MENSAJERÍA INSTANTÁNEA PARA LA PLATAFORMA EAFIT INTERACTIVA

SANTIAGO OQUENDO SOLANO

UNIVERSIDAD EAFIT
ESCUELA DE INGENIERÍAS
DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS
MEDELLÍN
2008

MENSAJERÍA INSTANTÁNEA PARA LA PLATAFORMA EAFIT INTERACTIVA

SANTIAGO OQUENDO SOLANO

Proyecto de grado

Asesor
CAROLINA PABÓN RAMÍREZ

UNIVERSIDAD EAFIT
ESCUELA DE INGENIERÍAS
DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS
MEDELLÍN
2008

Nota de aceptación:

_____
_____
_____
_____
_____
_____

_____

Firma del presidente del jurado

_____

Firma del jurado

_____

Firma del jurado

Medellín, ___ de _____ de ____

Contenido

LISTA DE FIGURAS

LISTA DE TABLAS

INTRODUCCIÓN

Desarrollar ambientes colaborativos que sean efectivos para los espacios académicos y laborales de las personas siempre ha sido una de las prioridades de investigación en el campo del trabajo colaborativo soportado por computador (CSCW[1] – Computer Supported Cooperative Work). Aunque las inversiones económicas en sistemas para trabajos grupales efectivos han sido inmensas, los resultados no son los mejores. La razón es simple: aunque existe una gran variedad de herramientas disponibles, muchas organizaciones continúan dependiendo principalmente del email como herramienta colaborativa principal, apoyados en otros elementos secundarios. Sin embargo, la naturaleza de las comunicaciones está cambiando. Las personas se comunican de una forma más fluida y hacen parte de muchas redes sociales en vez de pertenecer solo a un grupo de trabajo o comunidad.

Es por esto que muchas organizaciones, utilizan como ambiente colaborativo la mensajería instantánea, bien sea, como complemento del correo electrónico, o como su reemplazo definitivo. Generalmente, la adopción de los sistemas de mensajería instantánea en las organizaciones, es una tarea sencilla desde el punto de vista cultural, pues las personas heredan este hábito del hogar o de la sociedad.

Lo que hace de la mensajería instantánea, una tecnología popular entre las personas (en contextos formales e informales), es que adiciona facilidad y velocidad a la comunicación, y elimina el tiempo perdido empleado en otros sistemas de comunicación como las llamadas telefónicas, los desplazamientos físicos, o los correos electrónicos.

La mensajería instantánea y la presencia, constituyen un método bastante efectivo para facilitar y agilizar el proceso colaborativo. Presencia debe

[1] CSCW: Es uno de los principales énfasis de aplicación en sistemas informáticos distribuidos, según expertos CSCW se define como “Un termino genérico donde se combina el entendimiento de las personas trabajando en grupos con las tecnologías disponibles en redes de computadoras y son su hardware, software, servicios y técnicas asociadas”. (webopedia, 2003).

entenderse como la disponibilidad inmediata de un determinado individuo para atender un incidente o proporcionar respuestas a cualquier tipo de solicitudes.

El presente trabajo de grado, pretende, incorporar dentro de la Plataforma de EAFIT Interactiva[2], un sistema de mensajería instantánea que le proporcione a los usuarios, estudiantes y profesores, un espacio colaborativo en tiempo real, y con tecnologías de última generación. Para ello, se describen los argumentos que soportan la elección de la tecnología a emplear y se detalla la estructura tecnológica de la aplicación final.

[2] El Campus Bimodal EAFIT Interactiva es una red de personas y recursos telemáticos al servicio de la enseñanza y el aprendizaje universitario, ésta nace para apoyar el proceso más importante de la universidad, que es el proceso de docencia e incluye herramientas como: Contenidos de materias, correo, foro, anuncios, bibliografia, calendario.

1. DEFINICIÓN DEL PROBLEMA

EAFIT Interactiva es una plataforma bastante robusta e importante para el desarrollo de la cooperación académica y la planeación de las actividades que componen el semestre de cada una de las materias de la Universidad para todos los estudiantes.

