje, son escritos en un Advertisement y publicados usando el servicio de Discovery, permitiendo que la información sea guardada en la caché por

otros peers y replicada al resto. Con este propósito utilizamos advertisements de JXTA de vida larga los cuales permanecen en la caché de los peers por dos semanas. Para facilitar la consulta de los anuncios en la cartelera, se decidió que cada uno esté etiquetado con una estampa de tiempo indicando su hora de publicación. Pero en un sistema P2P, no hay un reloj que sea aplicable a todo el sistema, ya que cada nodo tiene su propio reloj, y puede darse el caso que uno o más nodos tengan configurada la hora del sistema de manera incorrecta. Para evitar inconvenientes de anuncios con estampas de tiempo incorrectas, la hora de publicación del anuncio de cartelera la obtenemos contactando al servidor SMTP del host smtp.espol.edu.ec. Para implementar el servicio de cartelera es necesario:

Un advertisement: El servicio de cartelera necesita un formato para que la información de presencia sea intercambiada con otros peers usando el servicio de Discovery.

5.15.1 El advertisement de cartelera

Es responsable de publicar los nuevos mensajes en la cartelera.

Para implementar el advertisement de presencia se define una clase abstracta derivada de la clase

net.jxta.document.Advertisement. El código de esta implementación se encuentra en el Apéndice C.16.

5.15.2 Implementación del advertisement de cartelera

El código de esta implementación se encuentra en el Apéndice C.17.

5.15.3 Publicar el advertisement de cartelera

El código de esta implementación se encuentra en el Apéndice C.18.

5.15.4 Descubrimiento de advertisement de Lista de Contactos

Es responsable de colocar un nuevo mensaje en la cartelera. Lo utiliza el usuario que desea enviar un nuevo mensaje a la cartelera comunal. El código de esta implementación se encuentra en el Apéndice C.19.

5.16 Lista de Contactos

En un diseño preliminar, nos encontramos con la situación de que cuando un usuario se conectaba a nuestra aplicación desde otra computadora que no era la suya, no obtenía la lista de contactos ya que la misma se almacena en su computador. En una aplicación cliente/servidor la solución es sencilla ya que la lista de contactos se puede almacenar en el servidor. Pero en una aplicación P2P no existe un servidor central que pueda administrar las listas de contactos de los usuarios. Una solución es que cada vez que se haga una modificación en la lista se publique un advertisement con los cambios. Pero se corre el riesgo de que la próxima vez que el usuario se conecte no obtenga la última lista publicada sino una versión anterior y se le presenten resultados viejos, ya que el peer que la tiene en su caché no se encuentra conectado o no replicó la lista lo suficiente a sus demás peers para tener redundancia. En consecuencia tomamos la decisión de no incorporar la opción de almacenar la lista de contactos en un peer externo. Como solución alterna, se optó por permitir exportar la lista y posteriormente importarla. La lista puede ser exportada en un documento XML y almacenada en el disco duro o algún otro dispositivo externo. Pero exportar e importar la lista cada vez que el usuario se conecta resulta molesto, y además, puede que cuando el usuario se conecte no tenga a la mano el dispositivo en donde almacenó la lista de contactos y por lo tanto no los pueda importar. Este problema lo

solucionamos respaldando la lista en un correo electrónico enviado a la cuenta de e-mail de la ESPOL del usuario. De esta manera al abrir la aplicación la lista puede ser recuperada y cuando se cierra la aplicación, puede ser actualizada.

El proceso es el siguiente: al cerrar una sesión nuestra aplicación crea un mensaje de correo que contiene la lista de contactos, el cual se envía a la propia cuenta de e-mail del usuario, abriendo un socket al puerto 25 del host smtp.espol.edu.ec. En el título del correo se especifica que es un mensaje que no debe ser borrado si se desea preservar la lista de contactos. Luego al iniciar una sesión, nuestra aplicación utiliza el mismo usuario y contraseña de autenticación para conectarse al servidor POP con el puerto 110 (pop.espol.edu.ec) para hacer una búsqueda en los mails del usuario y rescatar la lista de contactos. Cabe recalcar que esta información es enviada por un canal seguro, utilizando SSL. Si la lista de contactos no se puede recuperar del correo del usuario, se intenta recuperar la última lista que el usuario dejó en la computadora que está utilizando. Si no se recupera ninguna, se presenta una lista vacía, la cual puede ser llenada si el usuario la almacenó anteriormente, gracias a la opción de exportar/importar que le brindamos.

5.17 Búsqueda de contactos por usuario

Se puede realizar una búsqueda de contactos por su usuario de la cuenta ESPOL. El código de esta implementación se encuentra en el Apéndice C.20.

5.18 Comunicación de uno a muchos

Para la implementación del envío de un mismo mensaje a todo un grupo de conversación en la opción de chat de uno-a-muchos utilizamos una combinación de multicast ingenuo con diseminación por “chismes”, en la cual el origen envía una copia del mensaje a cada miembro del grupo (35). Esta solución es adecuada y eficiente para nuestra aplicación porque en una conversación uno a muchos no se suele involucrar una gran cantidad de usuarios en el grupo. Esta solución “ingenua” no es escalable a grupos grandes, pero esto no representa un problema ya que el escenario planteado no lo requiere. Si un usuario desea agregar contactos a una conversación éstos serán parte del árbol teniendo como nodo anfitrión al nodo que los invocó. En la Figura 5.9 se muestra cuando el usuario 1 agrega tres contactos a una conversación. Luego el usuario 3 agrega 2 usuarios más a la conversación.

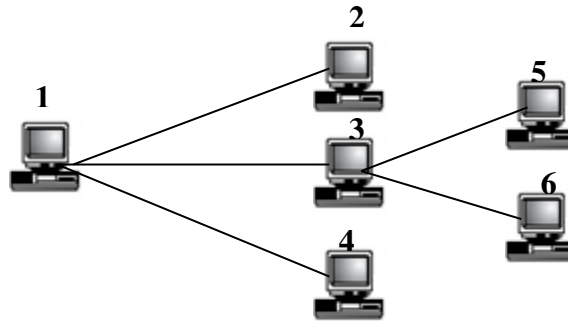


Fig. 5.9

Contactos agregados a una conversación

Cuando el usuario 1 envía un mensaje se entrega una copia a cada uno de sus contactos invocados, y el usuario 3 hace lo mismo con sus contactos invocados. La Figura 5.10 muestra esta característica.

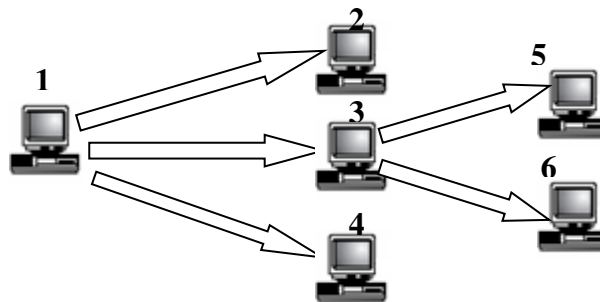


Fig. 5.10

Envío de mensajes desde el nodo invocador

Cuando un usuario que ha sido agregado por otro nodo desea enviar un mensaje, lo envía a todos los nodos conectados a él, es decir al nodo invocador y los nodos que él invocó. El nodo que lo invitó a la

conversación se encarga de enviar una copia de ese mensaje a todos los nodos que fueron invocados por él menos al nodo del cual lo recibió y así mismo se comportan todos los nodos que reciban el mensaje. Por ejemplo, el nodo 2 al recibir el mensaje tratará de enviar una copia al nodo que lo invocó, a los nodos que él ha invocado pero no al nodo que lo envió, entonces como no ha invocado a ningún nodo y el nodo que lo invocó es el mismo que le envió el mensaje, no envía una copia a ningún nodo. La Figura 5.11 demuestra esta característica.

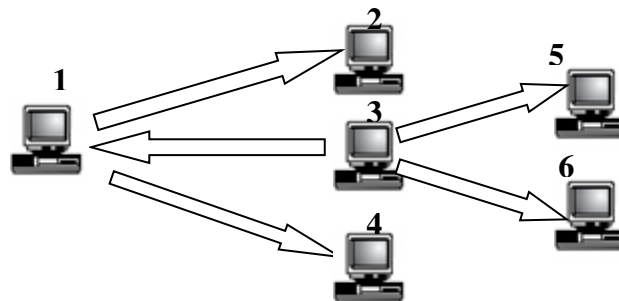


Fig. 5.11

Envío de mensajes desde un nodo invocado

Para hacerlo tolerante a fallos, si un nodo se desconecta, todos los nodos que dependen de él se desconectarán también. Pero se les envía un mensaje a los nodos dependientes en el cual se les indica que el nodo invocador fue desconectado y que si quieren ingresar de nuevo al grupo de conversación deben de ser invitados por alguien que todavía se encuentre en el grupo.

CAPÍTULO 6

PRUEBAS DE ESCALABILIDAD

En Jxta.org existe un proyecto que sirve como un repositorio central para **benchmarks** y artefactos de pruebas, para de esta manera mejorar el desempeño y escalabilidad de las implementaciones de JXTA (36).

Para poder medir la capacidad de escalabilidad de JXTA, un ambiente de benchmarking fue construido en el laboratorio inalámbrico de Ingeniería de Sun Microsystems en Menlo Park, California. Dado los varios tipos de redes y configuraciones, plataformas, etc, el número de varios ambientes de pruebas puede crecer. El ambiente de prueba del benchmarking en el laboratorio inalámbrico de Sun consiste en una variedad de hardware corriendo Solaris 8, Solaris 9 y Linux. Una máquina controladora llamada jxta03 corre Solaris 9 en una E450 con 4 procesadores va a ser usada para simular la distribución de peers y para controlar las prueba de benchmarks.

Los esfuerzos del Jxta Benchmarking se pueden clasificar en las siguientes categorías y en el orden de prioridades:

1. Carga/ Pruebas de stress

2. Tiempos de round trip para mensajes
3. Prueba de desempeño de un peer independiente
4. Una aplicación de la vida real

6.1 Pruebas de Carga/ Pruebas de stress

Es para medir los límites de la carga Jxta que los súper peers pueden tolerar mientras escalan razonablemente bien. El término razonablemente bien es definido como manejo de carga sin perder ningún mensaje.

A continuación hay algunos de los escenarios de prueba de carga.

6.1.1 Prueba de carga Discovery

Esto prueba un escenario donde un número de pedidos de discovery son enviados al mismo peer Rendezvous desde varios peers simultáneamente y periódicamente. Los pedidos de discovery son generados aleatoriamente para evitar encontrar advertisements cacheados. Esto debe ser repetido al variar el número de peers rendezvous donde otros peers rendezvous también envían pedidos de discovery al rendezvous bajo ataque.

Resultado esperado: Cuántos pedidos de descubrimiento simultáneos pueden ser manejados por un peer rendezvous.

Versión	Tiempo Máximo	Tiempo Mínimo	Tiempo Promedio	Queries Ignorados	Número de Queries
2.3	111743	56	35777.37	N/C	17429

Tabla I

Tabla de Prueba de carga Discovery a un peer Rendezvous

4 peers rendezvous (uno es contribuyente), 5 TCP peers, 10 minutos de prueba.

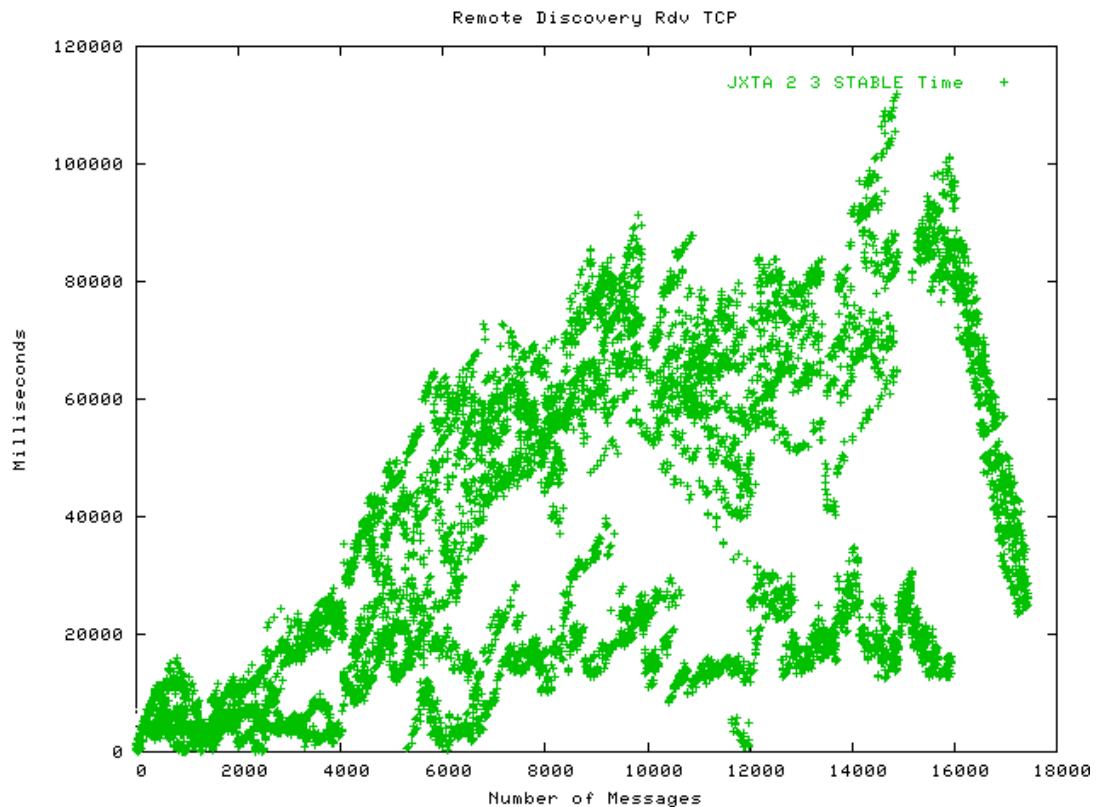


Fig. 6.1 (REF. 36)

Prueba de carga Discovery a un peer Rendezvous

Como podemos ver en la Figura 6.1, se observa una tendencia lineal que al aumentar el número de mensajes discovery hacia un rendezvous aumenta el tiempo de respuesta. Pero aún así con una gran cantidad de mensajes como 17429 podemos ver que el tiempo máximo de respuesta fue de 111743 milisegundos al enviarse 17000 mensajes, lo que consideramos aceptable para esta clase de aplicación universitaria. Así mismo el menor tiempo

fue de 56 milisegundos cuando los mensajes enviados eran casi mínimos.

6.1.2 TCP Relay

Esto mide el número de conexiones simultáneas que un **TCP relay** puede sostener antes de empezar a ignorar mensajes. TCP relay es un simple tipo de retransmisión. Un relay escucha en un puerto en una cierta dirección IP en el firewall y replica todo el tráfico al servidor especificado. Mientras hay conexión cada peer está ocupado enviando mensajes de JXTA, conectándose a un rendezvous, enviando chat, etc. TCP relay es una nueva característica añadida bajo las actividades de JXTA actuales.

Resultado esperado: Número de peers que un TCP relay puede soportar.

Versión	Tiempo Máximo	Tiempo Mínimo	Tiempo Promedio	Queries Ignorados	Número de Queries
2.3	94358	109	33826.64	2835	18686

Tabla II

Tabla de Prueba de carga Discovery a un peer Rendezvous por TCP relay

4 peers rendezvous (uno es contribuyente), 5 TCP peers, 10 minutos de prueba.

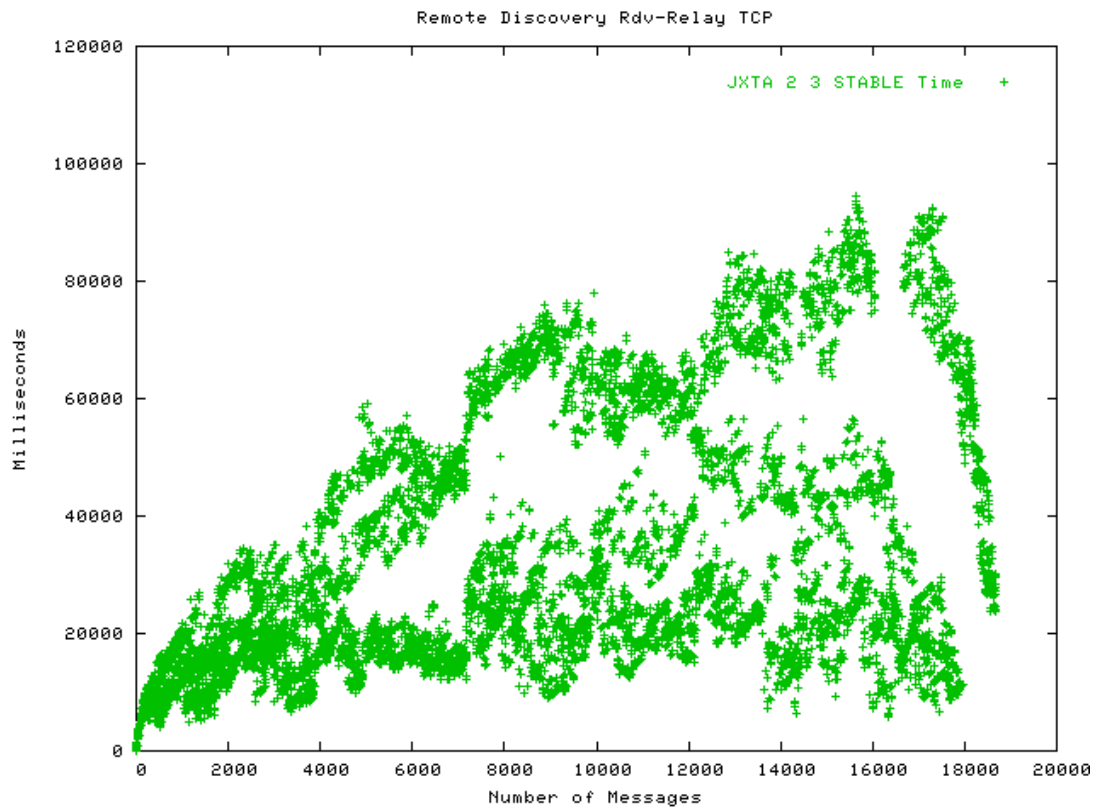


Fig. 6.2 (REF. 36)

Prueba de carga Discovery a un peer Rendezvous por TCP relay

En la gráfica se observa una tendencia lineal que al aumentar el número de mensajes discovery hacia un rendezvous aumenta el tiempo de respuesta de TCP relay. Con una gran cantidad de mensajes como 18686 podemos ver que el tiempo máximo de respuesta fue de 94358 milisegundos al enviarse 18500 mensajes.

Así mismo el menor tiempo fue de 109 milisegundos cuando los mensajes enviados fueron casi mínimos.

6.2 Pruebas de Tiempos de Round Trip

Esta clase de prueba de benchmark mide el tiempo round trip que un mensaje lleva en la red JXTA.

6.2.1 Pipes Unicast

Se realizan pruebas de latencia en los cuales se mide el tiempo que le toma a un mensaje ir de un punto designado a otro.

Versión	Tiempo Máximo	Tiempo Mínimo	Tiempo Promedio	Número de Mensajes
2.3	69970	27	9332.62	108576

Tabla III

Tabla de Pruebas de latencia de Pipes Unicast entre Peers

5 TCP peers, 10 minutos de prueba.

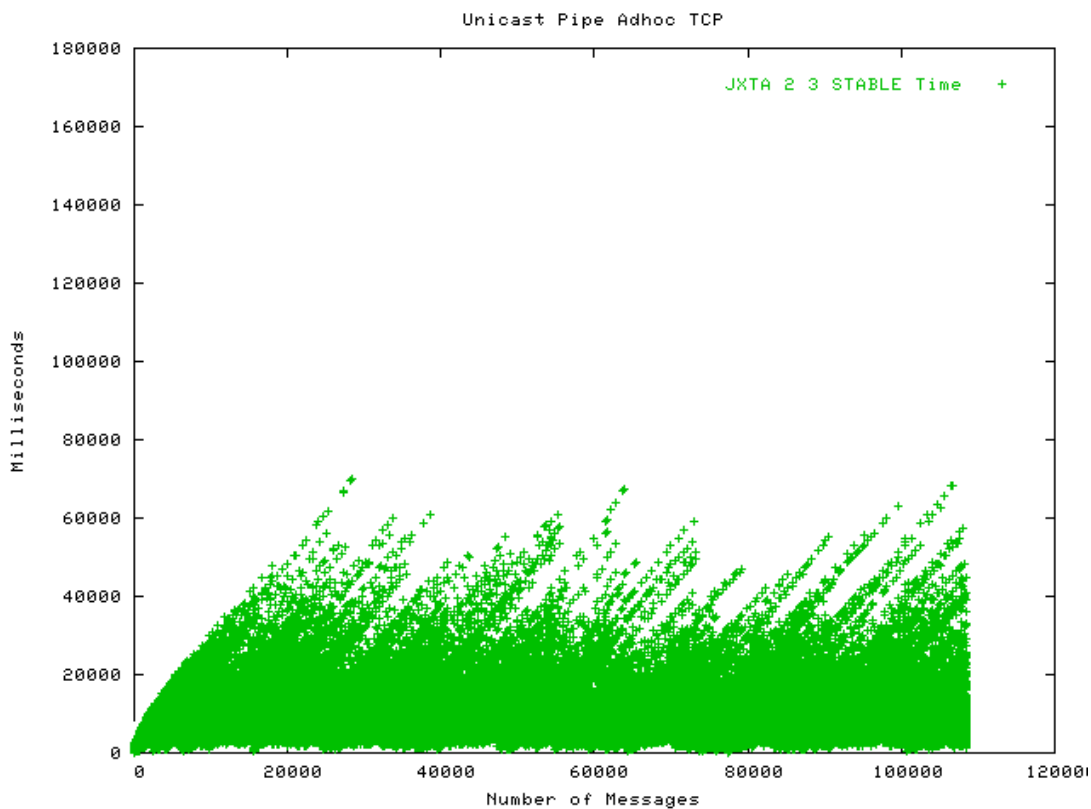


Fig 6.3 (REF. 36)

Pruebas de latencia de Pipes Unicast entre Peers

Como se puede observar en la Figura 6.3, a medida que aumentan los mensajes enviados el tiempo de respuesta no crece demasiado. Se obtiene un tiempo mínimo de 27 milisegundos y uno máximo de 69970 milisegundos. Con la cantidad de 108576 mensajes se obtuvo un tiempo promedio de 9332.62 milisegundos. Esto muestra la escalabilidad de la red al aumentar la carga de los mensajes en la red JXTA.

Versión	Tiempo Máximo	Tiempo Mínimo	Tiempo Promedio	Número de Mensajes
2.3	50298	6	5105.58	60020

Tabla IV

Tabla de Pruebas de latencia de Pipes Unicast entre Rendezvous y Peers

4 peers rendezvous (uno es contribuyente), 5 TCP peers, 10 minutos de prueba.

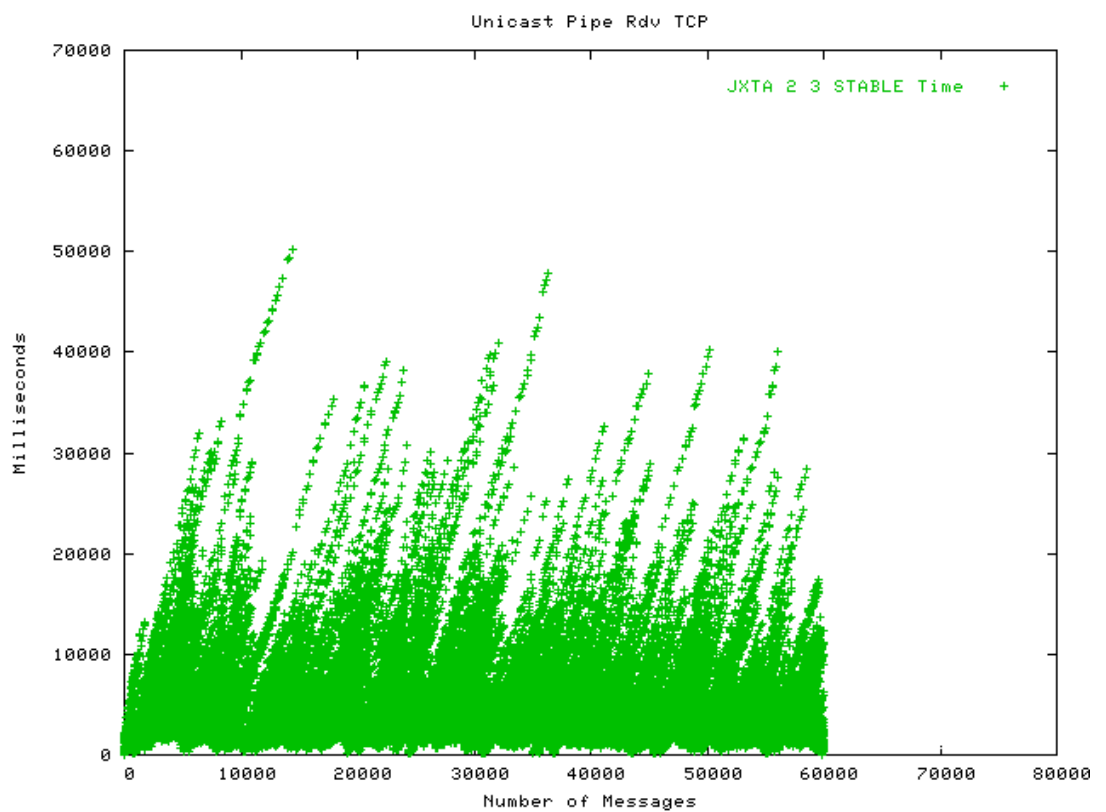


Fig. 6.4 (REF. 36)

Pruebas de latencia de Pipes Unicast entre Rendezvous y Peers

Como se puede observar en la Figura 6.4, a medida que aumentan los mensajes enviados el tiempo de respuesta no crece demasiado. Se obtiene un tiempo mínimo de 6 milisegundos y uno máximo de 50298 milisegundos. Con la cantidad de 60020 mensajes se obtuvo un tiempo promedio de 5105.58 milisegundos. Esto muestra la capacidad de los peers al aumentar la carga de los mensajes en la red JXTA hacia un rendezvous.

