**RETO03**

**Servicio de Presencia**

**“Chat entre Usuarios”**

**Tópicos Especiales en Telemática**

**Alumnos:**

**Esteban Arango Medina**

**Sebastián Duque Jaramillo**

**Daniel J. Duque Tirado**

**Profesor:**

**Edwin Montoya Múnera**

**Fecha:**

**17 Abril de 2012**

**UNIVERSIDAD EAFIT**

**MEDELLIN**

**2012**

[Análisis y Diseño 4](#_Toc322465327)

[1.1. Requisitos 4](#_Toc322465328)

[1.1.1. Funcionales 4](#_Toc322465329)

[1.1.2. No Funcionales 4](#_Toc322465333)

[1.2. Entidades 5](#_Toc322465338)

[1.3. Interacción simétrica y sincrónica 7](#_Toc322465342)

[1.4. Modelo 7](#_Toc322465345)

[1.5. Arquitectura 7](#_Toc322465347)

[1.6. Middleware sistema distribuido 7](#_Toc322465349)

[1.6.1. Definición y Especificaciones del Entorno 8](#_Toc322465352)

[1.6.2. Suposiciones del Entorno 8](#_Toc322465353)

[1.6.3. Vocabulario de Mensajes 9](#_Toc322465354)

[1.6.4. Codificación de Mensajes 9](#_Toc322465355)

[1.6.5. Reglas de Procedimiento 11](#_Toc322465356)

[1.7. Detalles de diseño 13](#_Toc322465357)

[1.8. Manejo de sesión y estado 14](#_Toc322465358)

[1.9. Modelo de manejo de fallos 14](#_Toc322465359)

[1.10. Modelo de seguridad 14](#_Toc322465362)

[1.11. Niveles de transparencia 15](#_Toc322465364)

[1.11.1. Transparencia de acceso: 15](#_Toc322465365)

[1.11.2. Transparencia de ubicación: 15](#_Toc322465366)

[1.11.3. Transparencia frente a fallos 15](#_Toc322465367)

[1.11.4. Transparencia al escalado: 16](#_Toc322465368)

[1.12. Multiusuario 16](#_Toc322465369)

[1.13. Criterios o atributos de calidad 16](#_Toc322465370)

[1.13.1. Concurrencia: 16](#_Toc322465371)

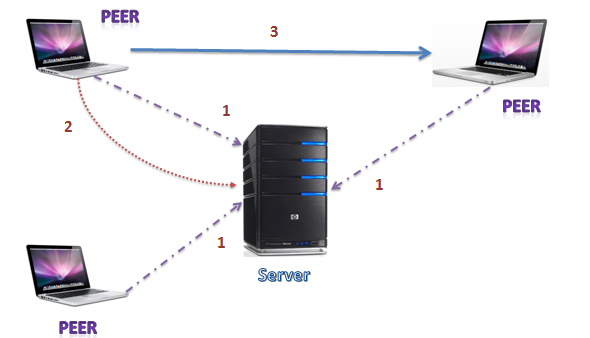
[1.13.2. Extensibilidad: 16](#_Toc322465372)

[1.13.3. Escalabilidad: 17](#_Toc322465373)

# Análisis y Diseño

* 1. Requisitos
     1. Funcionales
* Los usuarios pueden conocer el estado de otros usuarios de la aplicación, estos son: *online*, *busy*, *offline*.
* Los usuarios se comunicarán mediante un chat presencial, con envío de mensajes. Ellos podrán comunicarse en tiempo real con cualquiera de los usuarios que se encuentran conectados y disponibles (una conversación por instante de tiempo).
* Los usuarios podrán enviar mensajes a contactos *OFFLINE*, que son guardados para su posterior despliegue al inicio de sesión.
  + 1. No Funcionales
* El sistema debe permite múltiples usuarios.
* El sistema debe permitir múltiples conversaciones pero sólo una por usuario en un instante particular.
* La aplicación podrá utilizarse en diferentes plataformas, (Win32, Linux, Mac).
* Los usuarios podrán ingresar a la aplicación desde múltiples máquinas, con conocimiento de la dirección IP y puerto del servidor.
  1. Entidades

Entidades: Se identifica una entidad Servidor y entidad PEER, estas se conectan entre sí.