Hace poco tiempo se lanza a producción una nueva funcionalidad que busca agregar valor a la plataforma: Los mensajes instantáneos se desarrollan con el propósito de aumentar la cooperación académica entre las personas implicadas con el aprendizaje, es decir, alumnos y profesores.

Sin embargo para todos los que alguna vez usamos ésta mensajería de EAFIT Interactiva notamos que no cumplía con las especificaciones básicas, ni siquiera funcionales para darle el aprovechamiento necesario.

Es por esto que se pretende construir para la plataforma un sistema de mensajería instantánea ajustado a los últimos estándares tecnológicos que le permitan estar al nivel de sistemas de mensajería fuertemente posicionados como los que están integrados en aplicaciones tan prestigiosas como Gmail[3] o Facebook[4].

Sin embargo, lograr el nivel de estas aplicaciones, requiere de la solución de varios retos tecnológicos que están implícitos en las características que debe tener la plataforma de mensajería instantánea. Además, debe tenerse en cuenta que EAFIT Interactiva es un sistema crítico para las actividades académicas dentro de la Universidad, y existen exigencias en sus niveles de servicio, su arquitectura ya se encuentra fuertemente definida, y existen restricciones técnicas y de rendimiento que son políticas establecidas en el

---

[3] Llamado en otros lugares **Google Mail** (Alemania, Austria y Reino Unido) por problemas legales, es un servicio de correo electrónico con posibilidades POP3 e IMAP gratuito proporcionado por la empresa estadounidense Google que ha captado la atención de los medios de información por sus innovaciones tecnológicas y su capacidad.

[4] Es un sitio web de redes sociales. Fue creado originalmente para estudiantes de la Universidad de Harvard, pero ha sido abierto a cualquier persona que tenga una cuenta de correo electrónico. Los usuarios pueden participar en una o más redes sociales, en relación con su situación académica, su lugar de trabajo o región geográfica.

Centro de Informática de la Universidad que deben mantenerse para garantizar la calidad de las aplicaciones.

2. OBJETIVOS

2.1. Objetivos Generales

- Facilitar a los usuarios de la plataforma de EAFIT INTERACTIVA el acceso a un sistema de mensajería instantánea de última generación.

- Fortalecer las herramientas de comunicación disponibles para el cumplimiento de la misión académica.

- Ampliar los ambientes colaborativos y de cooperación académica.

2.2. Objetivos Específicos

- Acceder a tecnologías web de última generación para implementar el sistema de mensajería instantánea.

- Incursionar a la plataforma de EAFIT Interactiva en las formas de cooperación mas difundidas en los portales prestigiosos del mundo.

- Aumentar el interés de los usuarios por la plataforma de EAFIT Interactiva.

- Crear nuevos espacios de cooperación y difusión académica.

- Incrementar las opciones de comunicación Alumno-Alumno, Alumno-Profesor.

3. ALCANCE Y PRODUCTOS

3.1. La lógica del cliente

Constituye un verdadero desafió tecnológico para garantizar, que la aplicación del chat funcione en cualquiera de los navegadores de última generación. En esta parte debe incluirse todo el código necesario para la interacción con los diversos navegadores y garantizar de esta manera el éxito de la herramienta. Constituye sin duda la parte más gruesa del proyecto, pues el buen manejo de las tecnologías existentes es la clave para la construcción y la integración no intrusiva del sistema de mensajería instantánea con la plataforma de EAFIT Interactiva.

3.2. La lógica del servidor

Incluye todas las tecnologías necesarias para que el protocolo de comunicación sea claro y limpio. Desde el servidor deben manejarse los estándares para el intercambio de mensajes, los cuales deben garantizar un nivel de respuesta óptimo y un consumo mínimo de recursos.

3.3. Lógica de datos

Almacenar toda la información referente al uso del sistema de mensajería: historial de conversaciones, último acceso y demás datos que permitan generar estadísticas del uso de la herramienta además de generar valor a los usuarios.

3.4. La interfaz

La correcta elección de la interfaz que comunica la lógica del cliente con la del servidor, es otro factor fundamental para lograr un sistema de mensajería instantánea de última generación.