6.3 Pruebas independientes

Estas son pruebas independientes para medir tiempos de actividades independientes. La mayoría de estas son implementadas en un bajo nivel o como pruebas de unidad de benchmark.

Fueron llevadas a cabo en un equipo de 80GB de disco duro, 1024 MB de memoria de sistema, 1.8 GHz y Procesador AMD Turion 64. En la red existieron cuatro nodos.

6.3.1 Tiempo de Parsing de un XML

XML	# de Contactos	Tamaño	Tiempo de Parsing
Lista de contactos default (sin contactos)	0	1 kb	15 ms
Lista de contactos	20	4 kb	32 ms
Lista de contactos	44	8 kb	63 ms

Tabla V

Tabla de Pruebas de Tiempo de Parsing de un XML

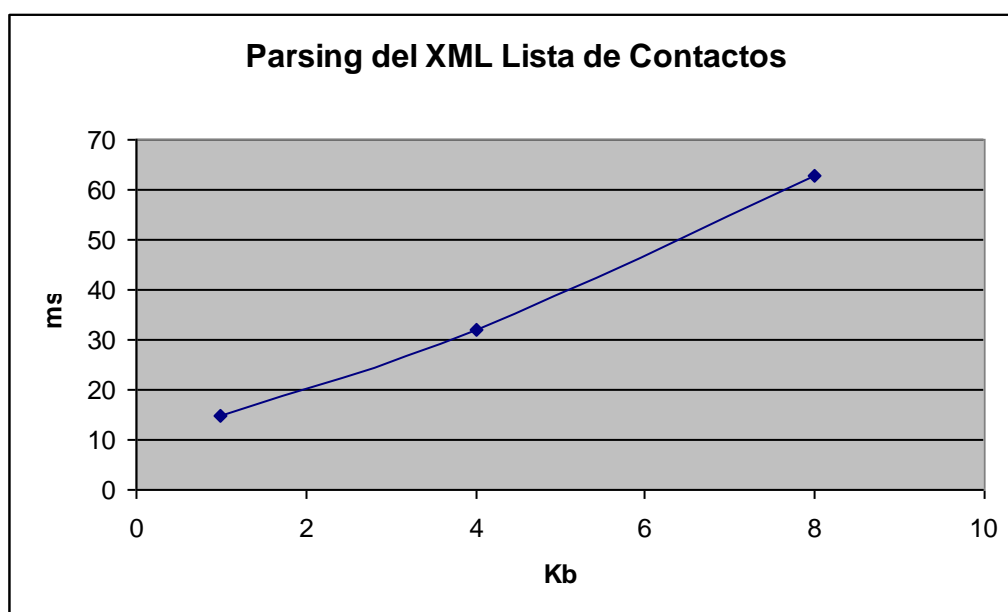


Fig. 6.5

Pruebas de Tiempo de Parsing de un XML

Con los datos de la tabla V se elaboró el gráfico de la Figura 6.5, en el cual puede observar una tendencia lineal. Al aumentar el

tamaño de un archivo XML aumenta el tiempo de parseo del mismo. Si doblo su tamaño se dobla el tiempo de su recorrido.

6.3.2 Tiempo a conectarse a un peer rendezvous

# intento	Tiempo en milisegundos (ms)
1	94265
2	127047
3	82547
4	105672
5	105797
6	82218
7	91843

Tabla VI

Tabla de Pruebas de Tiempo a conectarse a un peer rendezvous

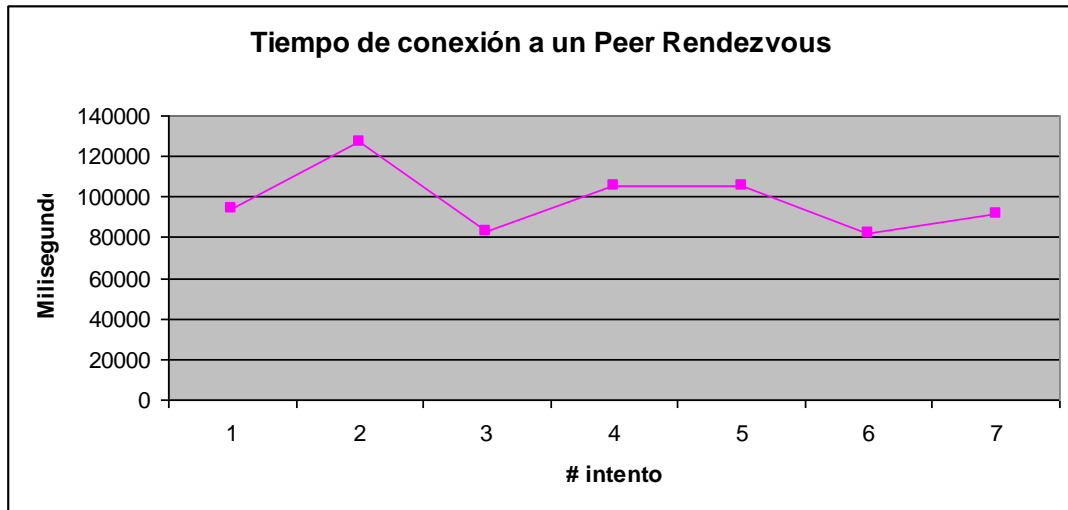


Fig. 6.6

Pruebas de Tiempo a conectarse a un peer rendezvous

Con los datos de la tabla VI se elaboró el gráfico de la Figura 6.6, en la cual se puede observar que en cada intento de conexión se obtuvo un tiempo entre 80000 y 140000 milisegundos. Los cuales son tiempos aceptables a esperar para conectarse a un peer rendezvous. Además de las pruebas realizadas, el promedio de tiempo que toma conectarse a un peer rendezvous es 98484 milisegundos. Cabe recalcar que para que éste resultado sea más confiable se deberían hacer más intentos para asegurarse que la muestra sea realmente representativa.

6.3.3 Tiempo a unirse a una red JXTA

El tiempo a unirse a la red JXTA es casi nulo ya que implica instanciar el NetPeerGroup por mi mismo. Es decir creo el grupo para formar parte de él. Luego si quiero actualizar la información de advertisement de algún peer Rendezvous esto ya se considera como Discovery.

CAPITULO 7

PRUEBAS DE USUARIO

7.1 Usabilidad

La definición de usabilidad conforme a la norma ISO 9241, parte 11 dice:

"la usabilidad es el rango en el cual un producto puede ser usado por unos usuarios específicos para alcanzar ciertas metas especificadas con efectividad, eficiencia y satisfacción en un contexto de uso especificado".(37)

De hecho, la usabilidad no se limita a sistemas computacionales exclusivamente, sino que es un concepto aplicable a cualquier elemento en el cual se va a producir una interacción entre un humano y un dispositivo.

En el caso de los sistemas computacionales, la usabilidad va a abarcar desde el proceso de instalación de la aplicación hasta el punto en que el sistema sea utilizado por el usuario, incluyendo también el proceso de mantenimiento.

7.2 Pruebas de usuario

Las pruebas de usuario permiten verificar la usabilidad de los sistemas, se basan en la observación de las técnicas que emplean diferentes personas al utilizar el sistema, y en el contraste de su experiencia de utilización del mismo. De esta manera, podemos distinguir las funciones que podrían estar causando problemas o confusión en los usuarios, ya sea por la manera en que se presentan o por el lenguaje utilizado en el mismo. Con la ayuda de los usuarios, a través de las encuestas y sus consejos, podemos mejorar la interfaz y la interacción para que nuestra aplicación demuestre buena usabilidad. No hay nadie mejor que los usuarios de la aplicación para observar errores o signos de una mala interacción.

Para su realización, seleccionamos a un grupo de usuarios atendiendo a los objetivos del proyecto. Personas con edades entre 17 y 52 años que representan a estudiantes futuros, novatos, estudiantes avanzados y trabajadores de la ESPOL. La Tabla VII muestra la información detallada de los usuarios.

Usuario	Nombre	Edad	Formación
1	Víctor Cedeño	17	Estudiante de quinto curso, Colegio Americano
2	Jaime Cedeño	21	Estudiante de Universidad, 6to semestre
3	Manuel Cedeño	52	Ing. eléctrico
4	Xavier Suárez	20	Estudiante de Universidad, 4to semestre
5	Luís Chiang	20	Estudiante de Universidad, 6to semestre
6	Luis Calle	24	Estudiante de Universidad, 8vo semestre

Tabla VII

Usuarios de prueba

Utilizamos cuestionarios ya que es una técnica para obtener datos demográficos y la opinión directa de los usuarios. Desarrollamos dos cuestionarios con enfoques diferentes. El primero se encuentra en el Apéndice D.1 y lo llamamos cuestionario de funcionalidades. Divide cada caso de uso en Tareas y así mismo, cada Tarea en Subtareas. De esta manera podemos distinguir específicamente en que parte de una función específica de la aplicación el usuario tiene problemas. El segundo cuestionario se encuentra en el Apéndice D.2 y trata sobre la interfaz en general. Contiene preguntas directas para identificar posibles causas de

problemas o confusión en los usuarios, ya sea por la estética en que se presentan o por el lenguaje utilizado en el mismo.

La evaluación de estos cuestionarios se discute en el sub-capítulo 7.4, Evaluación de pruebas, y las conclusiones a las que llegamos en el sub-capítulo 7.5, Mejoras en el proyecto.

7.3 Atributos de usabilidad

La usabilidad tiene cinco atributos definidos:

- **Facilidad de aprendizaje.** ¿Cuánto le toma al usuario típico de una comunidad aprender la manera en como se usan los comandos relevantes a un conjunto de tareas? Se refiere a que tan rápido el usuario va a aprender a usar un sistema con el cual no había tenido contacto previamente. Este punto se refiere a la consecución de tareas básicas por parte de un usuario novato.
- **Velocidad de desempeño.** ¿Cuánto le toma a un usuario completar un grupo de tareas específicas (benchmark tasks)? Una vez que el usuario ha aprendido a utilizar el sistema, se va a ponderar el lograr la velocidad con que puede completar una tarea específica.

- **Tasas de error por parte de los usuarios.** ¿Cuántos y qué errores comete el usuario al ejecutar un grupo de tareas específicas? Este apartado apunta hacia los errores cometidos por el usuario. Este atributo se refiere a aquellos errores que comete el usuario al utilizar el sistema. Una aplicación ideal evitaría que el usuario cometiera errores y funcionaría de manera óptima a cualquier petición por parte del usuario. En la práctica esto difícilmente se logra. Es vital que una vez que se produzca un error el sistema se lo haga saber rápida y claramente al usuario, le advierta sobre la severidad del mismo y le provea de algún mecanismo para recuperarse de ese error.
- **Retención sobre el tiempo.** ¿Qué tan bien recuerdan los usuarios la manera en como funciona el sistema después de una hora, un día o una semana? Cuando un usuario ha utilizado un sistema tiempo atrás, y tiene la necesidad de utilizarlo de nuevo la curva de aprendizaje debe de ser significativamente menor que el caso del usuario que nunca haya utilizado dicho sistema. Esto es de primordial importancia para aplicaciones usadas intermitentemente.
- **Satisfacción.** ¿Qué tanto le gustaron a los usuarios los distintos atributos del sistema? Este atributo se refiere a la impresión subjetiva del usuario respecto al sistema.

Quisimos realizar una investigación profunda acerca de la usabilidad de nuestro sistema. Aparte de los cuestionarios que se presentaron en el sub-capítulo anterior realizamos pruebas de observación. Estas consistían en monitorear a los usuarios realizando funciones específicas las cuales creemos son las más relevantes de nuestra aplicación. Estas pruebas cuales son descritas en la Tabla VIII. Para la creación de esta tabla escogimos los tres atributos de usabilidad que consideramos más importantes para medir la usabilidad de nuestro sistema. Con un instrumento de medición adecuado obtenemos resultados los cuales van a ser comparados con los resultados esperados. Luego se trata de mejorar la interfaz para mejorar los resultados en la segunda ronda de pruebas. De esta manera volver a medir y verificar si se ha obtenido una mejora en los resultados y en el desempeño del usuario hacia nuestro sistema que es el objetivo principal de las encuestas.

Atributo de usabilidad	Instrumento de medición	Valor a medir	Nivel actual	El peor nivel aceptable	Nivel destino planeado	Mejor nivel posible	Resultados Observados
Facilidad de Aprendizaje	Cronómetro	Cuánto tiempo le toma aprender a chatear con nuestra aplicación.	-	97 seg	60 seg	23 seg	40 seg
Retención sobre el tiempo	Cronómetro	Envío de archivo	-	20 seg	10 seg	5 seg	6 seg
Satisfacción subjetiva	Encuesta	Opinión de los usuarios hacia la aplicación.	-	Nada satisfactorio	Muy satisfactorio	Muy satisfactorio	Muy satisfactorio

Tabla VIII

Pruebas de Usabilidad 1

La Facilidad de Aprendizaje es un atributo de usabilidad crítico para nuestra aplicación, ya que los usuarios de la institución al descargar la aplicación no tendrán a nadie que les explique cómo se deben desenvolver al utilizarla. Por esto la usabilidad del sistema debe ser buena. El valor a medir que escogimos fue, “¿Cuánto tiempo le toma aprender a chatear con nuestra aplicación?”, que es la funcionalidad principal de una aplicación de mensajería instantánea. Realizamos la prueba con un cronómetro como instrumento de medición. Obtuvimos un tiempo de 40 segundos por parte de los usuarios el cual superó a los 60 segundos del nivel de destino planeado. Pero todavía hay oportunidad

de mejorar para llegar al mejor nivel posible que es de 23 segundos y nosotros consideramos el tiempo ideal necesario para aprender a chatear en nuestra aplicación.

También creímos esencial medir y mejorar el atributo de usabilidad Retención Sobre el Tiempo, ya que la aplicación de mensajería instantánea va a ser utilizada esporádicamente por ciertos usuarios. Por esto es muy importante que los usuarios no memoricen los procedimientos para realizar una función específica sino que los identifiquen fácilmente, sin importar el lapso de tiempo que ha pasado desde la última vez que utilizó la aplicación o el número de veces que la ha utilizado.

El valor a medir que escogimos fue, “Envío de archivo”, es decir cuánto tiempo le toma al usuario realizar la acción de enviar un archivo a otro contacto. Realizamos la prueba con un cronómetro como instrumento de medición. Obtuvimos un tiempo de 6 segundos por parte de los usuarios el cual superó a los 10 segundos del nivel de destino planeado. Pero todavía hay oportunidad de mejorar para llegar al mejor nivel posible que es de 5 segundos y nosotros consideramos el tiempo ideal para enviar un archivo en nuestra aplicación.

Por último escogimos la Satisfacción como atributo de usabilidad esencial para la mejora del proyecto. De esta manera nos aseguramos que el usuario se sienta satisfecho con la aplicación la cual fue pensada para él. El valor a medir que escogimos fue la, “Opinión de los usuarios hacia la aplicación”. Realizamos la prueba con la ayuda de una encuesta como instrumento de medición. Esta encuesta es la que se encuentra en el Apéndice D.2. Obtuvimos una respuesta de “Muy satisfactorio” por parte de los usuarios el cual igualó al nivel de destino planeado y al mejor nivel posible. Lo que nos dice que la aplicación sí gusta a los usuarios que la utilizan.

Atributo de usabilidad	Instrumento de medición	Valor a medir	Nivel actual	El peor nivel aceptable	Nivel destino planeado	Mejor nivel posible	Resultados Observados
Facilidad de Aprendizaje	Cronómetro	Cuánto tiempo le toma aprender a chatear con nuestra aplicación.	40 seg	97 seg	60 seg	23 seg	25 seg
Retención sobre el tiempo	Cronómetro	Envío de archivo	6 seg	20 seg	10 seg	5 seg	5 seg
Satisfacción subjetiva	Encuesta	Opinión de los usuarios hacia la aplicación.	Muy satisfactorio	Nada satisfactorio	Muy satisfactorio	Muy satisfactorio	Muy satisfactorio

Tabla IX

Pruebas de Usabilidad 2

Como lo demuestra la Tabla IX, la segunda vez que se realizaron las pruebas de usabilidad, pudimos obtener mejoras en los tiempos de medición. El atributo Facilidad de Aprendizaje obtuvo un tiempo de 25 segundos, muy cerca del mejor nivel posible. Esto nos dice que la usabilidad de nuestra aplicación se encuentra en un buen nivel. El atributo Retención sobre el Tiempo obtuvo el mejor nivel posible de 5 segundos y la Satisfacción Subjetiva se mantuvo en “Muy Satisfactoria”.

7.4 Evaluación de pruebas

Luego de realizar la encuesta del Apéndice D.1, llamada cuestionario de funcionalidades, e introducida en el sub-capítulo 7.2, obtuvimos resultados muy interesantes los cuales transformamos en un gráfico de barras para mejorar su visualización. Cada subtarea puede obtener una calificación máxima de 100%, la cual indica que los usuarios no tuvieron problema alguno al desempeñarla, por eso nos fijamos en las que no obtuvieron la calificación deseada ya que significaron alguna dificultad.

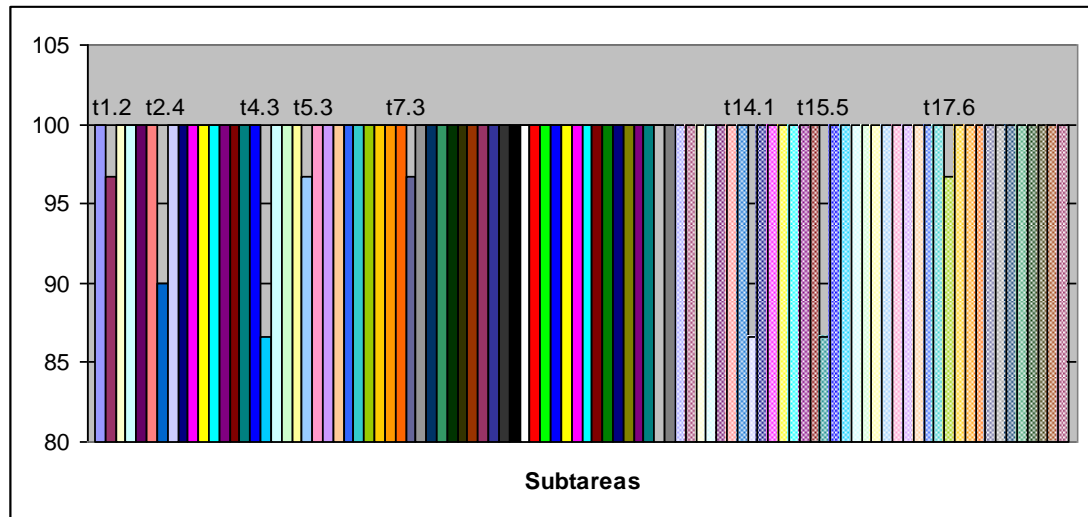


Fig. 7.1

Análisis de Subtareas

Mediante al gráfico en la Figura 7.1 observamos específicamente las subtareas que representaban un mayor grado de dificultad para los usuarios, las cuales fueron 8. El resto de subtareas obtuvo un resultado ideal lo que nos indica que no representan ningún problema para el usuario común.

Interrogamos al usuario sobre la razón por la cual algunas subtareas representaban un problema, esto nos permitió discutir, identificar la causa, mejorar la interacción y solucionar la dificultad. En el sub-capítulo 7.5, Mejoras en el proyecto, se muestra el análisis de estos resultados y las mejoras que aportaron a la usabilidad de nuestra aplicación.

Para evaluar la opinión de nuestros usuarios sobre la Interacción con nuestra aplicación desarrollamos el cuestionario que se encuentra en el Apéndice D.2, el cual es una encuesta compuesta de 7 preguntas. Los seis usuarios las respondieron y pudimos obtener algunas sugerencias para mejorar.

Estos fueron los resultados obtenidos:

Preguntas

1. ¿El lenguaje utilizado en los menús y submenús es claro y fácil de entender?
2. ¿Es fácil encontrar el procedimiento para realizar una acción específica?
3. ¿Presenta las funciones de una manera estéticamente agradable?
4. ¿Los colores utilizados en la aplicación son de su agrado?
5. ¿Los íconos y figuras son fácilmente identificables y agradables?
6. ¿Las funciones en la pantalla principal son fáciles de predecir?
7. ¿El uso de la cartelera es amigable?

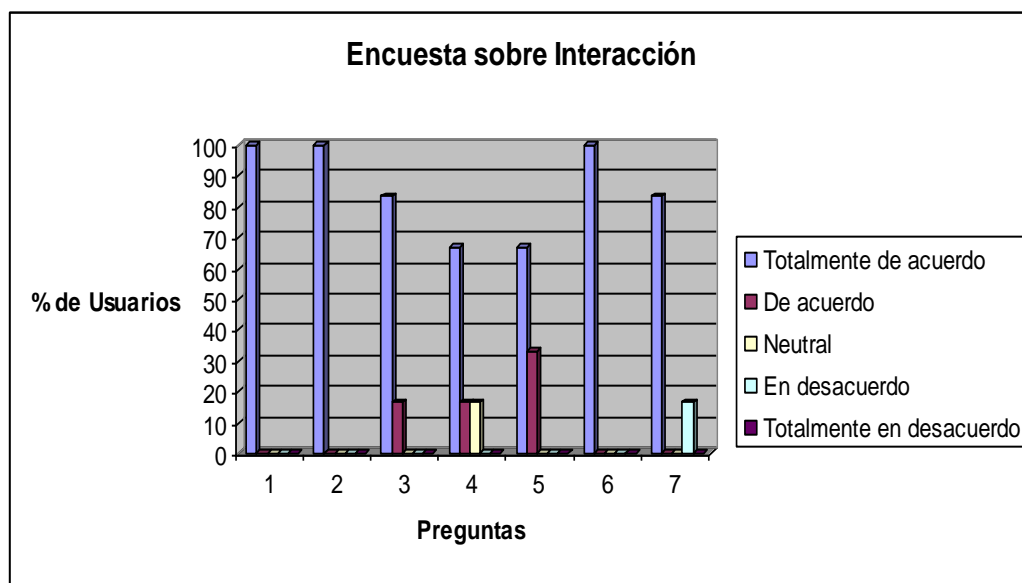


Fig. 7.2

Análisis de encuesta


Como lo muestra el gráfico de la Figura 7.2, todos opinaron que el lenguaje utilizado en los menús y submenús es claro y fácil de entender, es fácil encontrar el procedimiento para realizar una acción específica y que las funciones en la pantalla principal son fáciles de predecir. Casi todos piensan que se presenta las funciones de una manera estéticamente agradable y que los íconos y figuras son fácilmente identificables y agradables. Mientras que el resto de preguntas, sobre si los colores utilizados en la aplicación son de su agrado y el uso amigable de la cartelera no obtuvieron un consenso pero sirvieron para identificar

posibles fallas las cuales se discuten en el siguiente sub-capítulo 7.2 de Mejoras en el Proyecto.

7.5 Mejoras en el proyecto

En la Tabla X a continuación, presentamos las tareas con sus subtareas que representaron alguna dificultad para los usuarios, con su calificación, su descripción de por qué existe un problema y su solución.