Identificarse Envió Información de Conexión Chat

otro usuario.

Servidor: Entidad que centraliza la información para facilidad de localización de usuarios. Contiene un objeto de cada peer online con su respectiva información, cada peer se expone al servidor al conectarse al mismo. Este además también guarda los mensajes que han sido enviados a un usuario OFFLINE y los historiales de conversación.

Peer: Entidad que realiza la lógica de conexión y de envío de mensajes. Esta es la encargada de toda la interacción como tal de la aplicación.

**Flujo de Sincronización***:*

1. Un peer se conecta con el servidor y se identifica enviándole su IP y puerto, al igual que su dirección en por la cual expone un objeto (objeto chat).
2. Cuando un usuario desea hablar con otro usuario online, este le envía la petición al servidor y posteriormente se le envía la información necesaria para la conexión (URI y objeto expuesto).
3. Se realiza la conexión entre peer’s a partir de la información brindada por el servidor. Los usuarios pueden chatear libremente.
   1. Interacción simétrica y sincrónica

*Comunicación entre Peer’s*: La interacción es simétrica, los peer’s realizan peticiones entre sí, basados en la bidireccionalidad de los servicios. Un usuario envía y recibe mensajes cuando está conectado a otro peer’s, invoca y recibe invocaciones de servicios.

*Comunicación entre Peer-Servidor:* Para la comunicación servidor-usuario la interacción es asimétrica, en donde el usuario realiza peticiones al servidor, pero los servidores no invocan nada en los clientes, sólo responde.

*Interacción sincrónica:* En todas las posibles comunicaciones y peticiones, la interacción es sincrónica, incluso para cuando los usuarios se encuentran offline y un mensaje es enviado a ellos. En este caso la conexión con el otro peer nunca se materializa, si no que los mensajes son guardados en el servidor de manera sincrónica, para después ser desplegados de la misma manera.

* 1. Modelo

La aplicación tiende a un modelo Peer-to-Peer puro, en donde los clientes se conectan entre sí y se crean tuberías de comunicación para mensajes privados.

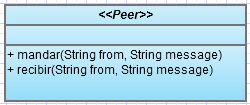
* 1. Arquitectura

La arquitectura es Peer-to-Peer híbrida server based, en donde se agrega un servidor que centraliza información para facilidad y efectividad de implementación. En este se guardan los datos de cada usuario (peer) que ingresa a la aplicación, además de un objeto del mismo.

* 1. Middleware sistema distribuido

Entre los peer’s y el servidor se utilizan conexiones TCP/IP Sockets, y para la interacción entre peer’s el middleware seleccionado es de tipo invocación remota, conocido como Distributed Ruby (DRb), este permite comunicar programas en la misma máquina o sobre una red. Este utiliza invocación de métodos para pasar comandos y data entre procesos. Las bibliotecas de DRb proveen una forma de compartir objetos sobre la red, permitiendo a los clientes de Ruby facilidad de conexión.

Definición de contrato para la invocación remota (Interfaz Peer): Cada peer es un ClientChat.

****

**Interfaz Peer**:

+ mandar(from, message):Este método invoca el metodo recibir (from, message) del objeto chat del otro participante de la conversación, es decir, del otro ClientChat que está en conexión directa.

+ recibir(from, message):Este método imprime en consola el mensaje que ha sido enviado por el ClientChat con el cual se está en conexión directa.

Se plantea un protocolo de comunicación entre los peer’s y el servidor:

* + 1. Definición y Especificaciones del Entorno

Se desea implementar un protocolo que permita a un conjunto de clientes comunicarse con un servidor, para identificación y funcionalidad de envío de mensajes.