3.5. El documento

Se describe toda la información importante que se tuvo en cuenta en términos de tecnologías, arquitecturas y estándares para la construcción de la aplicación de mensajería instantánea.

4. JUSTIFICACIÓN

4.1. Universidad EAFIT

La incorporación de un sistema de mensajería instantánea dentro de una plataforma académica como EAFIT Interactiva, posibilita el fortalecimiento del proceso académico y amplía el conjunto de herramientas de las que disponen los actores del cuadro de aprendizaje (profesores y alumnos). Con esto, la Universidad verá un incremento positivo de su imagen al igual que el Centro de Informática, responsable directo de las aplicaciones que apoyan todos los procesos académicos.

La unidad de negocio del Centro de Informática de la Universidad EAFIT, encargada de la comercialización de los excelentes productos software de la Universidad, puede demostrar a sus clientes con la implantación de este nuevo sistema de comunicación, el esfuerzo continuo que realiza la Universidad por mejorar los sistemas que vende, siempre con el objetivo de optimizar y enriquecer los procesos de aprendizaje dentro de otras instituciones universitarias que han visto en los productos de la Universidad gran potencial.

4.2 Personal

Este proyecto constituye un nuevo enfoque, un punto de aprendizaje diferente sobre este universo informático que tanto me interesa: la Web. Dentro de la experiencia académica y laboral de los últimos años, este proyecto en particular constituye un desafío a principios y limitaciones técnicas que hasta la fecha parecían inamovibles. Gracias al proceso de investigación realizado, fue posible descubrir nuevas y excitantes formas de atacar los desafíos que trae consigo el desarrollo Web, con el fin de hacerlo cada vez mas confiable, estable y seguro, además de ofrecer a los usuarios experiencias de navegación cada vez mas novedosas.

5. MARCO DE REFERENCIA

5.1. Mensajería Instantánea

5.1.1. Historia

La historia de la mensajería instantánea inicia mucho antes que la Internet. Las instituciones académicas y los laboratorios de investigación fueron los primeros en utilizar los PC alrededor de los inicios de 1970. De allí que los programadores de la época iniciaron el desarrollo de formas de comunicación con otros usuarios a través de mensajes basados en texto. El deseo de que las personas pudieran hablar con usuarios del mismo computador o con una máquina conectada a la red local de una respectiva universidad, son las bases que fundamentaron lo que hoy es una industria fuerte y en crecimiento.

Tres aplicaciones diferentes surgieron entre los años 70-80 que servirían de base para la mensajería instantánea tal como hoy la conocemos. La primera, llamada el protocolo peer-to-peer, permitía comunicación entre dos usuarios en el mismo computador. En la medida que el concepto de redes fue evolucionando, también lo hizo el concepto peer-to-peer, y de este modo, usuarios dentro del campus, en el pueblo, o en otra ubicación conectada podían acceder a esta comunicación de dos vías, basada en mensajes de texto, sin tener que estar en el mismo computador.

En 1983, Mark Jenks, construyó “Talk”, un sistema que le permitía a los estudiantes de la preparatoria de Washington acceder a sistema de boletines de ultima generación, con salas de redes sociales que permitían enviar mensajes privados a otros usuarios. La aplicación también conocida como “Talker”, requería que los usuarios se identificaran en la red usando un nombre identificador. Rápidamente, la aplicación fue expandiéndose a través

de los Estados Unidos en negocios privados y redes escolares hasta mediados de los 90.

Similar en naturaleza, ICR (Internet Relay Chat), ayudó a la industria del periodismo a utilizar el máximo potencial que la comunicación por Internet podía tener. Creado por Jarkko Oikarinen, en agosto de 1988, ICR permitía a sus usuarios hablar en grupos conocidos como canales, enviar mensajes privados a otros usuarios y compartir archivos a través de un sistema de transferencia de datos.

En agosto de 1982, Commodore International lanzó un computador de 8 bits que revolucionaria no solo el mundo de los computadores, sino también la próxima generación de programas para la mensajería instantánea. El Commodore 64 vendió mas de 30 millones de unidades, siendo el PC mas vendido de toda la historia, ofreciéndole a los usuarios de los hogares un primitivo servicio de Internet llamado, Quantum Link. Usando un sistema basado en texto, los usuarios podían enviarse mensajes entre si vía un MODEM telefónico y el servicio Quantum Link.