Tarea	Subtarea	Calificación	Descripción	Solución
1. Ingresar al chat.	1.2 Ingresar los datos requeridos de usuario, contraseña, facultad y nick (seudónimo).	96,7%	Un miembro de la Institución que posea más de una cuenta de la ESPOL no sabe fácilmente cuál debe ingresar en el campo usuario.	Dentro del campo de texto se colocó "@espol.edu.ec" para que cuando el usuario corra la aplicación identifique fácilmente la cuenta que debe ingresar.
2. Cambiar estado.	2.4 Seleccionar la opción de estado personalizado.	90%	Algunos usuarios no estaban seguros de lo que "Estado personalizado" significaba.	Al colocarse sobre la opción del menú de Estado Personalizado un breve texto descriptivo aparece, "Puede escribir el estado que usted desee, por ejemplo,

				<i>Durmiendo”.</i>
4. Configuración de Proxy	4.3 Ingresar los datos necesarios para la conexión como son Servidor Proxy, Puerto, Usuario y Contraseña.	86,7%	Un usuario no identificó la opción de ayuda que le explicaba lo que era un rendezvous. Un usuario indicó que hay que validar que en el campo de Puerto se validen el ingreso de únicamente datos numéricos.	El ícono de “?” que indicaba ayuda fue reemplazado por otro más significativo. Un ícono azul que representa Información.  También se validó el ingreso de únicamente datos numéricos en el campo Puerto.
5. Agregar Contacto	5.3 Ingresar el usuario al espacio de texto.	96,7%	Se observó la misma situación que en la autenticación. Se debe especificar qué clase de usuario se debe ingresar.	Dentro del campo de texto se colocó “@espol.edu.ec” para que cuando el usuario abra la ventana de agregar contacto identifique fácilmente la cuenta que debe ingresar.
7. Buscar un usuario.	7.3 Ingresar el usuario a buscar.	96,7%	Se observó la misma situación que en la de agregar contacto. Se debe especificar qué clase de usuario se debe ingresar.	Dentro del campo de texto se colocó “@espol.edu.ec” para que cuando el usuario abra la ventana de búsqueda identifique fácilmente la cuenta que debe ingresar.
14. Editar grupo.	14.1 Oprimir clic derecho sobre el	86,7%	Para poder editar un grupo se debe	Esta característica no puede ser modificada ya

	grupo a editar.		primero oprimir clic izquierdo sobre él para seleccionarlo, luego oprimir clic derecho.	que de esta manera trabaja la interfaz gráfica de Java Swing.
15. Eliminar grupo.	15.5 Oprimir clic derecho sobre el grupo a eliminar.	86,7%	Para poder eliminar un grupo se debe primero oprimir clic izquierdo sobre él para seleccionarlo, luego oprimir clic derecho.	Esta característica no puede ser modificada ya que de esta manera trabaja la interfaz gráfica de Java Swing.
17. Enviar archivo.	17.6 Hacer clic derecho sobre el contacto que se desea reciba el archivo.	96,7%	Para poder enviar un archivo a un contacto se debe primero oprimir clic izquierdo sobre él para seleccionarlo, luego oprimir clic derecho.	Esta característica no puede ser modificada ya que de esta manera trabaja la interfaz gráfica de Java Swing.

Tabla X

Mejoras en el Proyecto 1

Gracias a la encuesta realizada a los usuarios, la que se encuentra en el Apéndice D.2, pudimos mejorar la interacción de nuestra aplicación. En la Tabla XI a continuación, se muestran las preguntas que no obtuvieron el resultado ideal esperado por nosotros y los cambios que se realizaron.

Pregunta	Descripción	Solución
¿Los colores utilizados en la aplicación son de su agrado?	Algunos usuarios pensaron que las ventanas de procesos como agregar contactos, buscar usuario, etc., deben ser del mismo color azul del fondo de la ventana principal. Inicialmente esas ventanas de procesos eran grises.	La interfaz de la aplicación cambió de WindowsLookAndFeel a JavaLookAndFeel. De esta manera cambiamos las pantallas gris y botones comunes a nuevas pantallas azules con botones más permisibles.
¿El uso de la cartelera es amigable?	Un usuario opinó que sería buena idea notificar al usuario cuando hay nuevos anuncios en la cartelera que él no haya leído.	Se colocará la palabra "Nuevo!" junto a los anuncios nuevos publicados para alertar al usuario que posee mensajes que no han sido leídos aún.

Tabla XI

Mejoras en el Proyecto 2

Una buena sugerencia para mejorar la interfaz de nuestra aplicación fue hacer la interfaz personalizable y habilitar la posibilidad de intercambiar interfaces entre los usuarios. Es una opción interesante la cual puede satisfacer los gustos de decoración de cualquier usuario pero por falta de tiempo no será implementada, sin embargo se lo podría hacer en un futuro.

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

1. Tomamos una buena decisión al escoger el proyecto JXTA como la base de nuestra red por los siguientes motivos. La tecnología JXTA provee protocolos y servicios para el desarrollo de una aplicación P2P, lo cual nos permite enfocarnos en los aspectos de diseño propios de la aplicación sin perder tiempo en detalles de mecanismos de comunicación P2P, descubrimiento de peers, etc. También al preferir JXTA colaboramos con la evolución de la tecnología P2P en una solución de plataforma madura.
2. La aplicación de mensajería Universitaria propuesta demuestra las ventajas de utilizar JXTA para la implementación de servicios distribuidos. Estos servicios tienen las características de ser escalables, como se demuestra en el Capítulo 6. Así mismo, tienen la característica de ser seguros, como lo describe el Sub-capítulo 5.7 de Servicio de membresías y credenciales. También, son de fácil administración ya que JXTA permite definir específicamente los súper nodos (rendezvous) de forma anticipada y no tener que esperar la auto-organización incierta descrita en el Sub-capítulo 2.4 Características de las redes Par-a-Par.

3. Tras haber hecho una investigación sobre demás trabajos en el área, podemos concluir que somos los primeros en sugerir las siguientes soluciones novedosas para el diseño de aplicaciones P2P empresariales y universitarias: combinación de multicast ingenuo con propagación por chismes para chat uno-a-muchos, contactar a un servidor SMTP para que proporcione un reloj global al sistema, y exportar/importar archivos de configuración vía SMTP/POP. Esas soluciones pueden ser adoptadas en otros sistemas P2P del mismo contexto, permitiéndoles mantener su naturaleza P2P y a la vez sacar ventaja de los servidores que ya se encuentren instalados en la institución.

4. Aunque una red P2P aporte grandes beneficios como tolerancia a fallos, escalabilidad, bajo costo de implementación, mejor rendimiento y alta disponibilidad de recursos, como se lo indica en el sub-capítulo 2.4 Característica de las redes Par-a-par, nunca reemplazará a la arquitectura Cliente/Servidor. Pudimos observar como Cliente/Servidor posee ventajas sobre la arquitectura P2P al tener un control total sobre los peers y los recursos de la red ofreciendo alta confiabilidad de contenidos y bajos tiempos de búsqueda. Es por esto que JXTA incorpora esta modalidad en su concepto de red a través de los rendezvous.

RECOMENDACIONES

1. Sería recomendable que nuestra aplicación chat sea un módulo que forme parte de los sistemas académicos de la ESPOL para mejorar la comunicación y participación entre estudiantes y profesores. Su implementación en los laboratorios y oficinas de la institución facilitará las actividades colaborativas entre los miembros de la comunidad, la cual se espera incida en un incremento de la calidad educativa e investigativa de la misma.
2. Para la implantación de nuestra aplicación en la ESPOL es necesario realizar cuatro tareas específicas. La primera es mediante un análisis de las redes seleccionar las computadoras que cumplan con los requisitos para funcionar como rendezvous y posteriormente obtener una autorización de la ESPOL que permita correr nuestra aplicación en modo rendezvous en las computadoras seleccionadas, este término se lo vio en el sub-capítulo 2.3, Funcionamiento de redes Par-a-Par. La segunda es analizar y seleccionar uno o más servidores de la ESPOL con IP pública y salida a Internet que puedan operar como los “peer Relay”, este término es definido en el sub-capítulo 5.2 Servidor Proxy. La tercera es desarrollar una interfaz gráfica que permita definir cuáles son las computadoras que

están operando como "peer Relay". La cuarta tarea es opcional pero sugerida, y consiste en el desarrollo de una interfaz gráfica que me permita definir cuáles son las computadoras que están operando como rendezvous en la ESPOL y de esta manera mejorar notablemente el tiempo de conexión a un "peer rendezvous".

3. Como trabajo futuro se podría mejorar la funcionalidad e interfaz de nuestro programa implementando recursos multimedia como cámaras, micrófonos, etc., y permitir personalizar la interfaz gráfica.

APÉNDICES

APÉNDICE A

GLOSARIO

Advertise: Realizar un anuncio público, en el lenguaje de JXTA significa que los peers deben publicar sus servicios.

Advertisement: Estructura de meta data del proyecto JXTA que describe los recursos de un peer como son los peers, grupos de peer, pipes y servicios. Los advertisements son representados como documentos XML.

AIMC: Asociación para la Investigación de los Medios de Comunicación. Entidad sin ánimo de lucro cuyo fin social es aumentar el conocimiento sobre el uso de los diferentes medios de comunicación. AIMC ha llevado a cabo diversos estudios en torno a Internet cuyos resultados se pueden consultar en su página Web, www.aimc.es.

AOL: AIM (America-On-Line Instant Messenger) es un programa de mensajería instantánea de America On Line denominada habitualmente Instant Messenger. La popularidad de la herramienta varía, y suele ubicarse en el tercer lugar de uso, después del Messenger de Microsoft y del de Yahoo!.

API: Application Programming Interface, Interfaz de programación de aplicaciones, una serie de funciones que están disponibles para realizar programas para un cierto entorno.

Audiogalaxy: Sistema para compartir archivos que indexaba archivos MP3. Originalmente creado por Michael Merhej como un sitio FTP de índices llamado “The Borg Search”, evolucionó en un sistema peer to peer robusto.

Benchmarking: Proviene del inglés bench mark que significa marca o punto de referencia, es decir, se toma un punto de comparación para medir lo hecho por nosotros y por los demás.

Bindings: Múltiples implementaciones de los protocolos de JXTA.

BitTorrent: Plataforma de entrega asistida por peers que distribuye, descubre y consume grandes archivos de alta calidad en la Web.

Bluetooth: Sistema de comunicación inalámbrica el cual permite la interconexión de diferentes dispositivos electrónicos (PCs, teléfonos fijos o móviles, agendas electrónicas, auriculares, etc.) de forma que presenta un estándar creado por importantes empresas de los sectores de la informática y las telecomunicaciones.

CSI: Centro de Servicios Informáticos de la Escuela Superior Politécnica del Litoral, (ESPOL). Responsable de proveer la infraestructura de comunicación de servicios. (Más información en <http://www.csi.espol.edu.ec>)

DLL: Es el acrónimo de Dynamic Linking Library (Bibliotecas de Enlace Dinámico), término con el que se refiere a los archivos con código ejecutable que se cargan bajo demanda del programa por parte del sistema operativo.

Discovery: Descubrir, determinar la existencia de algo, en el lenguaje de JXTA significa descubrir a los peers en la red.

DMCA: Digital Millenium Copyright Act, Acta de copyright del milenio digital, firmada por ley por el presidente Clinton el 28 de Octubre de 1998. Está dividida en cinco títulos, siendo el segundo el Acta de Limitación de Responsabilidad de Infracción de Copyright en Línea. Esta crea las limitaciones de responsabilidad de los proveedores de servicio de Internet de infracciones de copyright en ciertos tipos de actividad.

DNS: Conjunto de protocolos y servicios que permite a los usuarios utilizar nombres en vez de tener que recordar direcciones IP numéricas.

eDonkey: Fue una aplicación de intercambio de archivos P2P. Se conectaba tanto a la red con el mismo nombre como a Overnet.

Endpoint: Son usados por los peers para establecer conexiones directas punto a punto entre dos peers.

Firewalls: Se coloca entre la red local e Internet. Se asegura que todas las comunicaciones entre la red e Internet se realicen conforme a las políticas de seguridad de la organización o corporación que lo utiliza.

Framework: Una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado.

Freenet: Red de comunicaciones entre peers descentralizada diseñada para resistir la censura, la cual utiliza el ancho de banda y espacio de almacenamiento de las computadoras de sus miembros para permitir publicar u obtener información de todo tipo en un completo anonimato.

Gnutella: Proyecto de software distribuido para crear un protocolo de red de distribución de archivos entre peers, sin un servidor central.

Grokster: Un programa peer to peer para compartir archivos para computadoras corriendo el sistema operativo de Microsoft Windows. El producto era similar en apariencia y funcionalidad a Kazaa.

HomePNA (Home Phoneline Networking Alliance): Alianza de varias empresas que trabajan en el desarrollo de una tecnología que permita implementar redes de área local usando la instalación telefónica de una vivienda.

HTTP: (Hypertext Transmisión Protocol) Protocolo para transferir archivos o documentos hipertexto a través de la red. Se basa en una arquitectura cliente/servidor.

ICQ: Es un servicio de mensajería instantánea y el primero de su tipo en ser ampliamente utilizado en Internet, mediante el cual es posible chatear y enviar mensajes instantáneos a otros usuarios conectados a la red de ICQ. También permite el envío de archivos, videoconferencias y charlas de voz.

IDE de NetBeans: “Integrated Development Environment” o Entorno Integrado de Desarrollo es un IDE de Java de código abierto robusto y gratis que provee al desarrollador con todo lo necesario para crear aplicaciones de escritorio, Web o móviles.

Indexar: El proceso de convertir una colección de datos en una base de datos, conveniente para su recuperación y búsqueda fácil.

Interoperabilidad: Condición mediante la cual sistemas heterogéneos pueden intercambiar procesos o datos.

IRC: Internet Relay Chat, es un protocolo de comunicación en tiempo real basado en texto, que permite debates en grupo o entre dos personas y que está clasificado dentro de la Mensajería instantánea. Las conversaciones se desarrollan en los llamados canales de IRC, designados por nombres que habitualmente comienzan con el carácter # o & (este último sólo es utilizado en canales locales del servidor). Es un sistema de charlas ampliamente utilizado por personas de todo el mundo.

ISP: Internet Service Provider, proveedores de servicios de Internet.

J2SE: Iniciales para “Java 2 Platform, Standard Edition”. Es una colección de APIs del lenguaje de programación Java usado para muchos programas de la plataforma Java. Incluye todas las clases en el Java SE además de otras que son más útiles para programas corriendo en servidores que en Workstations.

Jabber: Protocolo libre gestionado por Jabber Software Foundation basado en el estándar XML para mensajería instantánea. La red de Jabber está formada por miles de grandes y pequeños servidores en todo el mundo, interconectados por Internet.

Java Virtual Machine: Una implementación de software de una unidad de proceso central (CPU) que corre código Java compilado.

JIM: Aplicación de código abierto construida sobre una de las últimas tecnologías P2P. Trabaja sin servidor y provee a los grupos y la Internet con una herramienta para comunicarse de manera innovadora y segura. (Más información en <http://jxtaim.sourceforge.net/download.html>)

JXTA: grupo de protocolos que permite a cualquier dispositivo conectado a la red comunicarse y colaborar de una manera peer-to-peer. Los usuarios de JXTA, también llamados peers, crean una red virtual superpuesta donde cada peer puede interactuar con otros peers y recursos directamente incluso cuando algunos de ellos se encuentran detrás de firewalls y NATs o se encuentran en diferentes transportes de red.

Kazaa: Aplicación para distribución de archivos entre peers que utiliza el protocolo Fast Track. Comúnmente utilizado para intercambiar música y películas.

Lphant: Solución peer to peer para buscar, descargar y compartir cualquier archivo de cualquier tipo o tamaño con usuarios de la red eDonkey.

ModuleSpecID: Describe el ID de una especificación de una clase de módulo dado, es una forma de proveer la funcionalidad que un módulo de clase implica. Puede haber múltiples especificaciones de módulos para una clase de módulo dada. La especificación de módulo es primariamente usada para acceder a un módulo.

Morpheus: Una interfaz de búsqueda basada en Web para compartir archivos de forma peer to peer. Ofrecía búsquedas para toda clase de archivos no solo MP3.

MP3: MPEG-1 Audio Layer-3, es una tecnología y formato estandar para una secuencia de sonido en un archivo muy pequeño.

MSN Messenger: MSN Messenger es un conocido programa de mensajería instantánea para sistemas Windows y Mac OS.

Multicast: Un servicio de red en el cual un único flujo de datos, proveniente de una determinada fuente, se puede enviar simultáneamente a diversos receptores interesados.

MyJXTA: Es la re implementación de la aplicación InstantP2P para solucionar las limitaciones y características de no escalabilidad de la versión previa. Provee la habilidad de navegar a través de los grupos de peers de la red JXTA, crear y unirse a grupos, unirse a un grupo de chat, crear una sesión uno a uno de chat con otros miembros, buscar y compartir documentos. (Más información en <http://myjxta/jxta.org/>)

Napster: Servicio de distribución de archivos de música. Causo un impacto mayor en Internet durante el año 2000. Su tecnología permitía a los fanáticos de la música compartir sus colecciones de MP3 fácilmente con otros usuarios, causando así violaciones masivas de copyright.

NATS: Network Address Translation, permite que un sólo equipo, como un router, actúe como un agente entre la Internet o la red pública y una red privada o local.

Net Peer Group: El primer grupo que por defecto es instanciado por JXTA. Todos los peers pertenecen al net Peer Group.

Peer: Cualquier equipo de red que implemente uno o más de los protocolos de JXTA.

Peer ID: Un ID que identifica únicamente a un peer.

Pipe: Un mecanismo de transferencia de mensaje asíncrono y unidireccional usado por los peers para enviar y recibir mensajes. Los pipes están ligados a específicos peer endpoints como puerto TCP y dirección de IP asociada.

PKI (Primary Key Infrastructure): Infraestructura de clave pública. Es una combinación de políticas y procedimientos de hardware y software que permiten asegurar la identidad de los participantes en un intercambio de datos usando criptografía pública. Soporta firmas digitales y otros servicios de seguridad habilitados de clave pública.

Proxies: Una computadora que ofrece un servicio de red de computadores para permitir a los clientes conexiones de red indirectas a otros servicios de red.

Publish: Preparar algo para su distribución, en el caso de JXTA por ejemplo publicar grupos.

Rendezvous: Mantiene una caché de advertisements y devuelve pedidos de discovery a otros peers rendezvous para ayudar a los peers a descubrir recursos.

RIAA: Recording Industry Association of America, Asociación de la Industria Discográfica de Estados Unidos, es una asociación estadounidense que representa a la industria discográfica. Su misión es cuidar un negocio y el clima legal que soporta y promueve la viabilidad de sus miembros creativa y financieramente.

Script: Un programa o una secuencia de instrucciones que es interpretado y llevado a cabo por otro programa en lugar de ser procesado por el procesador de la computadora.

SETI@Home: Protector de pantalla gratuito que instalado en una PC analiza señales procedentes del espacio en busca de señales de inteligencia extraterrestre.

SMTP: Simple Mail Transfer Protocol, protocolo que utiliza Internet para el correo electrónico.

SSL: Secure Socket Layer, nivel de socket seguro. Es un protocolo de comunicación para transmitir información privada a través de Internet. Funciona mediante datos cifrados que se transmiten a través de la conexión SSL.

Sun Microsystems: Desarrollador de la tecnología Java que desde 1995 ha revolucionado el mundo de las aplicaciones en Internet. La visión de Sun es

“la red es el ordenador”. Para Sun “el valor real de todo sistema está en su capacidad de conexión a otros sistemas”. Y con ese fin todos los sistemas de Sun incluyen capacidades de red.

TCP/IP: Los dos protocolos más importantes en la que se basa Internet y que permiten la transmisión de datos entre redes de computadoras. Protocolo de Control de transmisión (TCP) y Protocolo de Internet (IP).

TCP relay: Es un simple tipo de retransmisión. Un relay escucha en un puerto en una cierta dirección IP en el firewall y replica todo el tráfico al servidor especificado. Un TCP relay solo procesa tráfico TCP. Ejemplos de servicios que pueden ser procesados por un TCP relay incluyen Telnet (terminal connections), SMTP (email), POP (email), NNTP (news), and HTTP (www). Desde el cliente el relay trabaja como un servidor, y desde el servidor el relay trabaja como un programa cliente.

TTL: El TTL es el tiempo de vida de los registros DNS cacheados e influye en cuánto tiempo se contacta a una IP.

Ubicuidad: Que se encuentra presente en todos los ambientes.

UNIX: Es un sistema operativo portable, multitarea y multiusuario desarrollado en principio por un grupo de empleados de los laboratorios Bell de AT&T.

URI: Uniform Resource Identifier, Identificador de Recurso Uniforme, una cadena de caracteres para identificar un recurso abstracto o físico.

URN: Uniform Resource Name, Nombre de Recurso Uniforme, un tipo de URI que provee identificadores persistentes para recursos de información.

Usenet: Usenet permite que dos computadoras intercambien información. No posee una unidad administrativa central, la distribución del contenido es administrado por cada nodo y el contenido de la red Usenet es replicado a través de sus nodos.

XML: eXtensible Markup Language, lenguaje de marcado ampliable o extensible desarrollado por el World Wide Web Consortium (W3C). Puede usarse para almacenar datos en un formato estructurado, basado en texto y definido por el usuario.

Yahoo! Messenger: Un programa popular para mensajes instantáneos, es gratis y puede ser usado con un ID de Yahoo.

APÉNDICE B

FLUJO DE VENTANAS

DISEÑO LÓGICO

B.1 Lista de Escenarios

Caso de uso 1: Conectarse a la red

- 1.1 Acceso exitoso a la red.
- 1.2 Acceso fallido por no conexión a Internet.
- 1.3 Acceso fallido por usuario inválido.
- 1.4 Acceso no otorgado por fallas técnicas.

Caso de uso 2: Agregar contacto

- 2.1 Agregar contacto exitosamente.
- 2.2 Falla en agregar por dirección electrónica no válida.
- 2.3 Falla en agregar por problemas técnicos.

Caso de uso 3: Eliminar contacto

- 3.1 Eliminar contacto exitosamente.
- 3.2 Falla en eliminar por problemas técnicos.

Caso de uso 4: Agrupar contactos por facultad o unidad administrativa

- 4.1 Agrupar contactos exitosamente.
- 4.2 Falla en agrupar por problemas técnicos.

Caso de uso 5: Crear grupo de amigos

- 5.1 Creación de grupo exitoso.

5.2 Creación fallida porque el grupo ya existe.

5.3 Creación fallida por problemas técnicos.

Caso de uso 6: Eliminar grupo

6.1 Eliminación de grupo exitoso.

6.2 Eliminación fallida porque contiene contactos en él.

6.3 Eliminación fallida por problemas técnicos.

Caso de uso 7: Iniciar conversación con un usuario

7.1 Conversación iniciada exitosamente.

7.2 Conversación fallida porque no se pudo establecer un “pipe” de conversación.

7.3 Conversación fallida por problemas técnicos.

Caso de uso 8: Agregar otros usuarios a la conversación

8.1 Usuario agregado con éxito.

8.2 Usuario no agregado porque no se pudo establecer un “pipe” de conversación.

8.3 Usuario no agregado por problemas técnicos.

Caso de uso 9: Enviar mensajes

9.1 Mensaje enviado exitosamente.

9.2 Mensaje fallido por problemas en el receptor.

9.3 Mensaje fallido por problemas técnicos.

Caso de uso 10: Recibir mensajes

10.1 Mensaje recibido exitosamente.

10.2 Mensaje fallido por problemas en el emisor.

10.3 Mensaje fallido por problemas técnicos.

Caso de uso 11: Enviar archivo

11.1 Archivo enviado exitosamente.

11.2 Archivo no enviado por pérdida de datos.

11.3 Archivo no enviado por problemas técnicos.

Caso de uso 12: Recibir archivo

12.1 Archivo recibido exitosamente.

12.2 Archivo no recibido por pérdida de datos.

12.3 Archivo no recibido por problemas técnicos.

Caso de uso 13: Guardar conversación

13.1 Conversación guardada exitosamente.

13.2 Conversación no guardada por problemas técnicos.

Caso de uso 14: Cambiar estado

14.1 Cambio de estado exitoso.

14.2 Cambio no exitoso por problemas técnicos.

Caso de uso 15: Leer anuncios de la cartelera comunal

15.1 Anuncio encontrado exitosamente.

15.2 Anuncio no encontrado por el nodo.

15.3 Anuncio no encontrado por problemas técnicos.

Caso de uso 16: Pegar anuncios a la cartelera comunal

16.1 Anuncio publicado exitosamente.

16.2 Anuncio no publicado por no disponibilidad de nodos en la red.

16.3 Anuncio no publicado por problemas técnicos.

Caso de uso 17: Buscar usuarios conectados

17.1 Usuarios encontrados exitosamente.

17.2 Usuario no encontrado por time-out.

17.3 Usuario no encontrado por problemas técnicos.

Caso de uso 18: Guardar lista de contactos

18.1 Lista guardada exitosamente.

18.2 Lista no guardada por problemas técnicos.

Caso de uso 19: Leer lista de contactos

19.1 Lista leída exitosamente.

19.2 Lista no leída por ruta de archivo incorrecta.

19.3 Lista no leída por archivo de lista de contacto inválido.

19.4 Lista no leída por archivo de lista de contacto corrupto.

19.5 Lista no leída por problemas técnicos.

Caso de uso 20: Cambiar opciones de configuración

20.1 Opciones cambiadas exitosamente.

20.2 Opciones no cambiadas por problemas técnicos.

Caso de uso 21: Cerrar sesión

21.1 Se cierra sesión exitosamente.

21.2 Sesión no cerrada por problemas técnicos.

B.2 Especificación de escenarios

Caso de uso 1: Conectarse a la red

Escenario 1.1 Acceso exitoso a la red.

Asunciones:

- El chat corre sin problemas.
- Se ha obtenido conexión con los peers disponibles en la red.

Resultados:

- El usuario accede al chat y obtiene la ventana principal con las opciones disponibles de la aplicación.

Escenario 1.2 Acceso fallido por no conexión a Internet.

Asunciones:

- No se tiene conexión a Internet.
- La red no está disponible para atender su petición.

Resultados:

- El usuario no se conecta a la red.

Escenario 1.3 Acceso fallido por usuario inválido.

Asunciones:

- El usuario no existe.
- La contraseña es incorrecta

Resultados:

- El usuario no se conecta a la red.

Escenario 1.4 Acceso no otorgado por fallas técnicas.**Asunciones:**

- El servidor de autenticación se encuentra fuera de servicio.
- El sistema falla por errores en la computadora que el cliente está utilizando.
- La aplicación está configurada incorrectamente.