Características:

1. Orientado a la conexión.
2. Asimétrico.
3. Sincrónico.
4. No sesión, no estado.
5. Sin seguridad (no autenticación ni encriptación).
   * 1. Suposiciones del Entorno

El protocolo se soporta en un protocolo confiable, específicamente sobre TCP, con la implementación de la API de Sockets de Ruby. Se realiza una entrega acertada y validada.

* + 1. Vocabulario de Mensajes

Se agrupa el vocabulario de mensajes para peer-servidor: {peer\_request, server\_response}

* + 1. Codificación de Mensajes

*Peer\_request & server\_response*

Una conexión TCP a través de un socket comienza la comunicación, el cliente se identifica con el exponiendo un objeto y su información, servidor envía una respuesta preguntando por el nickname del usuario, y después de que el cliente ingrese su nickname, la interacción y las peticiones se soportarán en la siguiente expresión regular:

{(?(?i)LIST USERS|CHAT|QUIT CONVERSATION|QUIT) ?(?\(.{1,}\))?}

El contenido que el cliente envía contiene un match (aceptación) de esta expresión.

Posibilidades:

1. LIST USERS: Este mensaje indica al servidor que el cliente solicita el estado de los usuarios existentes en la aplicación.

Respuesta del servidor: El servidor despliega en el cliente todos los usuarios agrupados según su estado (online, busy, offline):

1. CHAT (USER): Este mensaje indica al servidor que el cliente desea enviar mensajes a otro cliente especificado. El servidor envía la URI del usuario especificado, con el fin de poder realizar la conexión peer-to-peer y comunicarse. Todo esto es transparente para el cliente que ejecuto el comando.

Respuesta del servidor: El servidor brinda la información del cliente solicitada al cliente solicitante.

En la lógica del cliente (sin participación del servidor) se genera la conexión a partir de esta información. Mensajes de información y confirmación son desplegados en ambos peer’s que están siendo conectados.

En el peer en el cual ha llegado la petición se despliega un mensaje que permite aceptar o rechazar la invitación:

“User [username] wants to chat with you? Would you like too?(Y/N)”

Y en el peer que solicita la conexión se despliega:

“Waiting for [username] responses..."

Cuando la invitación es aceptada, comienza la comunicación libre entre los peer’s. El mensaje siguiente es desplegado:

"Your now connected with [username]."

Otros mensajes que pueden ser desplegados en el usuario solicitante son:

* "User [username] does not want to chat with you."
* "User [username] is busy at this moment, you may interrupt."
* "User [username] is offline at this moment. Would you like to leave a message?(Y/N)" Si el usuario escoge Y, lo siguiente que escribiráw será el mensaje.
* “User [username] does not exist!”

1. -QUIT CONVERSATION: Este mensaje cierra la conversación entre dos ClientChat, actualizando el estado en el servidor de cada peer involucrado en disponible y terminando la conexión que existía entre estos.

Respuesta del servidor:

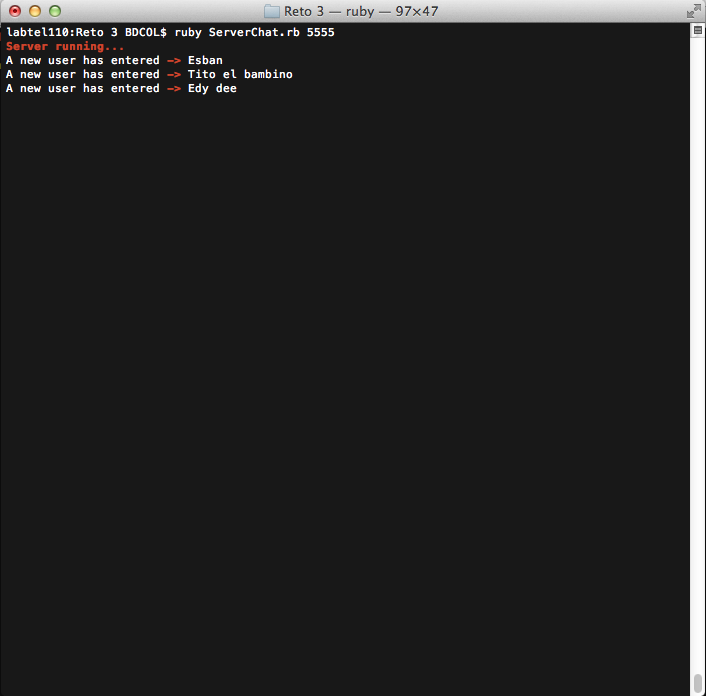
"You have just left your last conversation."