En los 90, Quantum Link cambió su nombre a America Online y ayudó a guiar la nueva era de la mensajería instantánea. Mientras que ICQ, un mensajero basado en texto, se convirtió en el primer sistema masificado en 1996, la llegada de “AOL Instant Messenger” en 1997 se convirtió en un punto de referencia para la industria pues miles de jóvenes y usuarios conocedores de tecnología encontraron en el producto la oportunidad de compartir mensajes entre ellos. Yahoo lanza su propio sistema de mensajería (Yahoo Messenger) en 1998, seguido por MSN en 1999 y muchos otros durante el 2000. Google Talk fue lanzado en el 2005.

Hasta el 2000, los usuarios de la mensajería instantánea no tenían más opción que usar varias aplicaciones para acceder a sus amigos en las diferentes redes. Entonces llega Jabber para cambiar las reglas.

Jabber actúa como una única puerta de ingreso para acceder a múltiples clientes de mensajería instantánea. De este modo, los usuarios de este

protocolo gratuito pueden hablar con sus amigos de las distintas redes a través de la misma aplicación.

5.1.2. Definición

Es una comunicación en tiempo real entre dos o más personas basadas en mensajes de texto. El mensaje de texto es transportado vía los computadores conectados en una red como la Internet.

La MI[5] es una aplicación basada en el protocolo de la Internet (IP) que provee comunicación conveniente entre personas usando una variedad de dispositivos diferentes. La forma mas familiar de la actualidad, es la mensajería computador a computador usando mensajes de texto, pero la MI también funciona con dispositivos móviles, como celulares y puede incorporar voz y video.

Para lograr su objetivo, las aplicaciones de MI incluyen:

- Un mecanismo de ventana emergente para mostrar los mensajes en el momento en el que son recibidos.
- Una lista de usuarios (contactos) relativos a cada usuario.
- Un método para indicar cuando los contactos están conectados y disponibles para recibir un mensaje.
- Un método para redactar y enviar el mensaje a su destinatario.

5.1.3. Modelo de Mensajería Instantánea

5.1.3.1 Servicio de Presencia

[5] De ahora en adelante MI se referirá al término Mensajería Instantánea.

El servicio de presencia sirve para aceptar información, almacenarla y distribuirla. La información almacenada se conoce como Información de presencia. Algunos ejemplos de esta información son el estado del cliente (Conectado, Ocupado, Alejado), el nombre de usuario, información pública, entre otros. Básicamente es la información que un usuario puede conocer sobre otros usuarios.

Para entender un poco como funciona el servicio de presencia desde la perspectiva del usuario, supongamos que Juan quiere suscribirse a un servicio de presencia (Figura 1). Para nuestro ejemplo, Juan ha seleccionado el servicio de MSN, baja el programa MSN Messenger y lo instala en su computador. Cuando Juan ingresa la información de su usuario, el servicio de presencia (MSN en este caso), acepta, almacena y distribuye la información de presencia de Juan que desde ese momento se convierte en una Entidad de Presencia[6]. En ese instante, el proceso de autenticación ha cambiado su estado como suscriptor a abierto y el servicio de presencia actualiza su información usando una tupla[7] de presencia enviada a través del protocolo de presencia. El protocolo es simplemente un conjunto de reglas predefinidas que cada participante en el proceso de comunicación ha aceptado usar con el objetivo de comunicarse exitosamente. La tupla usada en estos protocolos por lo general esta dividida en tres partes. Una de ellas, es una sección de estado, una sección opcional conocida como dirección de comunicación y otra adicional donde se almacenan otros datos de presencia importantes. El servicio de presencia mantiene toda la información relevante de sus suscriptores y envía notificaciones de los cambios de estado a los suscriptores en la medida que estos ocurran.

Estando ya autenticado exitosamente en el servicio, Juan comenzará a recibir inmediatamente notificaciones del servicio de presencia. Lo que pasa internamente es que el software en el cliente ha iniciado un proceso (llamado Fetcher) pidiendo el estado actual de cada uno de los contactos que Juan tiene almacenados en su archivo de datos de suscriptor.