Resultados:

- El usuario no accede al sistema y recibe un mensaje de fallas técnicas y le pide que lo intente luego.

Caso de uso 2: Agregar contacto**Escenario 2.1** Agregar contacto exitosamente.**Asunciones:**

- Dirección electrónica válida
- La red funciona sin problemas.

Resultados:

- El usuario agregado aparece en la lista de contactos dentro del grupo de contactos no determinados.

Escenario 2.2 Falla en agregar por dirección electrónica no válida.**Asunciones:**

- La dirección electrónica no se encuentra en la base de datos del servidor de autenticación.

Resultados:

- El usuario recibe un mensaje de usuario no existente, por favor ingrese una dirección válida.

Escenario 2.3 Falla en agregar por problemas técnicos.

Asunciones:

- El servidor de autenticación se encuentra fuera de servicio.
- El sistema falla por errores en la computadora que el cliente está utilizando.

Resultados:

- El usuario no agrega el contacto y recibe un mensaje que le dice que lo intente luego.

Caso de uso 3: Eliminar contacto

Escenario 3.1 Eliminar contacto exitosamente.

Asunciones:

- No se está produciendo una conversación con el contacto.
- La red funciona sin problemas.

Resultados:

- El contacto desaparece de la lista de contactos y se muestra un mensaje al usuario que dice, usuario ha sido eliminado de su lista.

Escenario 3.2 Falla en eliminar por problemas técnicos.

Asunciones:

- El sistema falla por errores en la computadora que el cliente está utilizando.

Resultados:

- El usuario no elimina el contacto y recibe un mensaje que le dice que lo intente luego.

Caso de uso 4: Mover contacto**Escenario 4.1** Mover contacto exitosamente.**Asunciones:**

- La red funciona sin problemas.

Resultados:

- El contacto se mueve al grupo elegido.

Escenario 4.2 Falla en mover contacto por problemas técnicos.**Asunciones:**

- El sistema falla por errores en la computadora que el cliente está utilizando.

Resultados:

- El usuario no mueve el contacto al grupo elegido, el contacto permanece en el grupo original.

Caso de uso 5: Crear grupo**Escenario 5.1** Creación de grupo exitoso.**Asunciones:**

- El usuario ingresa un nombre de grupo válido
- La red funciona sin problemas.

Resultados:

- Se crea un grupo nuevo en la ventana principal.

Escenario 5.2 Creación fallida porque el grupo ya existe.

Asunciones:

- Existe un grupo con el mismo nombre.

Resultados:

- No se crea al grupo y el usuario recibe un mensaje de, por favor ingrese otro nombre ya que el grupo especificado ya existe.

Escenario 5.3 Creación fallida por problemas técnicos.

Asunciones:

- El sistema falla por errores en la computadora que el cliente está utilizando.

Resultados:

- El usuario no crea el grupo y recibe un mensaje que le dice que lo intente luego.

Caso de uso 6: Eliminar grupo

Escenario 6.1 Eliminación de grupo exitoso.

Asunciones:

- No hay contactos dentro del grupo a eliminar
- La red funciona sin problemas.

Resultados:

- Se elimina el grupo de la lista de contactos del usuario.

Escenario 6.2 Eliminación fallida porque contiene contactos en él.

Asunciones:

- Hay contactos dentro del grupo que se desea eliminar.

Resultados:

- No se elimina el grupo y el usuario recibe un mensaje, de por favor remueva los contactos del grupo antes de proceder a eliminarlo.

Escenario 6.3 Eliminación fallida por problemas técnicos.**Asunciones:**

- El sistema falla por errores en la computadora que el cliente está utilizando.

Resultados:

- El usuario no elimina el grupo y recibe un mensaje que le dice que lo intente luego.

Caso de uso 7: Iniciar conversación con un usuario**Escenario 7.1** Conversación iniciada exitosamente.**Asunciones:**

- Se estableció un pipe de conversación.
- Hay conexión, la red funciona sin problemas.

Resultados:

- Se abre una ventana individual para mantener contacto con el usuario requerido.

Escenario 7.2 Conversación fallida porque no se pudo establecer un “pipe” de conversación.

Asunciones:

- No hay conexión.
- El usuario se ha desconectado.
- Problemas en la red.

Resultados:

- No se puede abrir una ventana individual con el usuario con el cual se quiere mantener una conversación sincrónica.

Escenario 7.3 Conversación fallida por problemas técnicos.**Asunciones:**

- La conversación falla por errores en la computadora que el cliente está utilizando.

Resultados:

- El usuario no abre la conversación individual y recibe un mensaje que le dice que lo intente luego.

Caso de uso 8: Agregar otros usuarios a la conversación**Escenario 8.1** Usuario agregado con éxito.**Asunciones:**

- Se estableció un pipe de conversación.
- Hay conexión, la red funciona sin problemas.

Resultados:

- Se agrega el nuevo contacto a la ventana de conversación.

Escenario 8.2 Usuario no agregado porque no se pudo establecer un “pipe” de conversación.

Asunciones:

- No hay conexión.
- El usuario se ha desconectado.
- Problemas en la red.

Resultados:

- No se puede incluir el usuario a la ventana de conversación, recibe un mensaje de que no es posible.

Escenario 8.3 Usuario no agregado por problemas técnicos.

Asunciones:

- La conversación falla por errores en la computadora que el cliente está utilizando.

Resultados:

- El usuario no agrega el contacto a la ventana de conversación y recibe un mensaje que le dice que lo intente luego.

Caso de uso 9: Enviar mensajes

Escenario 9.1 Mensaje enviado exitosamente.

Asunciones:

- Existe conexión a la red.
- El usuario emisor tiene una ventana de conversación abierta con el receptor.

Resultados:

- Se envía el mensaje al usuario receptor.

Escenario 9.2 Mensaje fallido por problemas en el receptor.

Asunciones:

- El usuario se ha desconectado.
- Problemas en la red.

Resultados:

- No se puede enviar el mensaje del emisor.

Escenario 9.3 Mensaje fallido por problemas técnicos.

Asunciones:

- El mensaje no es enviado por errores en la computadora que el cliente está utilizando.

Resultados:

- El usuario no envía el mensaje al receptor y recibe un mensaje que le dice que lo intente luego.

Caso de uso 10: Recibir mensajes

Escenario 10.1 Mensaje recibido exitosamente.

Asunciones:

- Existe conexión a la red.
- El usuario receptor tiene una ventana de conversación abierta con el emisor.

Resultados:

- Se recibe el mensaje del usuario emisor.

Escenario 10.2 Mensaje fallido por problemas en el receptor.

Asunciones:

- El usuario se ha desconectado.
- Problemas en la red.

Resultados:

- No se puede enviar el mensaje, el receptor no lo recibe.

Escenario 10.3 Mensaje fallido por problemas técnicos.

Asunciones:

- El mensaje no es enviado por errores en la computadora que el cliente está utilizando.

Resultados:

- El usuario no recibe el mensaje del receptor.

Caso de uso 11: Enviar archivo

Escenario 11.1 Archivo enviado exitosamente.

Asunciones:

- Existe conexión a la red.
- El usuario receptor tiene una ventana de conversación abierta con el emisor.

Resultados:

- Se recibe el archivo del usuario emisor.

Escenario 11.2 Archivo no enviado por pérdida de datos.

Asunciones:

- El archivo se corrompe
- Problemas en la red.

Resultados:

- No se puede enviar el archivo, el receptor no lo recibe.

Escenario 11.3 Archivo no enviado por problemas técnicos.

Asunciones:

- El archivo no es enviado por errores en la computadora que el cliente está utilizando.

Resultados:

- El usuario no recibe el archivo del emisor.

Caso de uso 12: Recibir archivo

Escenario 12.1 Archivo recibido exitosamente.

Asunciones:

- Existe conexión a la red.
- El usuario receptor tiene una ventana de conversación abierta con el emisor.

Resultados:

- Se recibe el archivo del usuario emisor.

Escenario 12.2 Archivo no recibido por pérdida de datos.

Asunciones:

- El archivo se corrompe

- Problemas en la red.

Resultados:

- No se puede recibir el archivo.

Escenario 12.3 Archivo no recibido por problemas técnicos.

Asunciones:

- El archivo no es recibido por errores en la computadora que el cliente está utilizando.

Resultados:

- El usuario no recibe el archivo del emisor.

Caso de uso 13: Guardar conversación

Escenario 13.1 Conversación guardada exitosamente.

Asunciones:

- Existe conexión a la red.

Resultados:

- Se guarda la conversación en la ruta elegida por el usuario.

Escenario 13.2 Conversación no guardada por problemas técnicos.

Asunciones:

- Problemas en la red.

Resultados:

- No se puede guardar la conversación en la ruta indicada, se le pide que lo intente luego.

Caso de uso 14: Cambiar estado

Escenario 14.1 Cambio de estado exitoso.**Asunciones:**

- Existe conexión a la red.

Resultados:

- Se cambia el estado del usuario inmediatamente.

Escenario 14.2 Cambio no exitoso por problemas técnicos.**Asunciones:**

- Problemas en la red.

Resultados:

- No se puede cambiar el estado del usuario, se le pide que lo intente luego.

Caso de uso 15: Leer anuncios de la cartelera comunal**Escenario 15.1** Anuncio encontrado exitosamente.**Asunciones:**

- Existe conexión a la red.
- Hay peers funcionando en la red.
- Se cumple el tiempo de proceso.

Resultados:

- Se muestra la ventana de cartelera comunal, con todos los mensajes de los usuarios.

Escenario 15.2 Anuncio no encontrado por el nodo.**Asunciones:**

- Problemas en la red.
- Se interrumpe el tiempo de búsqueda.

Resultados:

- No se puede mostrar la cartelera, muestra una cartelera vacía.

Escenario 15.3 Anuncio no encontrado por problemas técnicos.

Asunciones:

- La cartelera no es mostrada por errores en la computadora que el cliente está utilizando.

Resultados:

- La cartelera no se muestra o se muestra vacía.

Caso de uso 16: Pegar anuncios a la cartelera comunal

Escenario 16.1 Anuncio publicado exitosamente.

Asunciones:

- Existe conexión a la red.
- Hay peers funcionando en la red.
- Se cumple el tiempo de proceso.

Resultados:

- Se muestra la ventana de cartelera comunal, con todos los mensajes de los usuarios y el que se incluyó recientemente.

Escenario 16.2 Anuncio no publicado por no disponibilidad de nodos en la red.

Asunciones:

- Problemas en la red, no hay suficientes nodos.
- Se interrumpe el tiempo de publicación.

Resultados:

- No se puede pegar el anuncio en la cartelera.

Escenario 16.3 Anuncio no publicado por problemas técnicos.

Asunciones:

- El anuncio no es mostrado en la cartelera por errores en la computadora que el cliente está utilizando.

Resultados:

- El anuncio no se muestra en la cartelera.

Caso de uso 17: Buscar usuarios conectados

Escenario 17.1 Usuarios encontrados exitosamente.

Asunciones:

- Existe conexión a la red.
- Los usuarios se encuentran conectados a la red.
- Se cumple el tiempo de proceso de búsqueda.

Resultados:

- Se muestra la ventana de búsqueda y en ella a los usuarios encontrados.

Escenario 17.2 Usuario no encontrado por time-out.

Asunciones:

- Problemas en la red.

- Se interrumpe el tiempo de búsqueda.

Resultados:

- En la ventana de búsqueda no se muestra a los usuarios encontrados.

Escenario 17.3 Usuario no encontrado por problemas técnicos.**Asunciones:**

- La ventana de búsqueda no es mostrada por errores en la computadora que el cliente está utilizando.

Resultados:

- La ventana de búsqueda con los usuarios encontrados no es mostrada o es presentada vacía.

Caso de uso 18: Guardar lista de contactos**Escenario 18.1** Lista guardada exitosamente.**Asunciones:**

- Existe conexión a la red.

Resultados:

- Se guarda la lista de contactos en la ruta elegida por el usuario.

Escenario 18.2 Lista no guardada por problemas técnicos.**Asunciones:**

- Problemas en la red.

Resultados:

- No se puede guardar la lista en la ruta indicada, se le pide que lo intente luego.

Caso de uso 19: Leer lista de contactos**Escenario 19.1** Lista leída exitosamente.**Asunciones:**

- Existe conexión a la red.
- La lista de contactos se encuentra en buen estado.

Resultados:

- Se lee la lista de contactos desde la ruta elegida por el usuario.

Escenario 19.2 Lista no leída por ruta de archivo incorrecta.**Asunciones:**

- La ruta especificada no contiene ningún archivo de lista de contactos.

Resultados:

- No se puede leer la lista desde la ruta indicada, se le indica que existe un error.

Escenario 19.3 Lista no leída por archivo de lista de contacto inválido.**Asunciones:**

- El archivo de lista de contactos no es un archivo XML válido.

Resultados:

- No se puede leer la lista desde la ruta indicada, se le indica que existe un error de compatibilidad.

Escenario 19.4 Lista no leída por archivo de lista de contacto corrupto.**Asunciones:**

- El archivo de lista de contactos ha sido dañado.

Resultados:

- No se puede leer la lista desde la ruta indicada, se le indica que existe un error.

Escenario 19.5 Lista no leída por problemas técnicos.

Asunciones:

- Problemas en la red.

Resultados:

- No se puede leer la lista desde la ruta indicada, se le pide que lo intente luego.

Caso de uso 20: Cambiar opciones de configuración

Escenario 20.1 Opciones cambiadas exitosamente.

Asunciones:

- Existe conexión a la red.

Resultados:

- Se actualizan las opciones especificadas por el usuario.

Escenario 20.2 Opciones no cambiadas por problemas técnicos.

Asunciones:

- Problemas en la red.

Resultados:

- No se puede actualizar las opciones especificadas por el usuario, se le pide que lo intente luego.

Caso de uso 21: Cerrar sesión

Escenario 21.1 Se cierra sesión exitosamente.

Asunciones:

- Existe conexión a la red.

Resultados:

- Se cierra la sesión, se cierran todas las ventanas de la aplicación que pudieran estar abiertas y el usuario obtiene la página inicial de conexión de la aplicación.

Escenario 21.2 Sesión no cerrada por problemas técnicos.

Asunciones:

- Problemas en la red.

Resultados:

- No se puede cerrar sesión, el usuario debe cerrar la aplicación manualmente.

CÓDIGO

C.1 Configuración de JXTA para comunicarse via HTTP Proxy Servers

*/*Autenticación para trabajar con servidores proxies a través de JXTA*/*

```
System.setProperty("http.proxyHost", EspolMessenger.getHttpProxy());
```

```
System.setProperty("http.proxyPort", EspolMessenger.getHttpProxyPort());
```

C.2 Servicio Web

/ El código invoca la operación del DirectorioEspolSoap: autenticación en el Servicio Web*/*

```
try { webservice.DirectorioEspol directorioEspol = new
```

```
webservice.DirectorioEspol_Impl();
```

```
    webservice.DirectorioEspolSoap _directorioEspolSoap =
```

```
directorioEspol.getDirectorioEspolSoap();
```

*/*Retorna 0 si la información de usuario y contraseña es correcta, 1 si es inválida*/*

```
    return _directorioEspolSoap.autenticacion(name, pass)?0:1;
```

```
    } catch(javax.xml.rpc.ServiceException ex) {
```

*/*se encarga de la excepción remota*/*

```
System.out.println("Error en el servicio de autenticación");
```

```
return 2;
```

```

    } catch(java.rmi.RemoteException ex) {
        /*se encarga de la excepción remota*/
        System.out.println("No se pudo establecer conexión con el servidor de
autenticación");
        return 3;
    } catch(Exception ex) {
        /*se encarga de la excepción remota*/
        System.out.println("Error3");
        return 4;
    }
}

```

C.3 Servicio Web

```

/*Autenticación para trabajar con servicios Web a través de un servidor
proxy o firewall*/
System.setProperty("https.proxyHost", EspolMessenger.getHttpProxy());
System.setProperty("https.proxyPort", EspolMessenger.getHttpProxyPort());
Authenticator.setDefault( new httpAuthenticateProxy() );
public class httpAuthenticateProxy extends Authenticator {
    protected PasswordAuthentication getPasswordAuthentication() {
        return new
        PasswordAuthentication(EspolMessenger.getHttpProxyUser(),EspolMesseng
er.getHttpProxyPassword().toCharArray());
    }
}

```

```

    }
}

```

C.4 Unirse al Net Peer Group

```

/*Inicializa la plataforma JXTA y crea el netPeerGroup*/
netPeerGroup=PeerGroupFactory.newNetPeerGroup();

/*Obtiene el servicio de RendezVous*/
RendezVousService rdv = netPeerGroup.getRendezVousService();

/*Si es que mi peer no es rendezvous, espero indefinidamente hasta
conectarme a un peer Rendezvouz y se actualice el NetPeerGroup conmigo
como nuevo peer*/
if(!netPeerGroup.isRendezvous())
    while (! rdv.isConnectedToRendezVous()) {
        try {
            Thread.sleep(2000);
        } catch (InterruptedException ex)
        }
}

```

C.5 Creacion del grupo ESPOL

```

/*Este método crea el grupo ESPOL donde ntp es netPeerGroup*/
private static PeerGroup crearGrupoEn(PeerGroup ntp, String nombre) {
    PeerGroup pg;
    PeerGroupAdvertisement adv;

/*Esto luego me permitirá publicar el advertisement*/
}

```

```
DiscoveryService descubridor=ntp.getDiscoveryService();

try {

    /*Aprovecho los servicios del netPeerGroup para definir el advertisement del
    módulo de implementación del grupo*/

    ModuleImplAdvertisement implAdv =
ntp.getAllPurposePeerGroupImplAdvertisement ();

    /*Otra vez aprovecho los servicios del netPeerGroup para crear el nuevo
    grupo*/

    pg = ntp.newGroup(null, // asigna un ID de grupo

        implAdv, // El advertisement del módulo de implementación
        nombre, // El nombre del grupo

        "Grupo para la tesis"); // Una descripción del grupo

    System.out.println("Se creo el grupo " + pg.getPeerGroupName() + " " +
pg.getPeerGroupAdvertisement().getName() );

    /*Se obtiene el advertisement del grupo creado*/

        adv=pg.getPeerGroupAdvertisement();

    catch (Exception e) {

        System.out.println("No se pudo crear el nuevo grupo, lo siento");

        return (null);

    }

    try {
```

*/*Publico el advertisement a los peers y peers rendezvous para actualizar la información de grupo con mi nuevo grupo creado, es decir ahora quiero formar parte del grupo ESPOL con los peers que igual que yo siguieron el proceso anterior*/*

```

        descubridor.remotePublish(adv);

        System.out.println("El grupo se publico con éxito.\n");
    }

    catch (Exception e) {

        System.out.println("No se pudo publicar el grupo, lo siento");

        return (null);

    }

    return(pg);
}

```

C.6 Servicio de membresías y credenciales

*/*Deseo ingresar al grupo ESPOL*/*

```

private static void unirseAlGrupo(PeerGroup grp) {

    System.out.println("Joining peer group...");

    StructuredDocument creds = null;

    try {

/* Creo un objeto credencial de autenticación*/

        AuthenticationCredential authCred = new

        AuthenticationCredential( grp, null, creds );
    }
}

```

```
/* Obtengo el servicio de membresía del grupo que fue creado (ESPOL)*/
```

```
MembershipService membership = grp.getMembershipService();
```

```
/* Creo un objeto autenticador para aplicarla con mi credencial de autenticación*/
```

```
Authenticator auth = membership.apply( authCred );
```

```
/* Si la credencial de autenticación fue aceptada*/
```

```
if (auth.isReadyForJoin()){
```

```
/*Creo mi credencial ya aprobada pasándole como parámetro el autenticador, al mismo tiempo que me uno al grupo*/
```

```
Credential myCred = membership.join(auth);
```

```
System.out.println("Me uní al grupo " + grp.getPeerGroupName() +  
" " + grp.getPeerGroupAdvertisement().getName() );  
}
```

C.7 Descubrimiento de peers

```
/*Se obtiene el servicio de descubrimiento del grupo ESPOL*/
```

```
discovery = peerGroup.getDiscoveryService();
```

```
public void run() {
```

```
Thread thisThread = Thread.currentThread();
```

```
try {
```

```
//espera por un descubrimiento
```

```
discovery.addDiscoveryListener(this);
```

```
while (hilo == thisThread) {
```

```

// búsqueda de peers
discovery.getRemoteAdvertisements(null, //por propagación
    DiscoveryService.PEER, //descubra advertisement tipo peer
    campo, //el campo por el cual desea buscar
    nombreABuscar, //el nombre a buscar
    10);

// espera un poco antes de enviar el próximo mensaje de descubrimiento
try {
    Thread.sleep(30 * 1000);
} catch(Exception e) {
    System.out.println("Error mandando a descubrir");
}
} //termina el while
} catch(Exception e) {
    e.printStackTrace();
}
}

//si hay un evento de descubrimiento de peer se llama al método
public synchronized void discoveryEvent(DiscoveryEvent ev) {

//recibo mensaje de respuesta

DiscoveryResponseMsg res = ev.getResponse();

PeerAdvertisement adv;

```

```

Usuario usu;

//se obtiene los advertisements que contienen mensajes
Enumeration en = res.getAdvertisements();

    if(en!=null)

//se compara lo obtenido con lo buscado
if(nombreABuscar==null || res.getQueryValue().equals(nombreABuscar)){

    while (en.hasMoreElements()) {

        adv = (PeerAdvertisement) en.nextElement();

        System.out.println (" Peer name = " + adv.getName());

        //obtengo el usuario que buscaba

        usu=new Usuario(adv);

    }

}

```

C.8 Pipe de entrada de mensajes

```

/*cuando un peer se quiere conectar conmigo*/

public void iniciaPipa() {

    /*Obtiene el servicio de pipes*/

    PipeService pipeService = pg.getPipeService();

    PipeAdvertisement pipeAdv=null;

    hilo=new Thread(this);

    try {

        /*Busca un pipe advertisement en su caché si no lo crea*/

```


Enumeration

```
en=pg.getDiscoveryService().getLocalAdvertisements(DiscoveryService.ADV
, "Name", "pipaEntradaEspol"+peer.getName());
```

```
    if(en!=null && en.hasMoreElements())
```

```
        pipeAdv=(PipeAdvertisement)en.nextElement();
```

```
    else{
```

```
        /*Crea el nuevo pipe advertisement*/
```

```
        pipeAdv = (PipeAdvertisement)
```

```
AdvertisementFactory.newAdvertisement(PipeAdvertisement.getAdvertisement
ntType());
```

```
        pipeAdv.setType(PipeService.UnicastType);
```

```
        pipeAdv.setDesc(peer.getDesc());
```

```
        pipeAdv.setName("pipaEntradaEspol"+peer.getName());
```

```
        pipeAdv.setPipeID(IDFactory.newPipeID(pg.getPeerGroupID() ) );
```

```
        pg.getDiscoveryService().publish(pipeAdv);
```

```
    }
```

```
    } catch (Exception e) {
```

```
        System.out.println("failed to read/parse pipe advertisement");
```

```
        e.printStackTrace();
```

```
    }
```

```
    try {
```

```

        System.out.println(pipeAdv.getPipeID());
        /*en espera de pedidos del pipe advertisement*/
        serverPipe=new JxtaServerPipe(pg, pipeAdv);
        serverPipe.setPipeTimeout(0);
        hilo.start();
    } catch (Exception e) {[
        e.printStackTrace();
    }
}

public void run() {
    System.out.println("starting ServerSocket");
    Thread currentThread=Thread.currentThread();
    JxtaBiDiPipe bidiPipe;
    while (hilo==currentThread) {
        try {
            System.out.println("Calling accept");
            /*Aquí crea el pipe bidireccional con un peer*/
            bidiPipe = serverPipe.accept();
            if (bidiPipe != null) {
                System.out.println("biDiPipe creada");
            }
        }
    }
}

```

```

        catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

C.9 pipe de salida de mensajes

*/*Esto es cuando uno se quiere conectar a otro peer*/*

//se obtiene el servicio de descubrimiento del grupo ESPOL

```
discovery = peerGroup.getDiscoveryService();
```

```
public void run() {
```

```
    Thread thisThread = Thread.currentThread();
```

```
    try {
```

```
        discovery.addDiscoveryListener(this);
```

```
        while (hilo == thisThread) {
```

```
            discovery.getRemoteAdvertisements(peerIdDestino, /*Peer con el
que quiere conectarse*/
```

```
                DiscoveryService.ADV, /*advertisement*/
```

```
                "Name", /*Busca por nombre*/
```

```
                criterio, /*el criterio*/
```

```
                5);
```

```
/*Espera un poco antes de enviar el próximo mensaje de descubrimiento*/
```

```
    try {
        Thread.sleep(30 * 1000);
    } catch(Exception e) {
        System.out.println("Error mandando a descubrir pipas");
    }
}
} catch(Exception e) {
    e.printStackTrace();
}
}
```

```
//si hay un evento de descubrimiento de peer se llama al método
```

```
public synchronized void discoveryEvent(DiscoveryEvent ev) {
    DiscoveryResponseMsg res = ev.getResponse();
    System.out.println("Me dispere evento pipa");
    Enumeration en = res.getAdvertisements();
    if(en!=null && res.getQueryValue().equals(criterio)){
        if (en.hasMoreElements()) {
            /*encontró el pipe advertisement destino*/
            advEncontrado = (PipeAdvertisement) en.nextElement();
            stop();
        }
    }
}
```