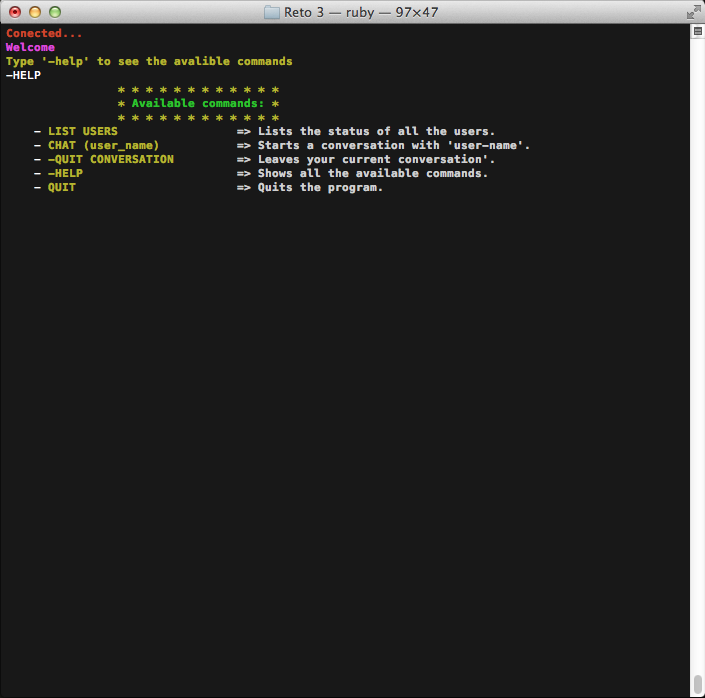
1. QUIT: Este mensaje desconecta el cliente de la aplicación, detiene la ejecución. Debe hacerse por fuera de una conversación, primero se deberá cerrar la conversación activa, si esta existe.