```
/*Crea un pipe bidireccional confiable*/
```

```

bidiPipe= new JxtaBiDiPipe();

bidiPipe.setReliable(true);

/*Se conecta con el pipe destino*/

        bidiPipe.connect(peerGroup , null, advEncontrado,180000, this);
    }
}
}

```

C.10 Pipe de entrada de archivos

```

/*Cuando un peer nos envía un archivo*/

public void iniciaSocket() {

    PipeService pipeService = pg.getPipeService();

/*Declaro un pipe advertisement para crear mi pipe de entrada*/

    PipeAdvertisement pipeAdv=null;

    hilo=new Thread(this);

    try {

/*Busca un pipe advertisement en su caché si no lo crea*/

        Enumeration

en=pg.getDiscoveryService().getLocalAdvertisements(DiscoveryService.ADV

, "Name", "socketEntradaEspol"+peer.getName());

        if(en!=null && en.hasMoreElements())

            pipeAdv=(PipeAdvertisement)en.nextElement();

        else{

```

```
/*Crea el nuevo pipe advertisement*/
```

```

        pipeAdv = (PipeAdvertisement)
AdvertisementFactory.newAdvertisement(PipeAdvertisement.getAdvertiseme
ntType());

        pipeAdv.setType(PipeService.UnicastType);
        pipeAdv.setDesc(peer.getDesc());
        pipeAdv.setName("socketEntradaEspol"+peer.getName());
        pipeAdv.setPipeID(IDFactory.newPipeID(pg.getPeerGroupID() ) );
        pg.getDiscoveryService().publish(pipeAdv);
    }
} catch (Exception e) {
    System.out.println("failed to read/parse pipe advertisement");
    e.printStackTrace();
}
try {
    System.out.println(pipeAdv.getPipeID());

```

```
/*En espera de conexión al socket*/
```

```

        serverSocket=new JxtaServerSocket(pg, pipeAdv, 10);
        serverSocket.setSoTimeout(0);
        hilo.start();
    } catch (Exception e) {
        e.printStackTrace();

```

```
    }  
}  
public void run() {  
    System.out.println("starting ServerSocket socket");  
    Thread currentThread=Thread.currentThread();  
    //JxtaSocket socket;  
    Socket socket;  
    while (hilo==currentThread) {  
        try {  
            System.out.println("Calling accept");  
/*Aquí crea el socket con un peer*/  
            socket = serverSocket.accept();  
            // set reliable  
            if (socket != null) {  
                Receptor rec=new Receptor(socket);  
                rec.start();  
            }  
        }  
        catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```

    }
}

```

C.11 Pipe de salida de archivos

*/*Cuando uno quiere enviar archivo a otro peer*/*

*/*se obtiene el servicio de descubrimiento del grupo ESPOL*/*

```
discovery = peerGroup.getDiscoveryService();
```

```
public void run() {
```

```
    Thread thisThread = Thread.currentThread();
```

```
    try {
```

```
        discovery.addDiscoveryListener(this);
```

```
        while (hilo == thisThread) {
```

```
            discovery.getRemoteAdvertisements(peerIdDestino, /*Peer con el que
```

quiere contactarse/*

```
                DiscoveryService.ADV, /*Advertisement*/
```

```
                "Name", /*Nombre a buscar*/
```

```
                criterio, 5);
```

*/*Espera un poco antes de enviar el próximo mensaje de descubrimiento*/*

```
        try {
```

```
            Thread.sleep(30 * 1000);
```

```
        } catch (Exception e) {
```

```
            System.out.println("Error mandando a descubrir pipas");
```

```
        }
```



```

    }
} catch(Exception e) {
    e.printStackTrace();
}
}

/*si hay un evento de descubrimiento de peer se llama al método*/
public synchronized void discoveryEvent(DiscoveryEvent ev) {
    DiscoveryResponseMsg res = ev.getResponse();
    Enumeration en = res.getAdvertisements();
    if(en!=null && res.getQueryValue().equals(criterio)){
        if (en.hasMoreElements()) {
            advEncontrado = (PipeAdvertisement) en.nextElement();
            stop();
        }
    }
}

/*inicia la transferencia del archivo*/
    iniciaTransferencia();
}
}

public synchronized void iniciaTransferencia()
{
    /*Se declara el socket*/
    JxtaSocket socket;

```

```

try{
    /*Se conecta el socket*/
        socket= new JxtaSocket(peerGroup, /*Grupo ESPOL*/
            null,
            advEncontrado, /*Advertisement para crear el pipe*/
            30000, /*Tiempo de espera para establecer la conexión
de 30 segundos*/
            true); /*Conexión confiable*/

        /*Se especifica el archive a enviar*/
        File f=new File(ruta);
        /*Se comienza a enviar el archivo*/
        Enviador env=new Enviador(dialogo, socket, f);
        env.start();
    }
    catch(java.io.IOException e){
        System.out.println("Error intentar abrir socket de archivo");
    }
}

```

C.12 El Advertisement de presencia

```

public abstract class PresenceAdvertisement extends Advertisement
{

```

```

/*El elemento raíz para el documento advertisement XML.*/
private static final String advertisementType = "PresenceAdvertisement";

/*El correo electrónico identificando al usuario cuya información de
presencia describe el advertisement.*/

private String emailAddress = null;

/*Un simple nombre para el usuario especificado por el correo electrónico
del advertisement.*/

private String name = null;

/*El peer ID que localiza el peer en la red.*/

private String peerID = null;

/*Un simple descriptor identificando el estado de presencia de los usuarios.
El usuario puede indicar que él o ella se encuentra conectado, desconectado,
etc.*/

private int presenceStatus = 0;

/*Retorna el tipo advertisement para el documento de advertisement.*/

private String mensaje="";

public static String getAdvertisementType()
{
    return advertisementType;
}

/*Retorna el string de correo electrónico describiendo al usuario cuyo
estado de presencia es descrito por este advertisement.*/

```

```
public String getEmailAddress()
{
    return emailAddress;
}

/*Retorna un identificador único para este documento. No existe para el
tipo advertisement, este método retorna el ID nulo.*/

public ID getID()
{
    return ID.nullID;
}

/*Retorna el nombre simple para el usuario descrito por este
advertisement.*/

public String getName()
{
    return name;
}

/*Retorna el Peer ID del usuario descrito por este advertisement.*/

public String getPeerID()
{
    return peerID;
}
```

*/*Retorna la información del estado de presencia del usuario descrito por el advertisement.*/*

```
public int getPresenceStatus()
{
    return presenceStatus;
}
```

```
public String getMensaje()
{
    return mensaje;
}
```

*/*Coloca el string del correo electrónico describiendo al usuario cuya presencia de estado es descrita por este advertisement.*/*

```
public void setEmailAddress(String emailAddress)
{
    this.emailAddress = emailAddress;
}
```

*/*Coloca el nombre simple para el usuario descrito por este advertisement.*/*

```
public void setName(String name)
{
    this.name = name;
}
```

*/*Coloca el Peer ID identificando la locación del peer en la red P2P.*/*

```
public void setPeerID(String peerID)
```

```
{
```

```
    this.peerID = peerID;
```

```
}
```

*/*Coloca la información del estado de presencia del usuario descrito por este advertisement.*/*

```
public void setPresenceStatus(int presenceStatus)
```

```
{
```

```
    this.presenceStatus = presenceStatus;
```

```
    /*0:No conectado
```

```
    *1:Conectado
```

```
    *2:No disponible
```

```
    *3:Vuelvo Enseguida
```

```
    *4:Ausente
```

```
    *5:Al telefono
```

```
    *6:Sali a comer
```

```
    *7:Personalizado*/
```

```
}
```

```
public void setMensaje(String mensaje)
```

```
{
```

```
    this.presenceStatus = 7;
```

```

        this.mensaje=mensaje;
    }
}

```

C.13 Publicar el Advertisement de presencia

*/*Publicar el advertisement*/*

```

public static void publicarEstado(int est, String mensaje){
    /*Se crea el advertisement*/
    PresenceAdv pAdv=new PresenceAdv();
    /*Se especifica el estado*/
    pAdv.setName("estado"+netPeerGroup.getPeerName());
    /*Se especifica el correo electrónico*/
    pAdv.setEmailAddress(netPeerGroup.getPeerName()+"@"+"espol");
    /*Verifica si es un mensaje personalizado o no*/
    if(est!=7)
        pAdv.setPresenceStatus(est);
    else
        pAdv.setMensaje(mensaje);
    /*Se especifica el peer ID*/
    pAdv.setPeerID(netPeerGroup.getPeerID().toString());
    try{
        grupoEspol.getDiscoveryService().publish(pAdv, 5*60*1000,
2*60*1000);
    }
}

```

```

        grupoEspol.getDiscoveryService().remotePublish(pAdv, 2*60*1000);
        System.out.println("Estado publicado con exito");
    }
    catch(Exception e){
        System.out.println("No se pudo publicar la información de estado");
    }
}

```

C.14 Descubrimiento de Advertisement de presencia

*/*Una implementación de la clase abstracta PresenceAdvertisement. Esta clase es responsable de parsear y formatear el documento XML usado para definir información de presencia de un peer.*/*

```

public class PresenceAdv extends PresenceAdvertisement
{

```

```

    /*Una constante conveniente para el tipo XML MIME.*/
    private static final String mimeType = "text/xml";

```

*/*El nombre del elemento para la información del correo electrónico del advertisement de presencia.*/*

```

    private static final String tagEmailAddress = "EmailAddress";

```

*/*El nombre del elemento para la información del nombre simple del advertisement de presencia.*/*

```

    private static final String tagName = "Name";

```



```
/*El nombre del elemento para el Peer ID del advertisement de presencia.*/
    private static final String tagPeerID = "PeerID";
/*El nombre del elemento para la información de estado del advertisement de
presencia.*/
    private static final String tagPresenceStatus = "PresenceStatus";
    private static final String tagMensaje = "Mensaje";
/*Un instanciador usado por el AdvertisementFactory para instanciar esta
clase.*/
    public static class Instantiator implements
AdvertisementFactory.Instantiator
    {
/*Retorna el nombre del elemento raíz del advertisement.*/
        public String getAdvertisementType()
        {
            return PresenceAdvertisement.getAdvertisementType();
        }
/*Retorna una nueva instancia del advertisement de presencia.*/
        public Advertisement newInstance()
        {
            return new PresenceAdv();
        }
    }
}
```

```
/*Instancia una nueva implementación de la instancias  
PresenceAdvertisement poblada del elemento raíz dado.*/  
public Advertisement newInstance(Element root)
```

```
{  
    return new PresenceAdv(root);  
}  
};
```

```
/*Crea un nuevo advertisement de presencia.*/  
public PresenceAdv()
```

```
{  
    super();  
}
```

```
/*Crea un nuevo advertisement de presencia parseando el stream dado.*/  
public PresenceAdv(InputStream stream) throws IOException
```

```
{  
    super();  
    StructuredTextDocument document = (StructuredTextDocument)  
    StructuredDocumentFactory.newStructuredDocument(  
        new MimeMediaType(mimeType), stream);  
    readAdvertisement(document);  
}
```

*/*Crear un nuevo advertisement de presencia parseando el documento
 dado.*/**

```
public PresenceAdv(Element document) throws IllegalArgumentException
{
    super();
    readAdvertisement((TextElement) document);
}

public String[] getIndexFields(){
    String[] indice={"Name"};
    return indice;
}
```

*/*Retorna un objeto documento conteniendo el árbol de documento del
 advertisement*/*

```
public Document getDocument(MimeMediaType asMimeType) throws
IllegalArgumentException
{
```

*/*Verifica que los elementos requeridos estén presentes.*/**

```
if ((null != getEmailAddress()) && (null != getPeerID()))
{
    PipeAdvertisement pipeAdv = null;
    StructuredDocument document = (StructuredTextDocument)
    StructuredDocumentFactory.newStructuredDocument(
```

```
        asMimeType, getAdvertisementType());  
    Element element;  
  
    /*Añade la información del Peer ID.*/  
    element = document.createElement(tagPeerID, getPeerID());  
    document.appendChild(element);  
  
    /*Añade la información del correo electrónico.*/  
    element = document.createElement(  
    tagEmailAddress, getEmailAddress());  
    document.appendChild(element);  
  
    /*Añade la información del nombre, si hay alguno.*/  
    if (null != getName())  
    {  
        element = document.createElement(tagName, getName());  
        document.appendChild(element);  
    }  
  
    /*Añade la información del estado de presencia.*/  
    element = document.createElement(tagPresenceStatus,  
    Integer.toString(getPresenceStatus()));  
    document.appendChild(element);  
    return document;  
}  
else
```

```

    {
        throw new IllegalArgumentException("Missing email address or peer
ID!");
    }
}

```

*/*Parsea el árbol de documento dado para el advertisement de presencia.**

```

public void readAdvertisement(TextElement document) throws
IllegalArgumentException
{
    if (document.getName().equals(getAdvertisementType()))
    {
        Enumeration elements = document.getChildren();
        while (elements.hasMoreElements())
        {
            TextElement element = (TextElement) elements.nextElement();

/*Verifica por el elemento del correo electrónico.*

            if (element.getName().equals(tagEmailAddress))
            {
                setEmailAddress(element.getTextValue());
                continue;
            }

/*Verifica por el elemento de nombre.*

```

```
if (element.getName().equals(tagName))
{
    setName(element.getTextValue());
    continue;
}
```

*/*Verifica por el elemento del correo electrónico.*/*

```
if (element.getName().equals(tagPresenceStatus))
{
    setPresenceStatus(
        Integer.parseInt(element.getTextValue()));
    continue;
}
```

*/*Verifica el elemento de muestra de nombre.*/*

```
if (element.getName().equals(tagMensaje))
{
    setMensaje(element.getTextValue());
    continue;
}
```

*/*Verifica el elemento Peer ID.*/*

```
if (element.getName().equals(tagPeerID))
{
    setPeerID(element.getTextValue());
}
```

```
        continue;
    }
}
else
{
    throw new IllegalArgumentException("Not a PresenceAdvertisement
document!");
}
}

/* Retorna un string de XML representando este advertisement.*/
public String toString()
{
    try
    {
        StringWriter out = new StringWriter();
        StructuredTextDocument doc = (StructuredTextDocument)
            getDocument(new MimeMediaType(mimeType));
        doc.sendToWriter(out);
        return out.toString();
    }
    catch (Exception e)
```

```

    {
        return "";
    }
}
}
}

```

C.15 Actualizar el Advertisement de presencia

*/*Descubre el advertisement de presencia*/*

```

public class ActualizadorPeerInfo implements DiscoveryListener {
    private static PeerGroup peerGroup;
    private DiscoveryService discovery;
    private Usuario peer;
    private String criterio;
    public ActualizadorPeerInfo(PeerGroup pg, Usuario usu) {
        peerGroup=pg;
        discovery = peerGroup.getDiscoveryService();
        discovery.addDiscoveryListener(this);
        this.peer=usu;
        criterio="estado"+peer.getEspolld();
    }
    public void buscar() {
        try {

```



```

        System.out.println("Mande a buscar " + criterio + "-");
        discovery.getRemoteAdvertisements(null, DiscoveryService.ADV,
"Name", criterio, 5);
    } catch(Exception e) {
        e.printStackTrace();
    }
}

public void discoveryEvent(DiscoveryEvent ev) {
    DiscoveryResponseMsg response = ev.getResponse();
    /*Extrae el PresenceAdvertisement de la respuesta.*/
    System.out.println("Evento encontrado, criterio:" +
response.getQueryValue());
    if(response.getQueryValue().equals(criterio)){
        Enumeration responses = response.getResponses();
        while (responses.hasMoreElements())
        {
            String responseElement = (String) responses.nextElement();
            /*Verifica por un advertisement de respuesta nula.*/
            if (null != responseElement)
            {
                /*Se parsea advertisement.*/          try
                {

```

```
        java.io.ByteArrayInputStream stream =
            new java.io.ByteArrayInputStream(
                responseElement.getBytes());

        peer.actualizarEstado(advertisement) ;
        principal.PanelPrincipal.listaContactos.repaint();
        System.out.println(advertisement.getPresenceStatus());
    }
    catch (java.io.IOException e)
    {
        System.out.println("Error in discoveryEvent: " + e);
    }
}
else
{
    System.out.println("Response advertisement is null!");
}
}
}
}
}
```

C.16 El Advertisement de cartelera

```
/*Una clase abstracta definiendo un advertisement conteniendo los
elementos usados para describir los elementos de la cartelera. Se asume
que un usuario es único descrito por su correo electrónico. */
public abstract class AnuncioAdvertisement extends Advertisement
{
/*El elemento raíz para el documento XML de advertisement*/
    private static final String advertisementType = "AnuncioAdvertisement";
/*El correo electrónico identificando al usuario cuya información de cartelera
este advertisement describe.*/
    private String mensaje = "";
    private String titulo = "";
    private String user="";
/*Un simple nombre para el usuario especificado por el correo electrónico del
advertisement. */
    private String name = null;
/*El Peer ID localizando el peer en la red.*/
    private String peerID = null;
/*Retorna el tipo advertisement para el documento advertisement.*/
    public static String getAdvertisementType()
    {
        return advertisementType;
    }
}
```

```
public String getTitulo()
```

```
{
```

```
    return titulo;
```

```
}
```

```
public String getMensaje()
```

```
{
```

```
    return mensaje;
```

```
}
```

```
public String getUser()
```

```
{
```

```
    return user;
```

```
}
```

*/*Retorna un identificador único para este documento. No existe para el tipo advertisement, este método retorna el ID nulo.*/*

```
public ID getID()
```

```
{
```

```
    return ID.nullID;
```

```
}
```

*/*Retorna el nombre simple para el usuario descrito por este advertisement.*/*

```
public String getName()
```

```
{
```

```
    return name;
```

```
    }  
  
    /*Retorna el Peer ID del usuario descrito por este advertisement.*/  
  
    public String getPeerID()  
    {  
        return peerID;  
    }  
  
    /*Describe el titulo, mensaje y usuario.*/  
  
    public void setTitulo(String titulo)  
    {  
        this.titulo = titulo;  
    }  
  
    public void setMensaje(String mensaje)  
    {  
        this.mensaje = mensaje;  
    }  
  
    public void setUser(String user)  
    {  
        this.user=user;  
    }  
  
    /*Coloca el nombre simple para el usuario descrito por este advertisement.*/  
  
    public void setName(String name)  
    {
```

```

        this.name = name;
    }
    /*Coloca el Peer ID identificando la locación del peer en la red P2P.*/
    public void setPeerID(String peerID)
    {
        this.peerID = peerID;
    }
}

```

C.18 Implementacion del Advertisement de cartelera

*/*Una implementación de la clase abstracta AnuncioAdvertisement. Esta clase es responsable de parsear y formatear el documento XML usado para definir información de los anuncios en la cartelera.*/*

```

public class AnuncioAdv extends AnuncioAdvertisement
{
    static
    {
        net.jxta.document.AdvertisementFactory.registerAdvertisementInstance(getAdvertisementType(),new AnuncioAdv.Instantiator());
    }
    /*Una constante conveniente para el tipo XML MIME.*/
    private static final String mimeType = "text/xml";
}

```

```
    /*El nombre del elemento para la información del correo electrónico del  
advertisement de cartelera.*/
```

```
    private static final String tagTitulo = "Titulo";
```

```
    private static final String tagMensaje = "Mensaje";
```

```
    private static final String tagUser = "User";
```

```
    /*El nombre del elemento para la información del nombre simple del  
advertisement de presencia.*/
```

```
    private static final String tagName = "Name";
```

```
    /*El nombre del elemento para el Peer ID del advertisement de  
presencia.*/
```

```
    private static final String tagPeerID = "PeerID";
```

```
    /*Un instanciador usado por el AdvertisementFactory para instanciar esta  
clase.*/
```

```
    public static class Instantiator implements
```

```
AdvertisementFactory.Instantiator
```

```
{
```

```
    /*Retorna el nombre del elemento raíz del advertisement.*/
```

```
    public String getAdvertisementType()
```

```
{
```

```
        return AnuncioAdvertisement.getAdvertisementType();
```

```
}
```

*/*Retorna una nueva instancia del advertisement de cartelera.*/**

```
public Advertisement newInstance()  
{  
    return new AnuncioAdv();  
}
```

*/*Instancia una nueva implementación de la instancias*

PresenceAdvertisement poblada del elemento raíz dado./**

```
public Advertisement newInstance(Element root)  
{  
    return new AnuncioAdv(root);  
}  
};
```

*/*Crea un nuevo advertisement de cartelera.*/**

```
public AnuncioAdv()  
{  
    super();  
}
```

*/*Crea un nuevo advertisement de presencia parseando el stream dado.*/**

```
public AnuncioAdv(InputStream stream) throws IOException  
{  
    super();  
    StructuredTextDocument document = (StructuredTextDocument)
```



```
        StructuredDocumentFactory.newStructuredDocument(  
            new MimeMediaType(mimeType), stream);  
        readAdvertisement(document);  
    }  
  
    /*Crear un nuevo advertisement de presencia parseando el documento  
    dado.*/  
  
    public AnuncioAdv(Element document) throws IllegalArgumentException  
    {  
        super();  
        readAdvertisement((TextElement) document);  
    }  
  
    public String[] getIndexFields(){  
        String[] indice={"Name"};  
        return indice;  
    }  
  
    /*Retorna un objeto documento conteniendo el árbol de documento del  
    advertisement*/  
  
    public Document getDocument(MimeMediaType asMimeType) throws  
    IllegalArgumentException  
    {  
        /*Verifica que los elementos requeridos estén presentes.*/  
    }  
    }  
}
```

```
if (null != getPeerID())
{
    PipeAdvertisement pipeAdv = null;

    StructuredDocument document = (StructuredTextDocument)
    StructuredDocumentFactory.newStructuredDocument(
        asMimeType, getAdvertisementType());

    Element element;

    /*Añade la información del Peer ID.*/

    element = document.createElement(tagPeerID, getPeerID());
    document.appendChild(element);

    /*Añade la información del correo electrónico.*/

    element = document.createElement(
        tagTitulo, getTitulo());
    document.appendChild(element);

    /*Añade la información de unidad.*/

    element = document.createElement(
        tagMensaje, getMensaje());
    document.appendChild(element);

    /*Añade la información del usuario.*/

    element = document.createElement(
        tagUser, getUser());
    document.appendChild(element);
```

*/*Añade la información del mensaje, si hay alguno.*/*

```
    if (null != getName())
    {
        element = document.createElement(tagName, getName());
        document.appendChild(element);
    }
    return document;
}
else
{
    throw new IllegalArgumentException("Missing peer ID!");
}
}
```

*/*Parsea el árbol de documento dado para el advertisement de presencia.*/*

```
public void readAdvertisement(TextElement document) throws
IllegalArgumentException
{
    if (document.getName().equals(getAdvertisementType()))
    {
        Enumeration elements = document.getChildren();
        while (elements.hasMoreElements())
        {
```

```
TextElement element = (TextElement) elements.nextElement();

/*Verifica por el elemento del correo electrónico.*/
if (element.getName().equals(tagTitulo))
{
    setTitulo(element.getTextValue());
    continue;
}

/*Verifica por el elemento del nombre.*/
if (element.getName().equals(tagMensaje))
{
    setMensaje(element.getTextValue());
    continue;
}

/*Verifica por el elemento del correo electrónico.*/
if (element.getName().equals(tagUser))
{
    setUser(element.getTextValue());
    continue;
}

/*Verifica el elemento de muestra de nombre.*/
if (element.getName().equals(tagName))
{
```

```
        setName(element.getTextValue());
        continue;
    }

    /*Verifica el elemento Peer ID.*/
    if (element.getName().equals(tagPeerID))
    {
        setPeerID(element.getTextValue());
        continue;
    }
}
}
else
{
    throw new IllegalArgumentException("Not a PresenceAdvertisement
document!");
}
}

/* Retorna un string de XML representando este advertisement.*/
public String toString()
{
    try
    {
```

```

        StringWriter out = new StringWriter();
        StructuredTextDocument doc = (StructuredTextDocument)
        getDocument(new MimeMediaType(mimeType));
        doc.sendToWriter(out);
        return out.toString();
    }
    catch (Exception e)
    {
        return "";
    }
}
}

```

C.18 Publicar el Advertisement de cartelera

*/*Publicar anuncio*/*

```

public static void publicarAnuncio(String titulo, String mensaje){

    AnuncioAdvertisement anunc;

    try{

        anunc=new AnuncioAdv();

        anunc.setPeerID(netPeerGroup.getPeerID().toString());

        anunc.setName("anuncio");

        anunc.setUser(netPeerGroup.getPeerName());
    }
}

```

```

        anunc.setTitulo(titulo);
        anunc.setMensaje(mensaje);
        int quinceDias=15*(24*60*60*1000);
        grupoEspol.getDiscoveryService().publish(anunc, quinceDias,
quinceDias);
        grupoEspol.getDiscoveryService().remotePublish(anunc, quinceDias);
        System.out.println("Anuncio publicado con exito");
    }
    catch(Exception e){
        System.out.println("No se pudo publicar el anuncio");
    }

```

C.19 Descubrimiento de Advertisement de cartelera

```

public class BuscadorAnuncio implements DiscoveryListener {
    private static PeerGroup peerGroup;
    private DiscoveryService discovery;
    private Cartelera ventana;
    public BuscadorAnuncio(PeerGroup pg, Cartelera ventana) {
        peerGroup=pg;
        this.ventana=ventana;
        discovery = peerGroup.getDiscoveryService();
        discovery.addDiscoveryListener(this);
    }

```

```

}

public void buscar() {
    try {
        System.out.println("Mande a buscar anuncio");

        /*Restrinjo a que devuelvan resultados solo 10 anuncios*/
        discovery.getRemoteAdvertisements(null, DiscoveryService.ADV,
"Name", "anuncio", 10);
    } catch(Exception e) {
        e.printStackTrace();
    }
}

public void discoveryEvent(DiscoveryEvent ev) {
    DiscoveryResponseMsg response = ev.getResponse();

    /*Extra el PresenceAdvertisement de la respuesta*/

    System.out.println("Evento disparado:" + response.getQueryValue());
    if(response.getQueryValue().equals("anuncio")){
        Enumeration responses = response.getResponses();
        while (responses.hasMoreElements())
        {
            String responseElement = (String) responses.nextElement();

            /*Verifica por advertisements de respuesta nula*/
            if (null != responseElement)

```



```
{  
    /*Parsea el advertisement*/  
    try  
    {  
        java.io.ByteArrayInputStream stream =  
            new java.io.ByteArrayInputStream(  
                responseElement.getBytes());  
        AnuncioAdvertisement advertisement = new  
AnuncioAdv(stream);  
        ventana.escribirMensaje(advertisement);  
    }  
    catch (java.io.IOException e)  
    {  
        System.out.println("Error in discoveryEvent: " + e);  
    }  
}  
else  
{  
    System.out.println("Response advertisement is null!");  
}  
}
```

```

    }
}

```

C.20 Búsqueda de contactos por usuario

*/*Se va a realizar una búsqueda de usuario en el grupo ESPOL*/*

```

public Busqueda(PeerGroup pg, BusquedaUsuario bu) {
    peerGroup=pg;

```

*/*Se hace un discovery para el descubrimiento del usuario*/*

```

    discovery = peerGroup.getDiscoveryService();
    this.bu=bu;

```

```

}

```

```

public void start() {

```

```

    hilo = new Thread(this);

```

```

    hilo.start();

```

```

}

```

```

public void run() {

```

```

    Thread thisThread = Thread.currentThread();

```

```

    try {

```

```

        discovery.addDiscoveryListener(this);

```

```

        while (hilo == thisThread) {

```

*/*Buscar un peer*/*

```

System.out.println ("Mande a buscar");

queryId=discovery.getRemoteAdvertisements(null,
                                           DiscoveryService.PEER,
                                           campo,
                                           nombreABuscar, 10);

/*Espera un poco antes de enviar el siguiente mensaje de descubrimiento*/

    try {
        Thread.sleep(30 * 1000);
    } catch(Exception e) {
        System.out.println("Error mandando a descubrir");
    }

} //termina el while
} catch(Exception e) {
    e.printStackTrace();
}
}

public synchronized void discoveryEvent(DiscoveryEvent ev) {
    DiscoveryResponseMsg res = ev.getResponse();
    PeerAdvertisement adv;
    Usuario usu;
    System.out.println("Me dispere evento");
    System.out.println(res.getQueryValue() + " " + nombreABuscar);
}

```

```
Enumeration en = res.getAdvertisements();

if(en!=null)

if(nombreABuscar==null||res.getQueryValue().equals(nombreABuscar)){

    while (en.hasMoreElements()) {

        adv = (PeerAdvertisement) en.nextElement();

        System.out.println (" Peer name = " + adv.getName());

        String[] fields=adv.getIndexFields();

        for(int i=0; i<fields.length; i++){

            System.out.println(fields[i]);

        }

        usu=new Usuario(adv);

        bu.agregarUsuario(usu);

    }

}

public void setNombreABuscar(String nombre){

    if(nombre.equals(""))

    {

        this.campo=null;

        this.nombreABuscar=null;