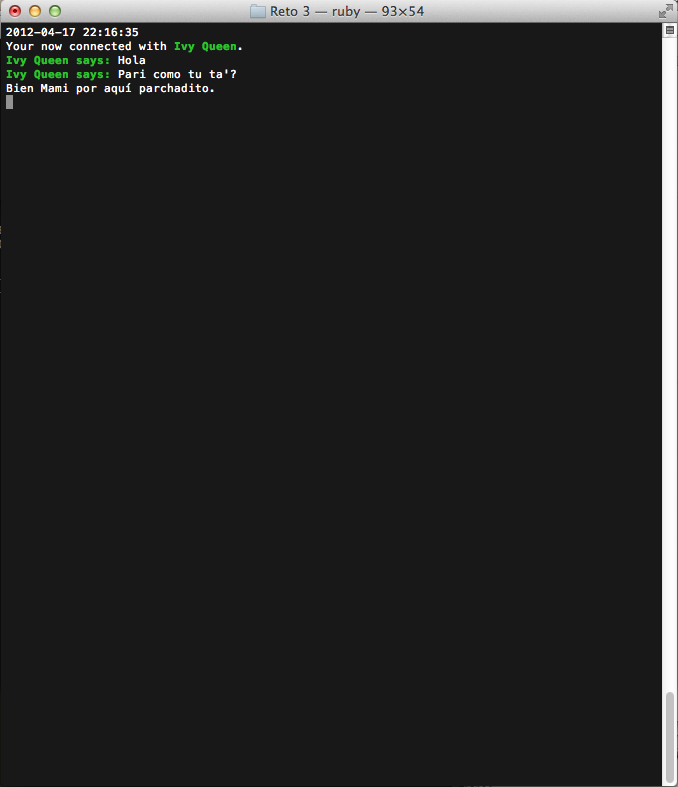
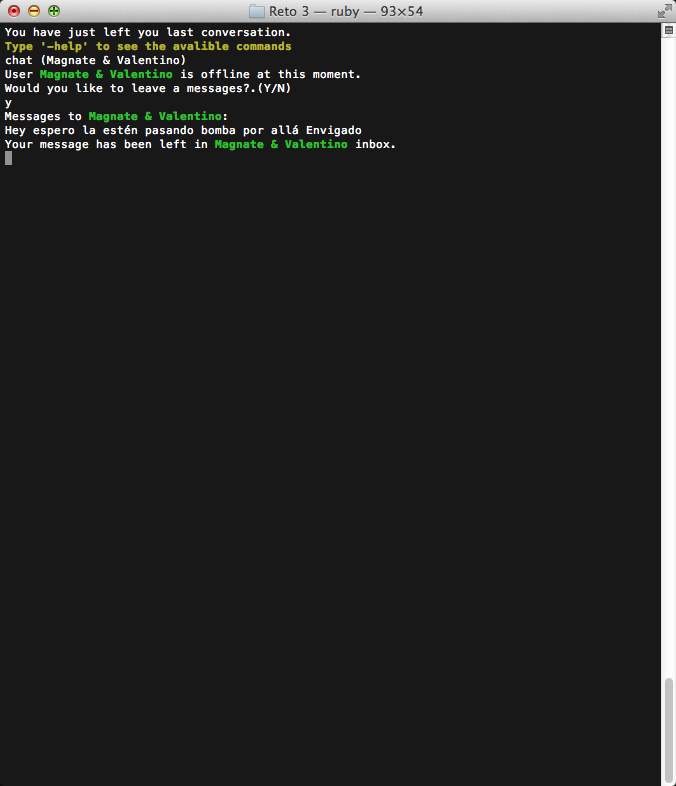
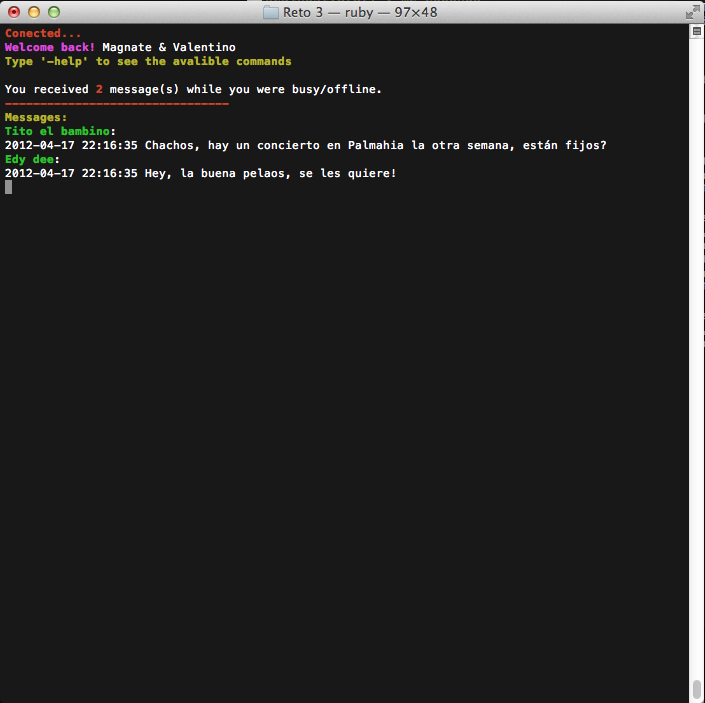