    }

    else
```

```
{
    this.campo="Name";
    this.nombreABuscar=nombre+"*";
}
}
/*La búsqueda se detiene*/
public void stop(){
    discovery.removeDiscoveryListener(this);
    hilo=null;
}
```

PRUEBAS DE USUARIO

D.1 Cuestionario de Funcionalidades

Aplicación: ESPOL Messenger

Encuestado: _____

Llene los siguientes datos:

Nombre: _____

Edad: _____

Formación: _____

Encierre la selección más apropiada:

Experiencia con aplicaciones de mensajería instantánea (Messenger)

Uso de algún tipo de Messenger Diariamente Semanalmente
 Mensualmente Nunca

Uso de carteleras de anuncios en la Web Diariamente Semanalmente
 Mensualmente Nunca

Tarea 1: Ingresar al chat.		Muy Fácil	Fácil	Intermedio	Difícil	Muy Difícil
Sub-tareas:	1	Abrir la aplicación.				
	2	Ingresar los datos requeridos de usuario, contraseña, facultad y nick (seudónimo).				
	3	Oprimir conectar.				

Tarea 2: Cambiar estado.		Muy Fácil	Fácil	Intermedio	Difícil	Muy Difícil
Sub-tareas:	1	Localizar el menú de archivo.				
	2	Localizar el menú de estado.				
	3	Seleccionar el estado deseado.				
	4	Seleccionar la opción de estado personalizado.				

	5	Ingresar el estado deseado en el espacio de texto.					
	6	Oprimir aceptar.					

Tarea 3: Cambiar datos de usuario.			Muy Fácil	Fácil	Intermedio	Difícil	Muy Difícil
Sub-tareas:	1	Localizar el menú de archivo.					
	2	Localizar el menú de datos.					
	3	Seleccionar la facultad deseada.					
	4	Ingresar el nick deseado en el espacio de texto.					
	5	Oprimir aceptar.					

Tarea 4: Configuración de Proxy.			Muy Fácil	Fácil	Intermedio	Difícil	Muy Difícil
Sub-tareas:	1	Localizar el menú de archivo.					
	2	Localizar el menú de opciones.					
	3	Ingresar los datos necesarios para la conexión como son Servidor Proxy, Puerto, Usuario y Contraseña.					
	4	Oprimir aceptar					

Tarea 5: Agregar Contacto.			Muy Fácil	Fácil	Intermedio	Difícil	Muy Difícil
Sub-tareas:	1	Localizar el menú de Contactos.					
	2	Localizar el menú de Agregar contacto.					
	3	Ingresar el usuario al espacio de texto.					

	4	Oprimir agregar.					
	5	Ubicarse en la ventana de chat de un usuario que no se encuentra agregado en mi lista de contactos.					
	6	Oprimir el botón de Agregar a Contactos					

Tarea 6: Eliminar contacto.			Muy Fácil	Fácil	Intermedio	Difícil	Muy Difícil
Sub-tareas:	1	Oprimir clic derecho sobre el contacto a eliminar.					
	2	Escoger eliminar contacto.					
	3	Escoger que está seguro de eliminar ese contacto.					
	4	Oprimir aceptar.					

Tarea 7: Buscar un usuario.			Muy Fácil	Fácil	Intermedio	Difícil	Muy Difícil
Subtare as:	1	Localizar el menú de Contactos.					
	2	Localizar el menú de Buscar usuarios.					
	3	Ingresar el usuario a buscar.					
	4	Oprimir buscar.					
	5	Oprimir detener para parar la búsqueda si se lo desea.					
	6	Elegir los usuarios a agregar					

		seleccionando su respectivo check box.					
	7	Oprimir agregar.					

Tarea 8: Abrir una ventana de conversación con otro usuario y chatear.		Muy Fácil	Fácil	Intermedio	Difícil	Muy Difícil
Sub-tareas :	1	Oprimir doble clic sobre un usuario conectado.				
	2	Oprimir clic derecho sobre el usuario con el cual se desea chatear y escoger enviar mensaje.				
	3	Localizar el menú de Enviar mensaje en Acciones.				
	4	Seleccionar de la lista presentada de usuarios conectados el usuario con el que se quiere chatear.				
	5	Oprimir aceptar.				
	6	Escribir un mensaje en el espacio de texto inferior.				
	7	Oprimir enviar.				
	8	Leer sin problemas la pantalla de conversaciones.				

Tarea 9: Invitar a otro contacto a la conversación.		Muy Fácil	Fácil	Intermedio	Difícil	Muy Difícil
Sub-tareas :	1	Ubicarse en la ventana de chat y oprimir el menú de Contactos.				

	2	Localizar el menú de Invitar un contacto a esta conversación.					
	3	Seleccionar de la lista presentada de usuarios conectados el usuario que se desea invitar.					
	4	Oprimir aceptar.					

Tarea 10: Guardar conversación.			Muy Fácil	Fácil	Intermedio	Difícil	Muy Difícil
Sub-tareas:	1	Ubicarse en la ventana de chat y oprimir el menú de Archivo.					
	2	Localizar el menú de Guardar conversación.					
	3	Seleccionar la dirección donde se guardará la conversación en un archivo de texto.					
	4	Oprimir aceptar.					

Tarea 11: Guardar lista de contactos.			Muy Fácil	Fácil	Intermedio	Difícil	Muy Difícil
Sub-tareas :	1	Localizar el menú de Contactos.					
	2	Localizar el menú de Guardar lista de contactos.					
	3	Seleccionar la dirección donde se guardará el archivo XML con todos los contactos que se tiene hasta ahora en la					

		aplicación.					
	4	Oprimir aceptar.					

Tarea 12: Leer lista de contactos.		Muy Fácil	Fácil	Intermedio	Difícil	Muy Difícil
Sub-tareas:	1	Localizar el menú de Contactos.				
	2	Localizar el menú de Leer lista de contactos.				
	3	Seleccionar la dirección donde se encuentra el archivo XML con todos los contactos que se tiene hasta ahora en la aplicación.				
	4	Oprimir aceptar.				

Tarea 13: Crear grupo.		Muy Fácil	Fácil	Intermedio	Difícil	Muy Difícil
Sub-tareas:	1	Localizar el menú de Grupos.				
	2	Localizar el menú de Crear grupo.				
	3	Ingresar el nombre deseado para el grupo en el espacio de texto.				
	4	Oprimir aceptar.				

Tarea 14: Editar grupo.		Muy Fácil	Fácil	Intermedio	Difícil	Muy Difícil
Sub-tareas:	1	Oprimir clic derecho sobre el grupo a editar.				
	2	Escoger editar grupo.				
	3	Ingresar el nuevo nombre del grupo, presionar enter.				

Tarea 15: Eliminar grupo.		Muy Fácil	Fácil	Intermedio	Difícil	Muy Difícil
Sub-tareas:	1	Localizar el menú de Grupos.				

	2	Localizar el menú de Eliminar grupo.					
	3	Seleccionar el grupo a eliminar de una lista.					
	4	Oprimir eliminar.					
	5	Oprimir clic derecho sobre el grupo a eliminar.					
	6	Escoger eliminar grupo.					
	7	Escoger que se está seguro de eliminar el grupo.					
	8	Oprimir aceptar.					

Tarea 16: Mover a los contactos en grupos específicos.		Muy Fácil	Fácil	Intermedio	Difícil	Muy Difícil
Sub-tareas:	1	Localizar el contacto que se quiere ubicar.				
	2	Oprimir clic derecho y escoger mover a.				
	3	Escoger el grupo al cual desea mover al contacto.				

Tarea 17: Enviar archivo.		Muy Fácil	Fácil	Intermedio	Difícil	Muy Difícil
Sub-tareas:	1	Localizar el menú de Acciones.				
	2	Localizar el menú de Enviar archivo.				
	3	Seleccionar el contacto conectado al que se desea enviar el archivo de una lista que se presenta.				
	4	Escoger la ruta donde se encuentra el archivo a				

		enviar.					
	5	Oprimir aceptar.					
	6	Hacer clic derecho sobre el contacto que se desea reciba el archivo.					
	7	Escoger la acción de enviar archivo.					
	8	Colocarse en la ventana de conversación del usuario al que se desea enviar el archivo.					
	9	Localizar el menú de archivo.					
	10	Localizar el menú de enviar archivo.					

Tarea 18: Leer y publicar un anuncio en la cartelera.			Muy Fácil	Fácil	Intermedio	Difícil	Muy Difícil
Sub-tareas:	1	Localizar el menú de Acciones.					
	2	Localizar el menú de Leer/Publicar anuncio en cartelera.					
	3	Oprimir el botón que dice cartelera y se muestra en la ventana principal del chat.					
	4	Proceder a leer la cartelera.					
	5	Oprimir publicar.					
	6	Escribir el asunto y el mensaje en los espacios de texto.					
	7	Oprimir aceptar.					

D.2 Cuestionario sobre la interfaz

Por favor califique si está de acuerdo o en desacuerdo con las siguientes declaraciones:

Pregunta	Totalmente de acuerdo	De acuerdo	Neutral	En desacuerdo	Totalmente en desacuerdo
El lenguaje utilizado en los menús y submenús es claro y fácil de entender.					
Es fácil encontrar el procedimiento para realizar una acción específica.					
Presenta las funciones de una manera estéticamente agradable.					
Los colores utilizados en la aplicación son de su agrado.					
Los íconos y figuras son fácilmente identificables y agradables.					
Las funciones en la pantalla principal son fáciles de predecir.					
El uso de la cartelera es amigable.					

¿Cuál cree que podría ser un buen nombre para la aplicación de chat de la ESPOL?

ESPOL Messenger

Javix

Espolssenger

EspolPeer

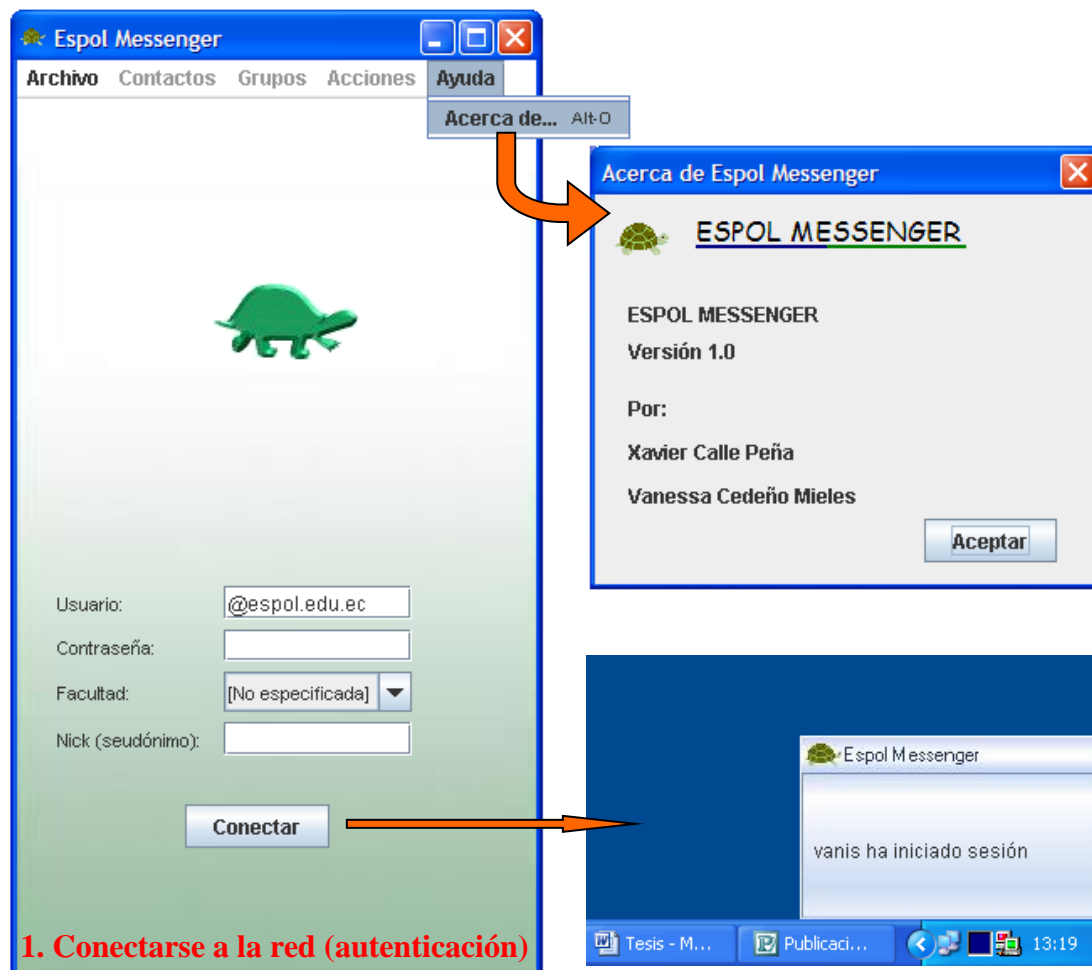
PoliJxta

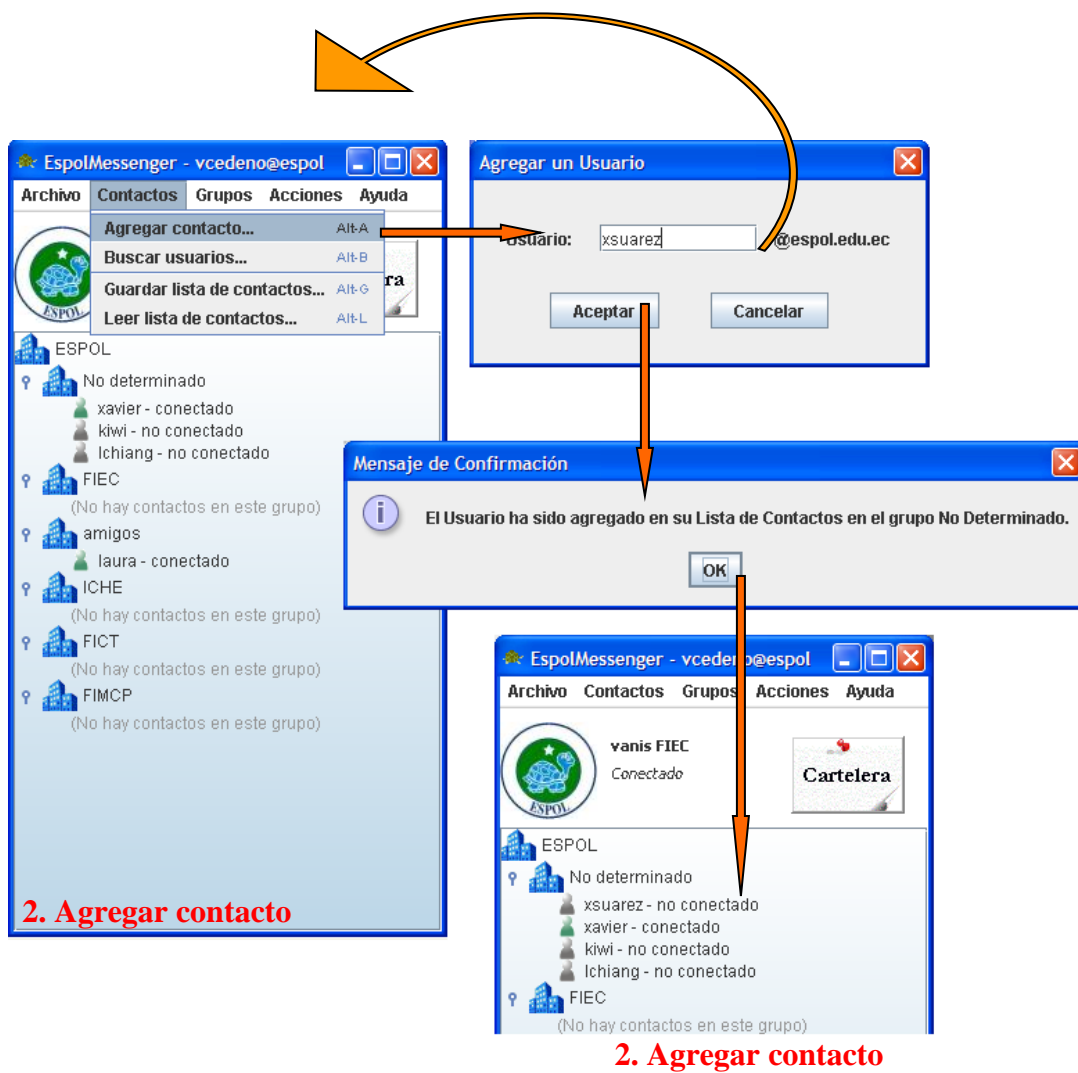
Poli-Chat

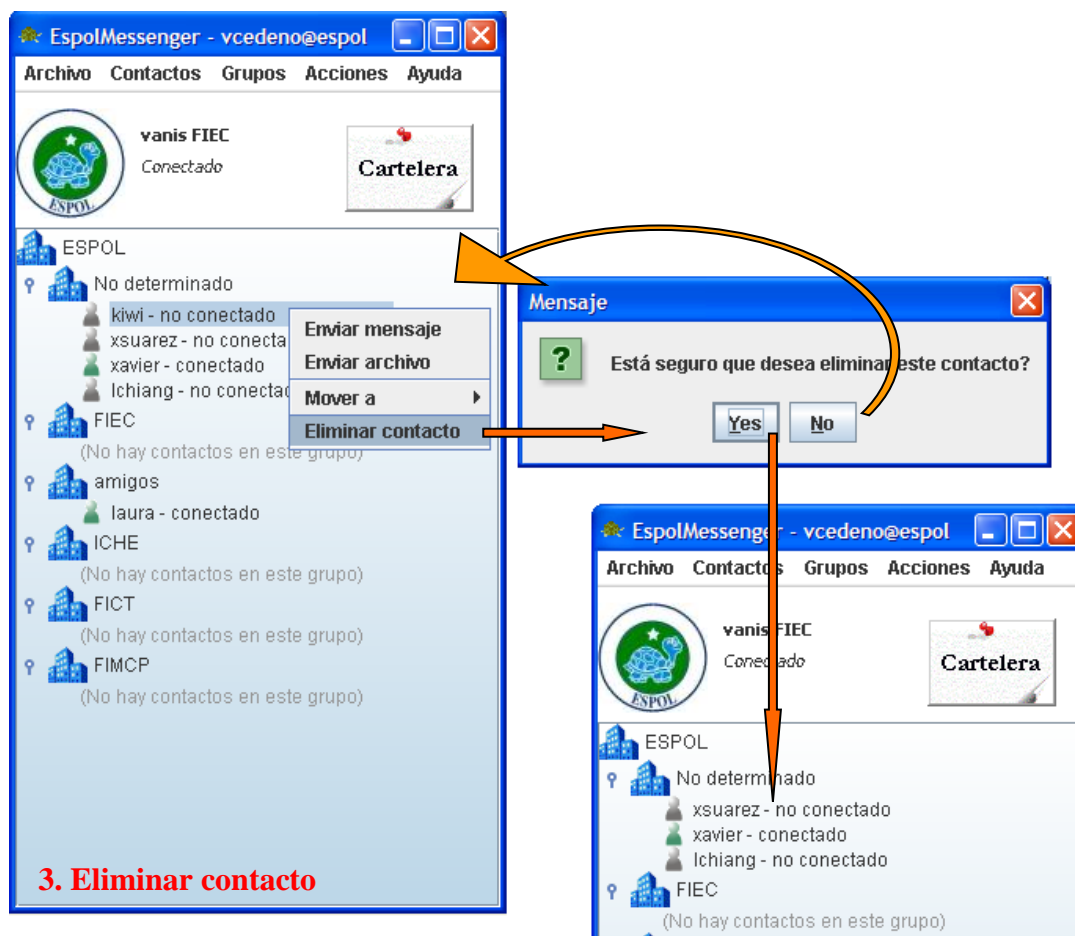
Por favor añada alguna recomendación para cambios en el diseño, lenguaje o procedimientos:

Gracias por su participación en la prueba de este prototipo.

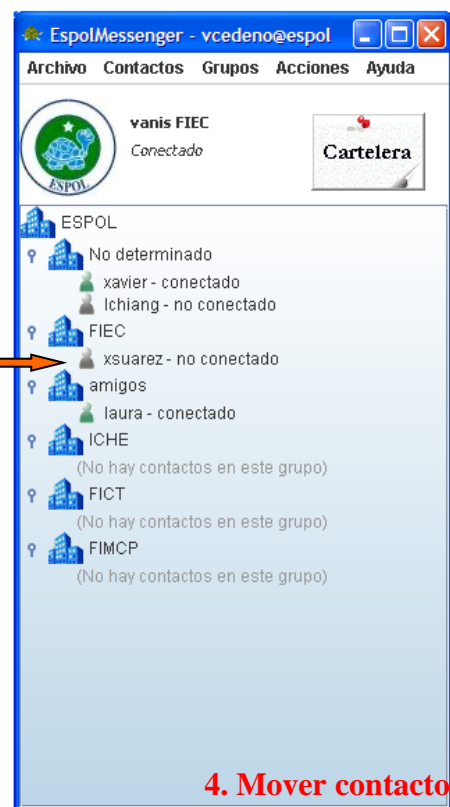
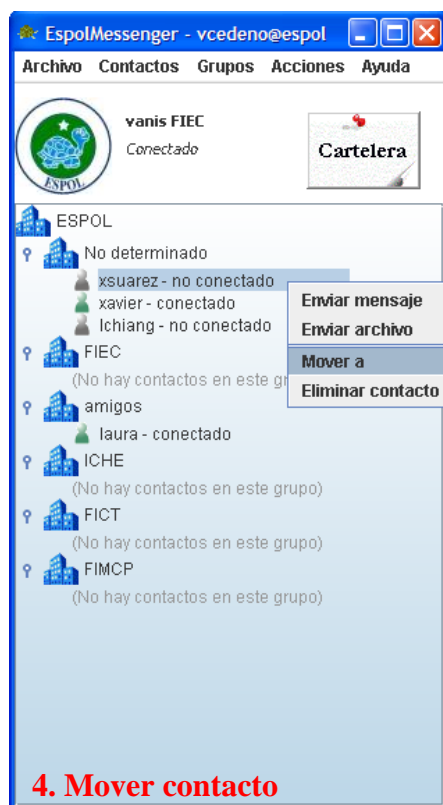
FLUJO DE VENTANAS

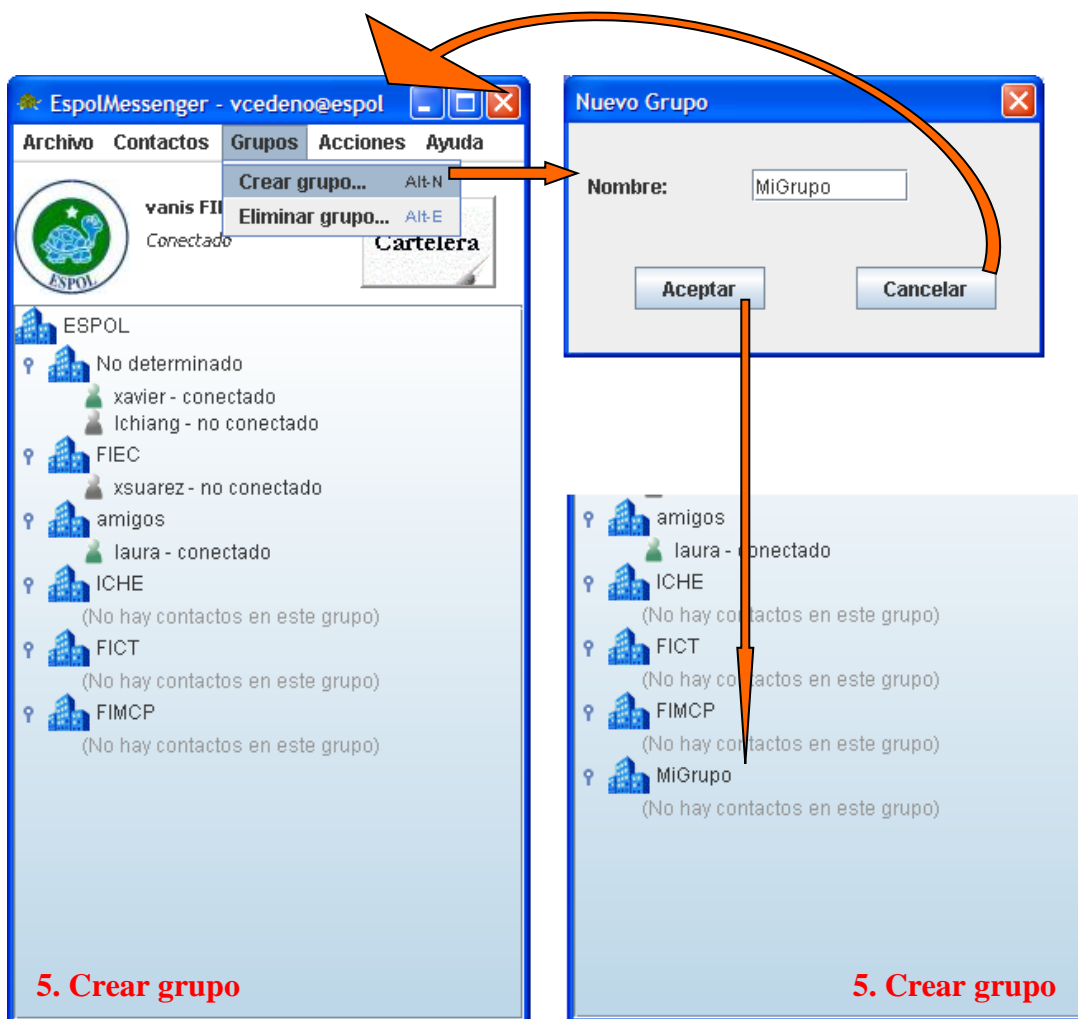


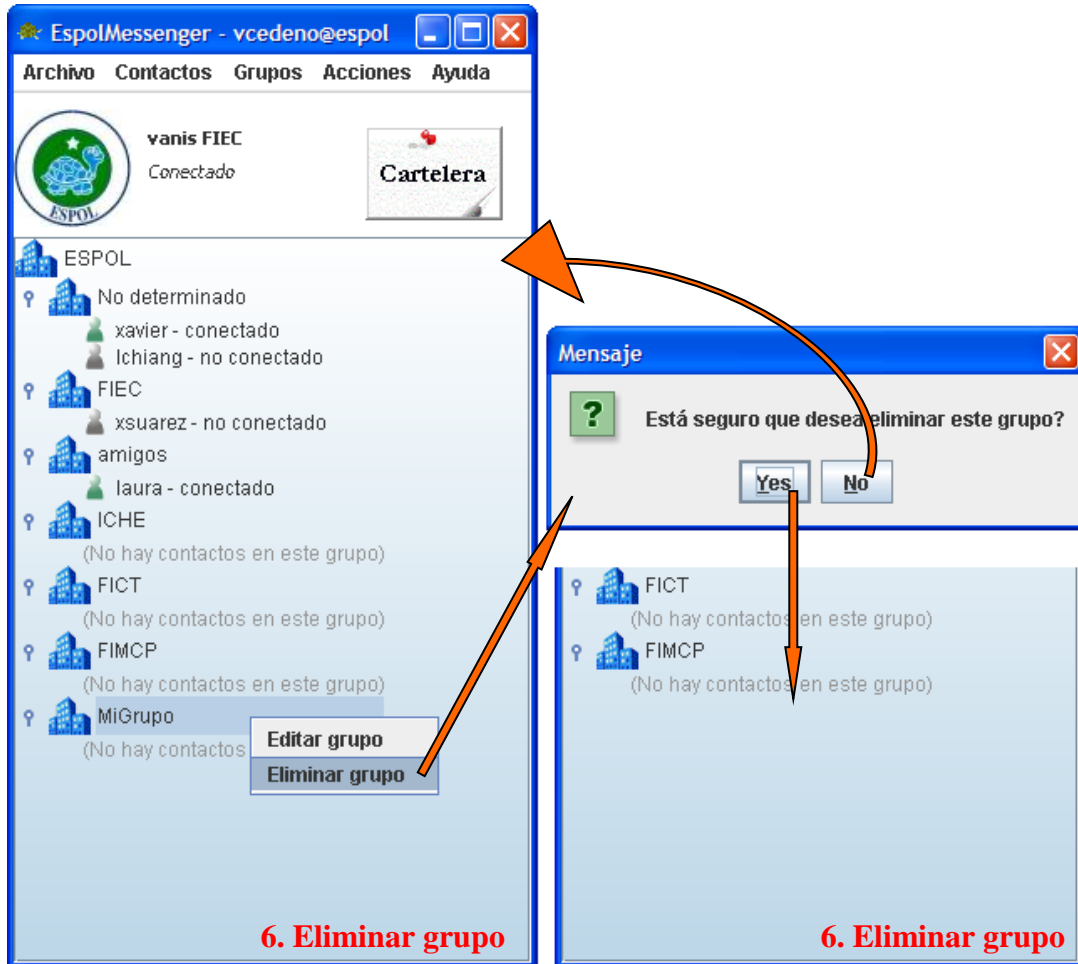


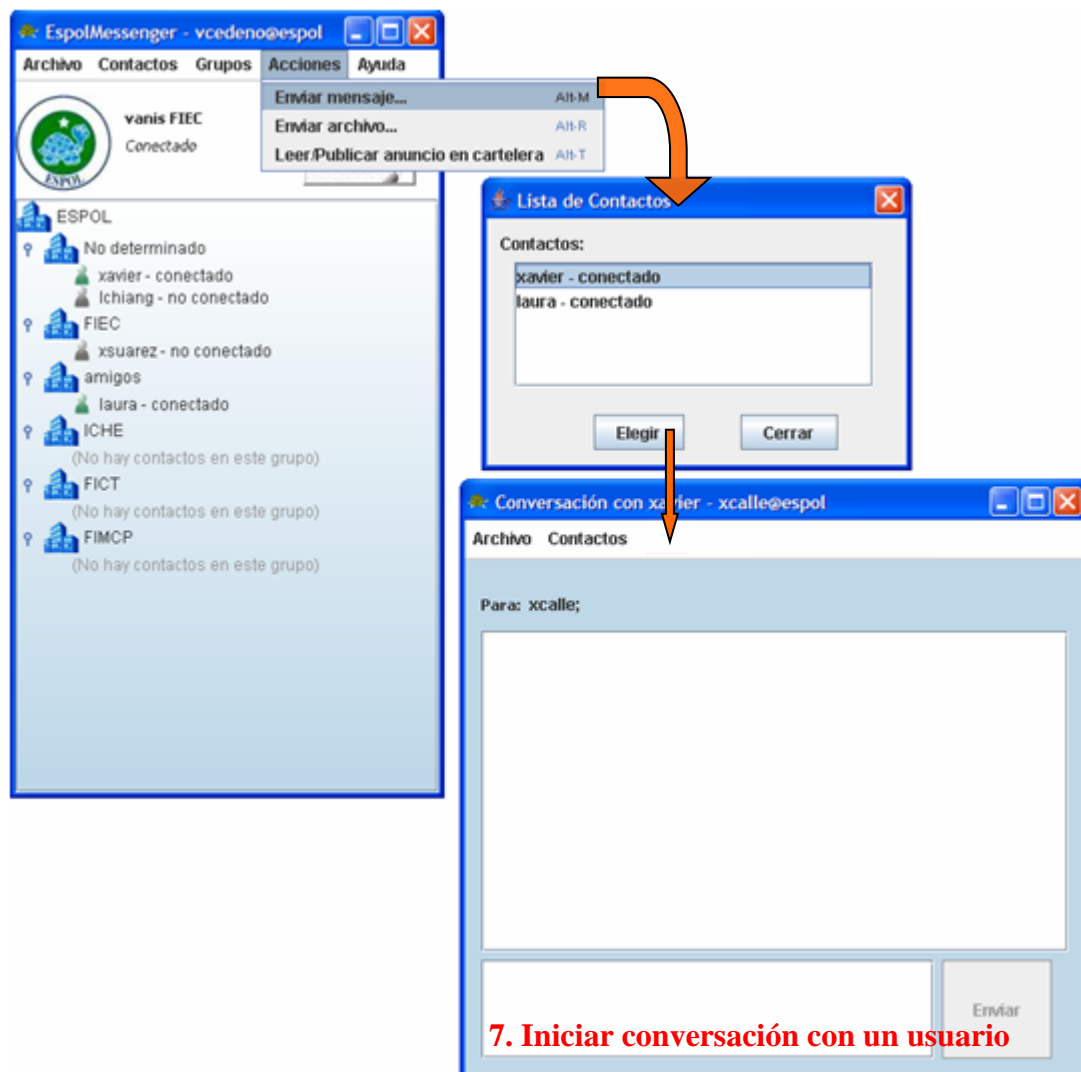


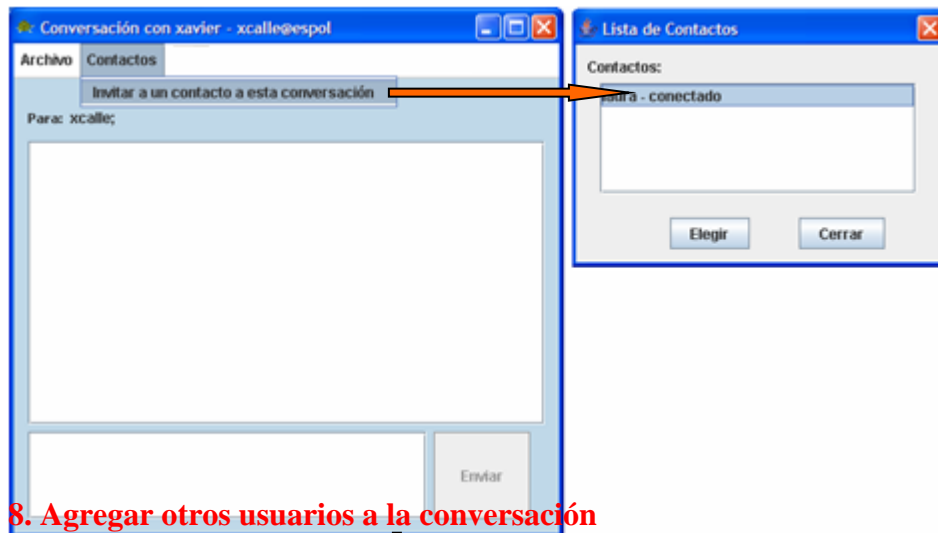
3. Eliminar contacto



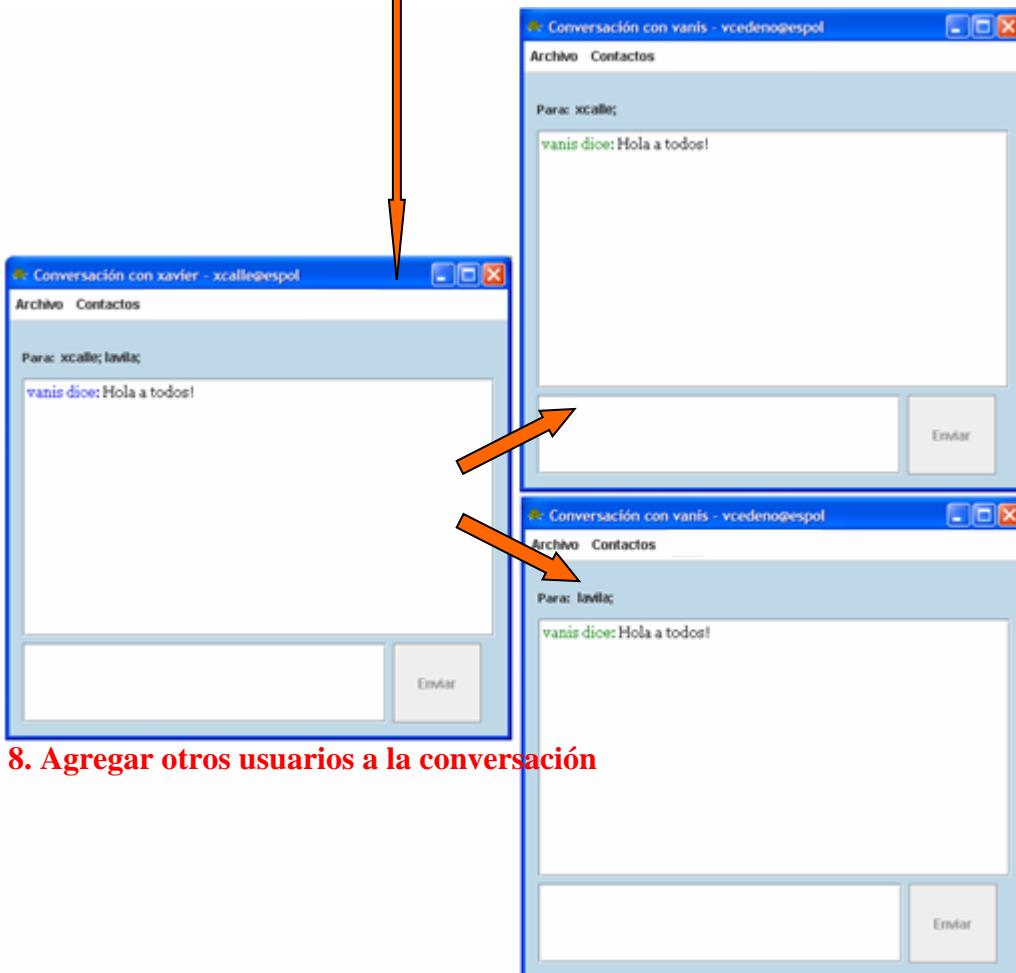




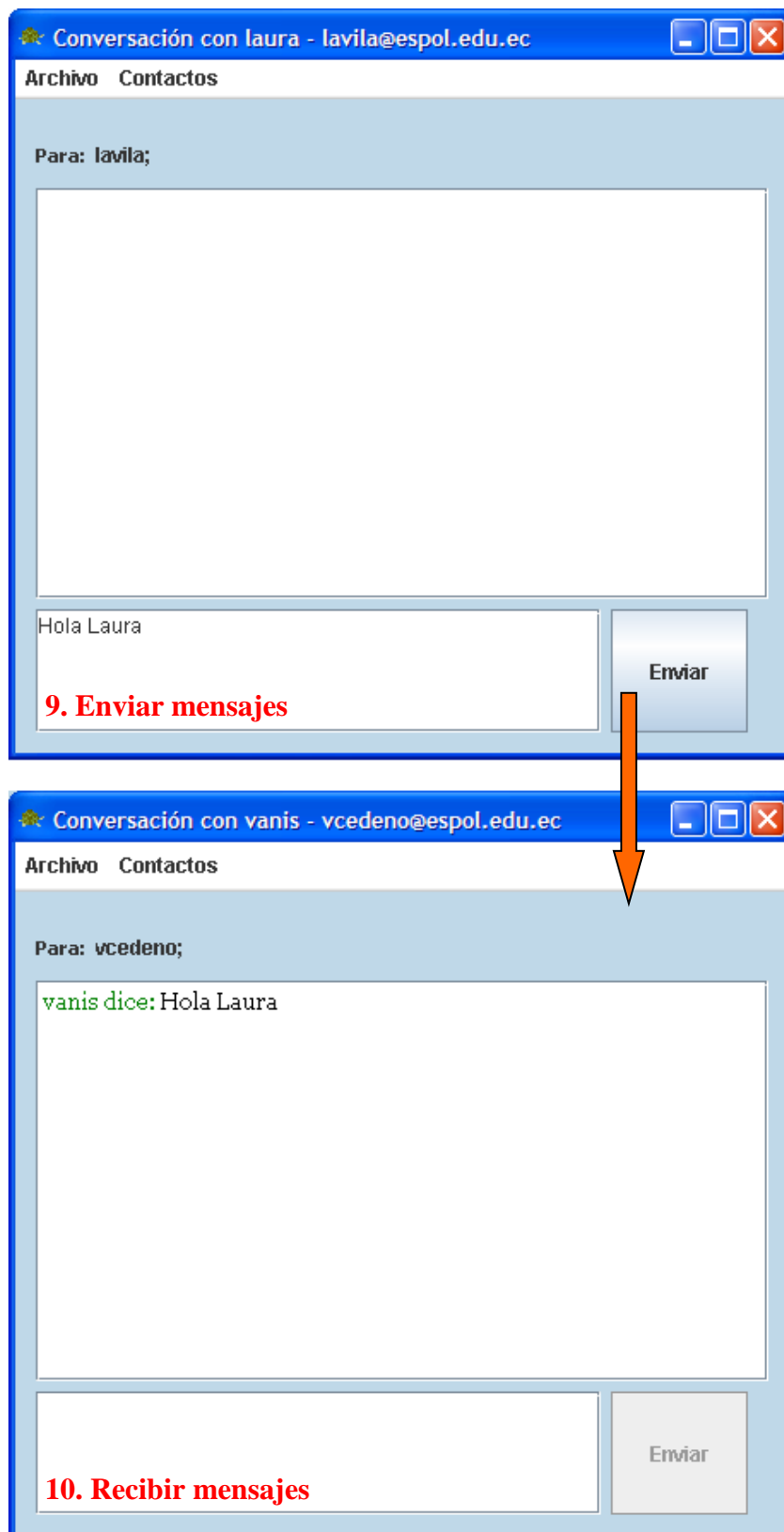


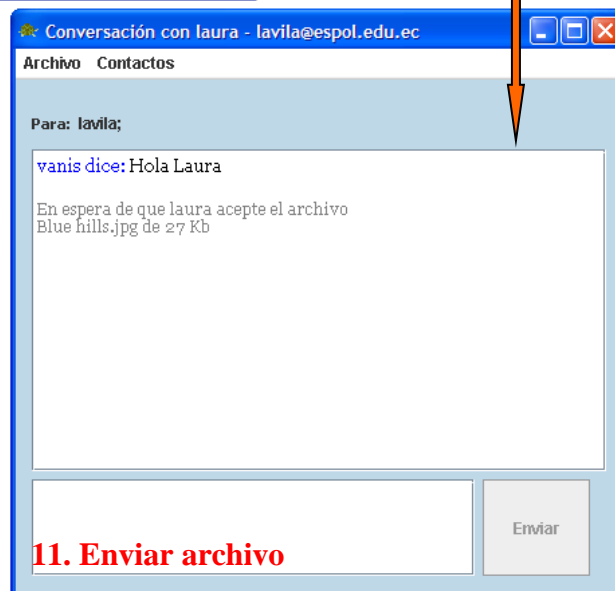
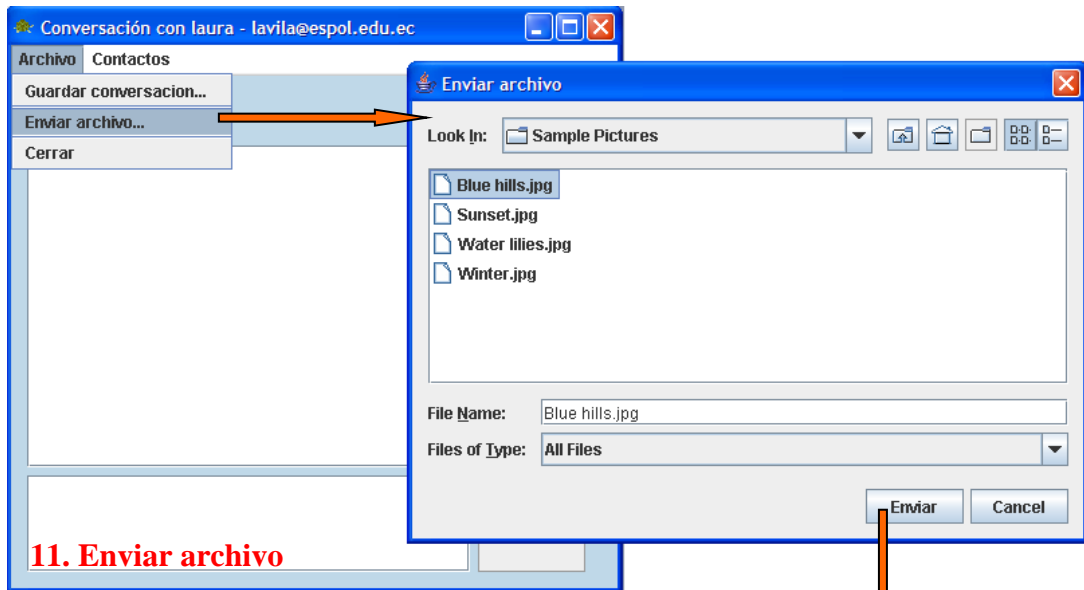


8. Agregar otros usuarios a la conversación

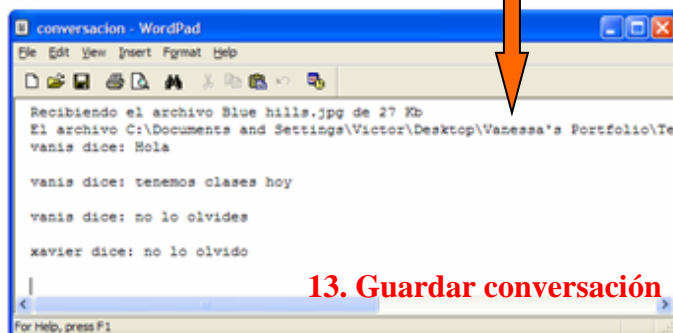
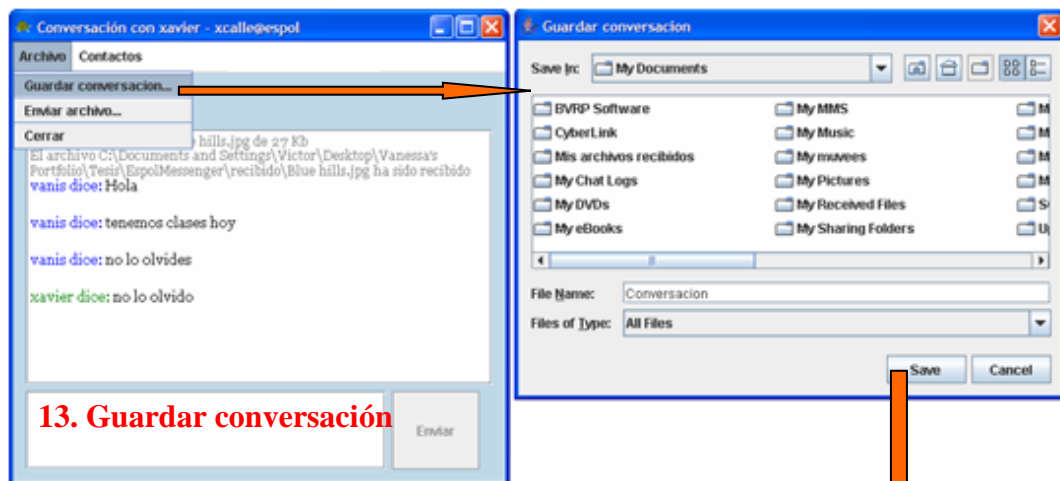


8. Agregar otros usuarios a la conversación









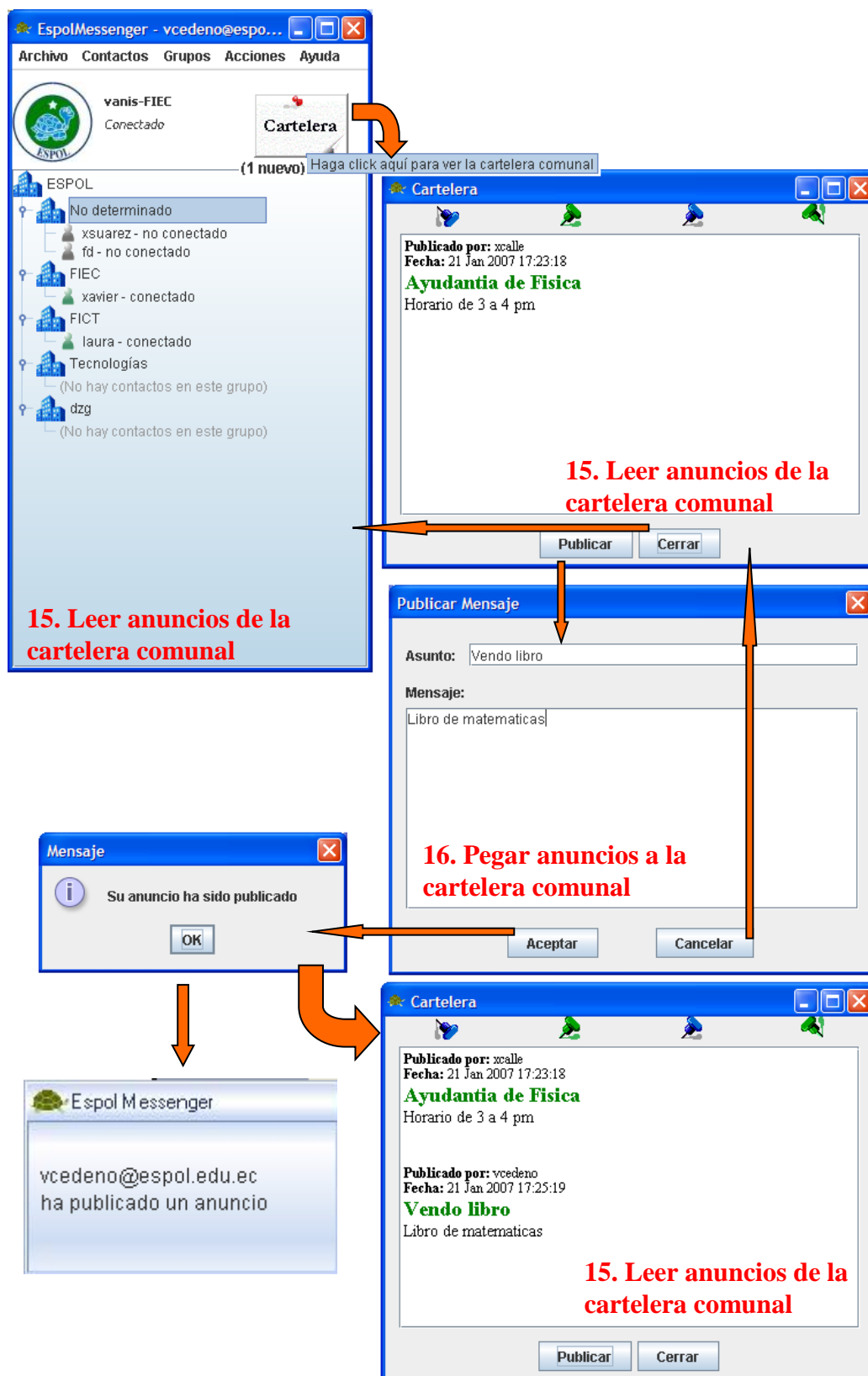
The image shows a sequence of three screenshots from the EspolMessenger application, illustrating how to change a user's status.

Top Screenshot: The main application window is open, showing a contact list on the left. The 'Estado' (Status) menu is open, and the 'Personalizado ...' (Customized ...) option is selected. An orange arrow points from this option to the 'Estado personalizado' dialog box.

Middle Screenshot: The 'Estado personalizado' dialog box is shown. It has a text input field containing 'Estudiando' and two buttons: 'Aceptar' (Accept) and 'Cancelar' (Cancel). An orange arrow points from the 'Aceptar' button back to the main application window.

Bottom Screenshot: The main application window is shown again, but now the contact 'vanis FIEC' is highlighted. Their status is shown as 'Estudiando'. A text box below the dialog box contains the text: "Puede escribir el estado que usted desea, por ejemplo, 'Durmiendo'".

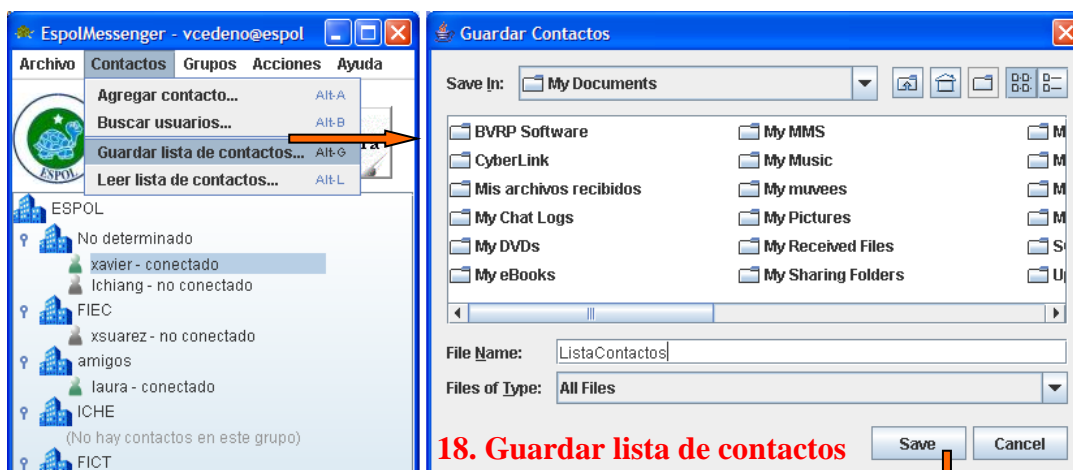
Text Labels: The text "14. Cambiar estado" is written in red at the bottom left of the first screenshot and at the bottom right of the third screenshot.



The image shows two windows from the EspoMessenger application. The left window, titled 'EspoMessenger - vcedeno@espol', has a menu with 'Buscar usuarios...' selected. An orange arrow points from this menu item to the 'Buscar' button in the right window. The right window, titled 'Búsqueda de usuarios', contains a search form with a 'Usuario:' field, a '@espol.edu.ec' domain, and 'Buscar' and 'Detener' buttons. Below the buttons is a table with the following data:

#		Usuario	Nick	Facultad
1	<input type="checkbox"/>	xcalte	xavier	FICT
2	<input type="checkbox"/>	lavila	laura	ICHE

At the bottom of the search window are 'Agregar' and 'Cerrar' buttons. A red text label '17. Buscar usuarios conectados' is positioned at the bottom of the search window. A large orange arrow points from this label to the bottom of the main messenger window, which also has a red text label '17. Buscar usuarios conectados' at its bottom.

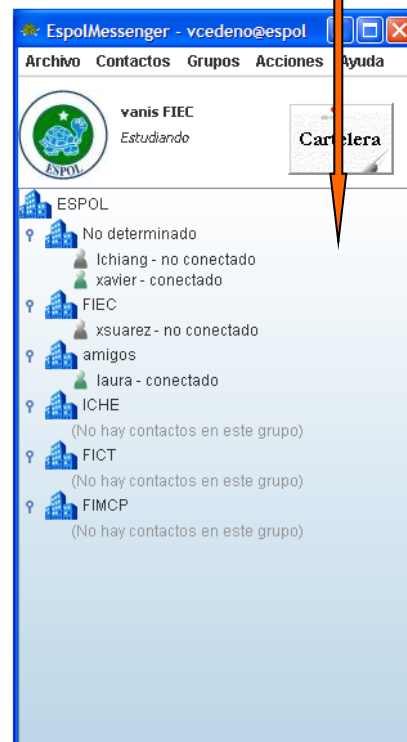
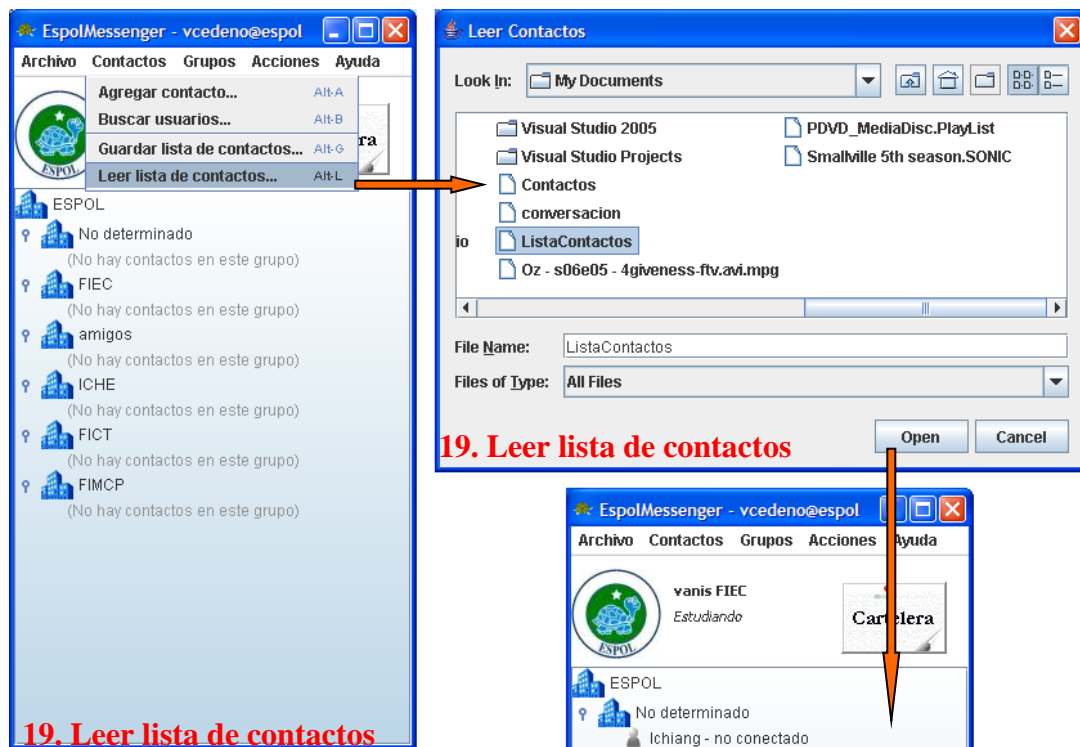


18. Guardar lista de contactos

18. Guardar lista de contactos



18. Guardar lista de contactos



The image shows two windows from the EspolMessenger application. On the left is the main application window, titled 'EspolMessenger - vcedeno@espo...'. It has a menu bar with 'Archivo', 'Contactos', 'Grupos', 'Acciones', and 'Ayuda'. Below the menu bar is a toolbar with buttons for 'Estado', 'Datos' (Alt-D), 'Opciones...' (Alt-P), 'Cerrar sesión' (Alt-C), and 'Salir' (Alt-S). The main area displays a contact list with groups like 'No determinado', 'FIEC', 'FICT', 'Tecnologías', and 'dzg'. An orange arrow points from the 'Opciones...' button to the configuration dialog on the right.

The configuration dialog, titled 'Opciones de Configuración', has a blue border and contains the following fields and options:

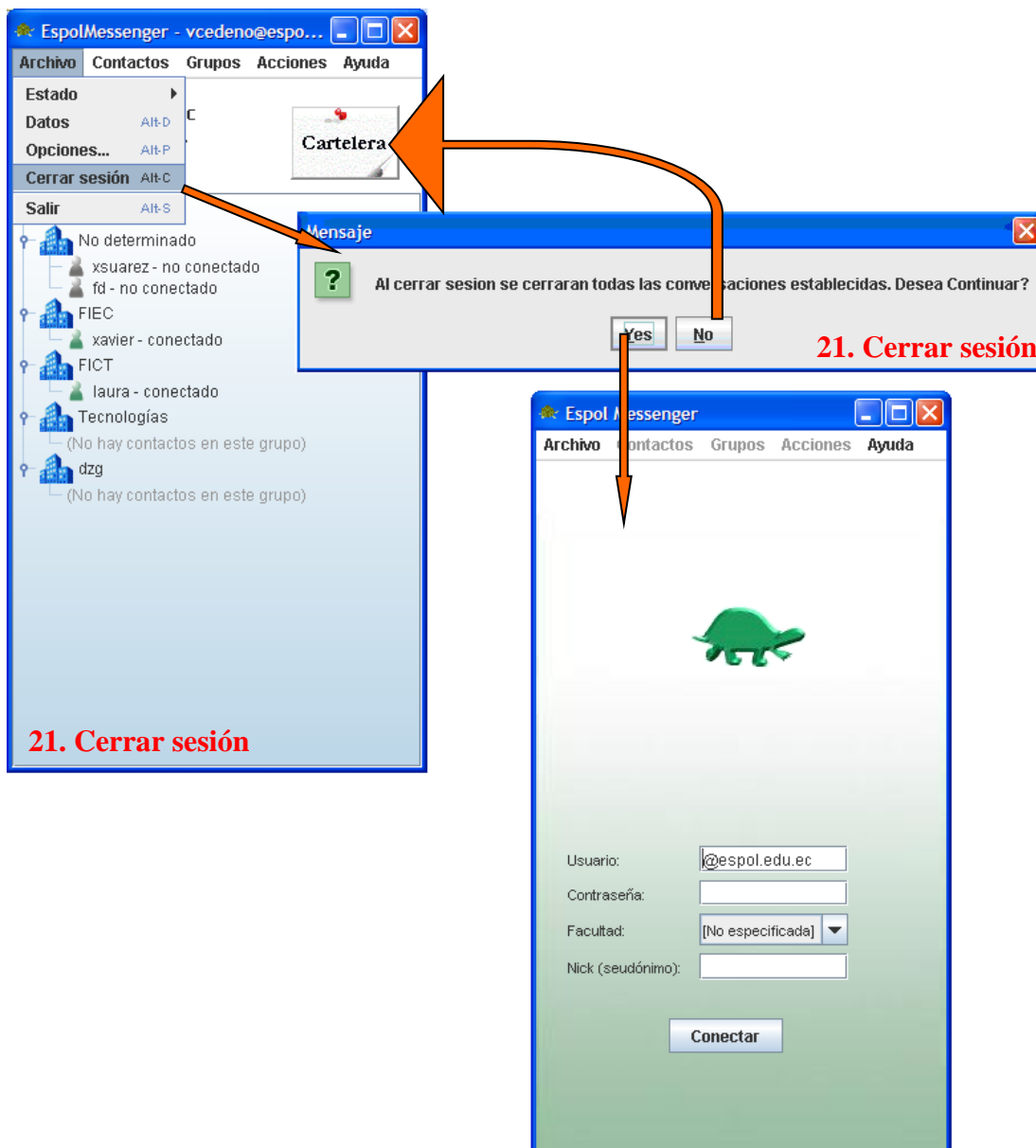
- Conexión** section:
 - Servidor Proxy: [input field]
 - Puerto: [input field]
- Usuario:** [input field]
- Contraseña:** [input field]
- Opciones de JXTA** section:
 - Deseo contribuir con la localización de usuarios y sus recursos [info icon]
- Buttons: 'Aceptar' and 'Cancelar'.

An orange arrow points from the info icon to a text box below the dialog.

20. Cambiar opciones de configuración

Seleccione esta opción si va a correr la aplicación en una pc que va a estar en línea durante horas y va a consumir pocos recursos del sistema. De esta manera su pc contribuirá con la localización de usuarios y sus recursos mejorando así el desempeño de la red EspolMessenger.

20. Cambiar opciones de configuración



BIBLIOGRAFÍA

1. Sun Microsystems. *JXTA v2.3.x: Java™ Programmer's Guide* [en línea]. V2.3. [Estados Unidos]: Abril 7, 2005 [ref. de 18 de mayo de 2006], pp. 6-29. Disponible en Web: http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf
2. Wikipedia La Enciclopedia Libre. *Jabber*. Artículo [en línea]. Abril 4, 2006 [ref. de 12 de Abril de 2006]. Disponible en Web: <http://es.wikipedia.org/wiki/Jabber>
3. Myjxta: Home. *Project Home*, [ref. de 20 de marzo de 2006]. Disponible en Web: <http://myjxta.jxta.org>
4. Instant J.I.M Messenger: Home, *J.I.M.* [ref. de 20 de marzo de 2006]. Disponible en Web: <http://jxtaim.sourceforge.net/index.html>
5. Wilson, Brendon. *JXTA* [en línea]. [Estados Unidos]: [ref. de 5 de agosto de 2006]. Chapter 3. Introducing JXTA P2P Solutions, pp. 47. Disponible en Web: <http://www.brendonwilson.com/projects/jxta-book/>
6. Santamaría Cabrera, Pablo. Santos López, Miguel. *Aplicaciones P2P* [en línea]. Universidad Complutense de Madrid [Madrid, España]: [ref. de 25 de junio de 2006], Diapositivas. 9-17. Disponible en Web: <http://asds.dacya.ucm.es/rafa/redes-itig/P2P.ppt>.

7. Wikipedia La Enciclopedia Libre. *Napster*. Artículo [en línea]. Octubre 3, 2006 [ref. de 4 de octubre de 2006]. Disponible en Web: <http://en.wikipedia.org/wiki/Napster>
8. Wikipedia La Enciclopedia Libre. *Gnutella*. Artículo [en línea]. Septiembre 26, 2006 [ref. de 4 de octubre de 2006]. Disponible en Web: <http://en.wikipedia.org/wiki/Gnutella>
9. Wikipedia La Enciclopedia Libre. *Freenet*. Artículo [en línea]. Septiembre 29, 2006 [ref. de 4 de octubre de 2006]. Disponible en Web: <http://en.wikipedia.org/wiki/Freenet>
10. Wikipedia La Enciclopedia Libre. *Kazaa*. Artículo [en línea]. Septiembre 29, 2006 [ref. de 4 de octubre de 2006]. Disponible en Web: <http://en.wikipedia.org/wiki/Kazaa>
11. Wikipedia La Enciclopedia Libre. *Peer-to-peer*. Artículo [en línea]. Febrero 8, 2006 [ref. de 10 de febrero de 2006]. Disponible en Web: http://es.wikipedia.org/wiki/Redes_P2P
12. Abad, Cristina. *Sobre el Potencial del Multicast en Capa de Aplicación* [en línea]. [Ecuador]: Octubre, 2005 [ref. de 24 de octubre de 2006]. Generalidades, pp. 2. Disponible en Web: http://www.rte.espol.edu.ec/archivos/Revista_2005/66.pdf
13. Abad, Cristina. *Redes de Distribución de Contenidos (2da parte)*. Notas de las diapositivas de la materia "Sistemas Distribuidos", Escuela

- Superior Politécnica del Litoral. [Guayaquil, Ecuador]. Diciembre 2005 [ref. de 11 de febrero de 2006], Diapositivas. 10, 35-36.
14. Asociación para la Investigación de Medios de Comunicación. *Navegantes en la Red: 8ª encuesta AIMC a usuarios de Internet* [en línea]. Sersa. [Madrid, España]: Febrero 2006 [ref. de 25 de junio de 2006], pp. 47. Disponible en Web: <http://download.aimc.es/aimc/03internet/macro2005.pdf>
 15. Wikipedia La Enciclopedia Libre. *Historia de las aplicaciones P2P*. Artículo [en línea]. Marzo 1, 2006 [ref. de 10 de febrero de 2006]. Disponible en Web: http://es.wikipedia.org/wiki/Historia_de_las_aplicaciones_P2P
 16. Wikipedia La Enciclopedia Libre. *IRC*. Artículo [en línea]. Octubre 3, 2006 [ref. de 4 de octubre de 2006]. Disponible en Web: <http://es.wikipedia.org/wiki/IRC>
 17. Wikipedia La Enciclopedia Libre. *Audiogalaxy*. Artículo [en línea]. Mayo 19, 2006 [ref. de 4 de octubre de 2006]. Disponible en Web: <http://en.wikipedia.org/wiki/Audiogalaxy>
 18. RIAA. *RIAA* [en línea]. [Estados Unidos]: 2003 [ref. de 4 de octubre de 2006]. Disponible en Web: <http://www.riaa.com/about/default.asp>
 19. Wikipedia La Enciclopedia Libre. *Grokster*. Artículo [en línea]. Septiembre 6, 2006 [ref. de 4 de octubre de 2006]. Disponible en Web: <http://en.wikipedia.org/wiki/Grokster>

20. Wikipedia La Enciclopedia Libre. *Morpheus*. Artículo [en línea]. Septiembre 27, 2006 [ref. de 4 de octubre de 2006]. Disponible en Web: http://en.wikipedia.org/wiki/Morpheus_%28computer_program%29
21. BitTorrent. *Company Overview*. Artículo [en línea]. [ref. de 4 de octubre de 2006]. Disponible en Web: <http://www.bittorrent.com/companyoverview.html>
22. Lphant. *lphant*. Artículo [en línea]. [ref. de 4 de octubre de 2006]. Disponible en Web: <http://www.lphant.com>
23. Astroseti.org. *SETI@Home* [en línea]. 2003 [ref. de 4 de octubre de 2006]. Disponible en Web: <http://seti.astroseti.org/setiathome/que.php>
24. Sun microsystems. *Compañía* [en línea]. [España]: 2005 [ref. de 4 de octubre de 2006]. Disponible en Web: <http://es.sun.com/infospain/compania/>
25. JXTA Get connected. *Project JXTA*, [ref. de 15 de marzo de 2006]. Disponible en Web: <http://www.jxta.org>
26. Sun Microsystems. *JXTA v2.3.x: Java™ Programmer's Guide*. Imagen, *Peers on the Expanded Web* [en línea]. V2.3. [Estados Unidos]: Abril 7, 2005 [ref. de 17 de mayo de 2006]. Chapter 2. JXTA™ Architecture, pp. 9. Disponible en Web: http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf
27. Sun Microsystems. *JXTA v2.3.x: Java™ Programmer's Guide*. Imagen, *Pipes* [en línea]. V2.3. [Estados Unidos]: Abril 7, 2005 [ref. de 17 de

- mayo de 2006]. Chapter 3. JXTA Concepts, pp. 16. Disponible en Web: http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf
28. Sun Microsystems. *JXTA v2.3.x: Java™ Programmer's Guide. Imagen, Protocols* [en línea]. V2.3. [Estados Unidos]: Abril 7, 2005 [ref. de 22 de junio de 2006]. Chapter 5. JXTA Protocols, pp. 26. Disponible en Web: http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf
29. NetBeans.Org. *IDE* [en línea]. Junio, 2000 [ref. de 2 de Octubre de 2006]. Disponible en Web: <http://www.netbeans.org/products/ide>
30. XML.com. *A Web Services Primer*. Artículo [en línea]. Abril 4, 2001 [ref. de 4 de septiembre de 2006]. Disponible en Web: <http://webservices.xml.com/pub/a/ws/2001/04/04/webservices/>
31. XML.com. *JXTA v2.3.x: A Web Services Primer. Imagen, Generic Web Service Architecture* [en línea]. [Estados Unidos]: Abril 4, 2001 [ref. de 4 de septiembre de 2006]. Disponible en Web: <http://webservices.xml.com/pub/a/ws/2001/04/04/webservices/>
32. Brookshier, Daniel. *The Socket API in JXTA 2.0*. Artículo [en línea]. Abril, 2003 [ref. de 15 de septiembre de 2006]. Disponible en Web: <http://java.sun.com/developer/technicalArticles/Networking/jxta2.0/>
33. Satchell, Stephen. *Adapting TSB 38 to TCP/IP Transport (Nagel Algorithm)*. Artículo [en línea]. Octubre 24, 1999 [ref. de 12 de noviembre de 2006]. Disponible en Web: <http://www.fluent-access.com/wtpapers/ip-test.html>

34. Sun Developer Network. *The Socket API in JXTA 2.0*. Imagen, *Figure 1: User 1 connects to User2* [en línea]. Abril, 2003 [ref. de 15 de septiembre de 2006]. Disponible en Web: <http://java.sun.com/developer/technicalArticles/Networking/jxta2.0/>
35. Y.-H. Chu, S. G. Rao, S. Seshan, H. Zhang. *A case for end system multicast*, IEEE J-SAC, 20(8), Octubre 2002 [ref. de 10 de diciembre de 2006], pp. 1456-1471.
36. Myjxta: Home. *JXTA Benchmarks and test Repository*, [ref. de 15 de Septiembre de 2006]. Disponible en Web: <http://bench.jxta.org>
37. Baeza, Ricardo. Rivera, Cuauhtémoc. *Ubicuidad y Usabilidad en la Web*. Artículo [en línea]. Diciembre, 2002 [ref. de 17 de septiembre de 2006]. Disponible en Web: <http://www.dcc.uchile.cl/~rbaeza/inf/usabilidad.html>