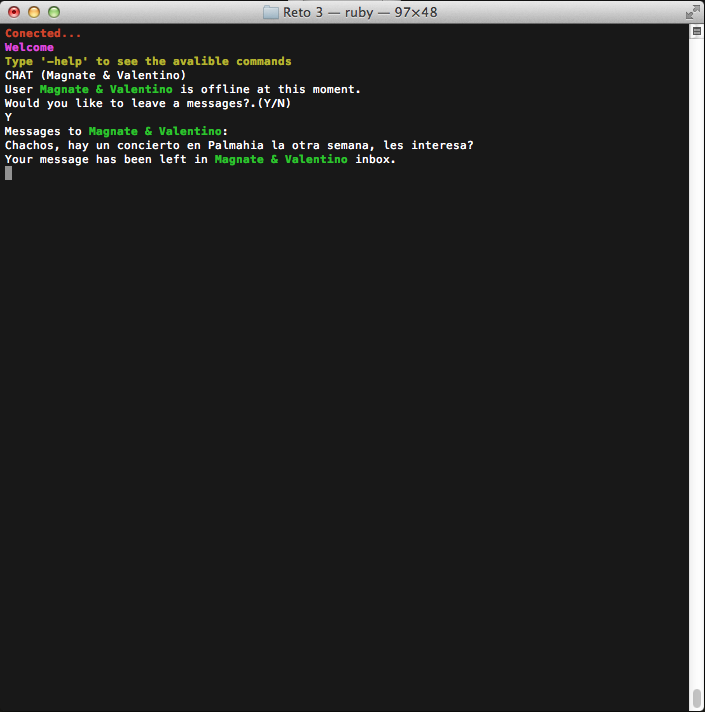
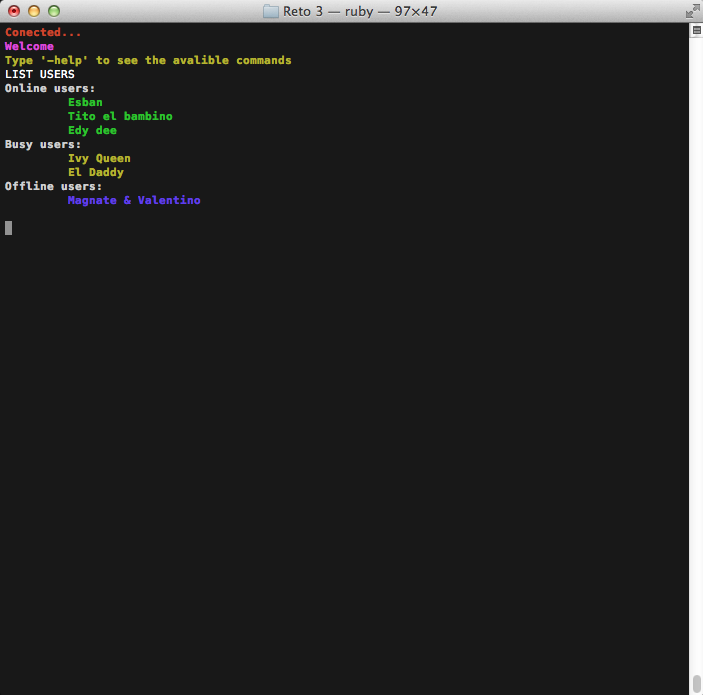
Respuesta del servidor:

"Good Bye! ☺."

Ejemplos de ejecución:

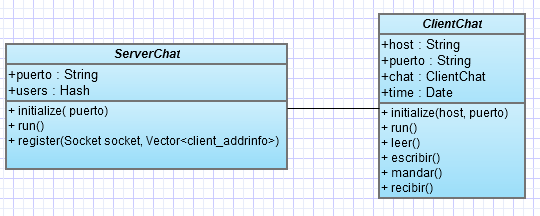
**

**

**

* + 1. Reglas de Procedimiento

Diagrama de clases de alto nivel:

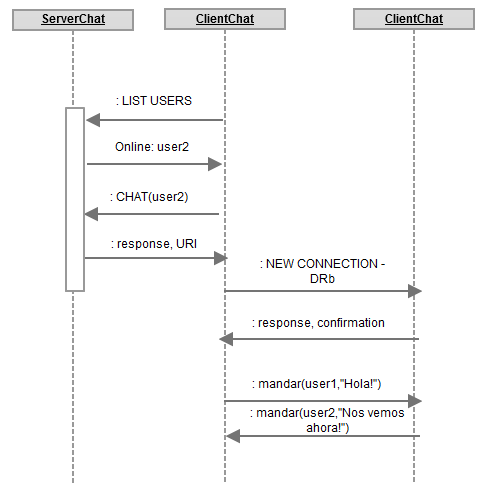


M

1 …

(Otras funcionalidades están presentes en los módulos designModules.rb y mainModules.rb)

Diagrama de secuencias: ejemplo sencillo del flujo de la aplicación con 1 cliente y el servidor.



* 1. Detalles de diseño

**Detalles del diseño**:

* ¿Qué ocurre si uno de los peer’s pierde la conexión?

La conversación actual en la que se encuentra será cerrada y el otro usuario que se encontraba en la conexión quedará por fuera de cualquier conversación pero en la aplicación. El estado de ambos cambiará, aquél que perdió la conexión quedará OFFLINE, y su similar que áun se encuentra conectado, pasará de BUSY a ONLINE.

* ¿Qué ocurre si un cliente no desea conversar con el ClientChat (cliente) que le solicita?

Un mensaje de confirmación siempre será desplegado, si el ClientChat solicitado no desea intercambiar mensajes con el ClientChat solicitante deberá responder ‘N’. De esta manera, no tendrá conversación activa y se mantendrá DISPONIBLE en la aplicación.

* Cada ClientChat (usuario, peer) podrá tener una sola conversación en un instante de tiempo.
* Cuando un ClientChat inicia una conversación su estado pasará a BUSY y estará sin disponibilidad para otros ClientChat’s
  1. Manejo de sesión y estado

No se tiene un manejo de estado, pero si existe sesión en la aplicación, implementada en los estados de los usuarios, en modo de presencia. En cada momento de la aplicación por de una petición de un usuario se debe conocer el estado del mismo en la aplicación, y la respuesta depende de ello. Las secuencias de invocación y envío de mensajes modifican el estado del usuario.

* 1. Modelo de manejo de fallos

En esta arquitectura Peer-to-Peer server based se encuentra una alta tolerancia a fallos. Si el servidor con la información de localización falla, se pierden los estados y la posibilidad de identificar clientes. Si un peer falla y pierde su conexión el mismo pasará a un estado OFFLINE, sin afectar el funcionamiento de la aplicación. Así mismo si se encuentra en una conversación, el peer destinatario de sus mensajes será informado y su conversación se cerrará controladamente.

Así mientras el servidor se mantenga corriendo, los clientes podrán fallas y la aplicación no se verá fuertemente afectada, podrá seguir prestando los servicios y características.

* 1. Modelo de seguridad

La seguridad e integridad de los datos en la aplicación se ve estructurada en que las conversaciones y el chat se mantiene entre dos peer’s ClientChat que voluntariamente han escogido compartir mensajes. Estos mensajes no serán enviados al servidor, serán privados, teniendo conversaciones seguras y sin transformación de datos. No existe autenticación de usuarios. No existen permisos ni elevación y todos los usuarios tienen las mismas prioridades. Cuando un ClientChat expone su información está siendo vulnerable a tipos de conexión con intenciones maliciosas, es su deber constatarse de la conexión proveniente.

* 1. Niveles de transparencia

La transparencia para el usuario se interpreta en la forma como se conecta a chatear con otro usuario. Su único requisito es conocer el servidor y enviarle una petición de conexión con otro usuario, inmediatamente el servidor entenderá la solicitud y brindará la información necesaria a la lógica del ClientChat para que este logre conectarse directamente con su destino. Para el usuario todo esto ocurre oculto, de una manera transparente, no deberá conocer direcciones IP y puerto de los usuarios con los cuales desea intercambiar mensajes.

* + 1. Transparencia de acceso:En este servicio los clientes (peer’s) acceden a un servidor conociendo su IP y a partir de ello pueden acceder a cualquier otro Client que se encuentra conectado al servidor. Así para el usuario los datos están centralizados y podrá accederlos mediante la comunicación con el servidor por medio del protocolo descrito previamente.
    2. Transparencia de ubicación:Para un cliente específico que desea acceder a la aplicación, los conocimientos previos deben ser sólo la información del servidor, la ubicación de clientes se podrá soportar en las herramientas que provee el middleware DRb, en donde cada nuevo cliente expone su información mediante un objeto con la funcionalidad que provee. Cada cliente solo debe preocuparse por dar su URI al servidor, y este el encargado de realizar la asociación.
    3. Transparencia frente a fallos***:***

Frente a los fallos, por su arquitectura basada en múltiples peer’s y la existencia de una entidad de comunicación que contiene lógica, tiende a ser muy eficiente mientras el servidor no caiga. Cualquier error y/o excepción que pueda ocurrir por un peer fallando se controla para mostrar mensajes amigables y flujo de eventos correctos, para así llegar a un estado seguro de nuevo. Si el servidor falla, la transparencia para el usuario es inexistente puesto que la aplicación fallará, se despliega un mensaje pero no se podrá retomar el servicio.

* + 1. Transparencia al escalado:Debido a la arquitectura P2P, el servicio puede incrementarse sin cambiar la estructura del sistema en general, nuevas funcionalidades pueden ser agregadas en cada uno de los componentes sin reemplazar la lógica de comunicación entre los mismos, soportada en el middleware y los objetos expuestos de los usuarios. El número de usuarios que puede estar en la aplicación es alto, puesto que aunque existe conexión con el servidor, el envío de mensajes se realiza entre peer’s.
  1. Multiusuario

La aplicación es multiusuario porque permite que múltiples usuarios estén enviando mensajes con un similar en un instante dado. Múltiples usuarios podrán acceder al sistema y generar petición de conversaciones y envío de mensajes.

* 1. Criterios o atributos de calidad
     1. Concurrencia: La concurrencia en la aplicación de presencia radica en permitir un alto número de  usuarios conectados simultáneamente. Se podría pensar que al haber un servidor que centraliza parte de la funcionalidad podrían presentarse inconvenientes de concurrencia, sin embargo sobre la arquitectura p2p hibrida que opera la aplicación este solo sirve como ente controlador en una parte mínima de las interacciones, por tanto no se verá saturado ni en procesamiento ni en solicitudes de entrada y salida de información permitiendo un funcionamiento adecuado. La mayor parte del flujo de datos en la aplicación será entre entidades de clientChat, permitiendo un nivel adecuado de concurrencia y escalabilidad en lo referido a usuarios.
     2. Extensibilidad: La implementación permite agregar nuevas funcionalidades sin cambios en la arquitectura. Al identificar una entidad peer y mapearla correctamente se permite que nuevos procedimientos puedan ser agregados sin cambiar las especificaciones de comunicación y relación. En cada peer(ClientChat) se podría agregar lógica con respecto a las conversaciones, como por ejemplo, brindar detalles adicionales de las misma, guardar registro de número de mensajes y usuarios, entre otras funcionalidades. Estas nuevas funciones se podrán agregar cargando la lógica de la entidad peer, pero no cambiarán la arquitectura de la solución. También podrá agregarse la posibilidad de hablar con múltiples usuarios al tiempo, generando un hilo por cada conversación.
     3. Escalabilidad: La escalabilidad de la aplicación es alta debido al hecho de que es una red peer to peer, permitiendo la adición de peer’s sin mucha carga en la red. Cada peer podrá también tener su conversación y esta no afectará en general la carga de la red, puesto que cuando se realiza una localización se obtiene la dirección IP del peer destino, y a partir de allí se realiza una conexión entre peer’s directa.