---

6 En inglés PRESENTITY (PRESense ENTITY)

7 En informática, una tupla es un objeto que bien puede tener datos o diversos objetos. Por ejemplo, una posible tupla para Juan seria: (nombre : “Juan Lopez”, estado : “Conectado”)

Para que Juan pueda recibir las notificaciones de cambios que se mencionaban previamente, el software inicia un proceso llamado Poller. El Poller se usa para hacer peticiones de actualización permanentes al servicio de presencia referente a la información de presencia de los contactos de Juan.

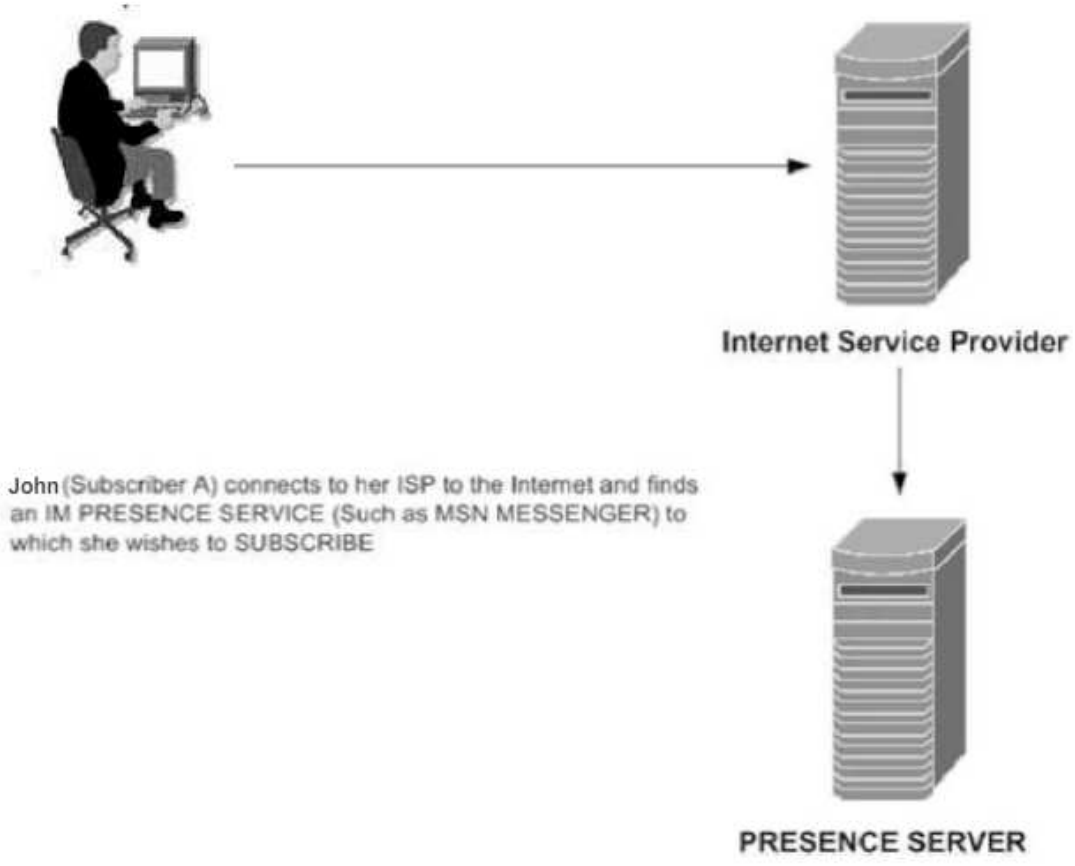


Figura 1: Un suscriptor se conecta a través de su servicio de Internet y encuentra un Servicio de Presencia de MI (como MSN) al cual quiere suscribirse. Nota: El servidor de presencia no tiene que ser un servidor dedicado a este fin. [3]

El servicio de presencia usa unas reglas de acceso para determinar de que manera los otros suscriptores verán la presencia de Juan.

5.1.3.2 Servicio de mensajería instantánea

Este servicio acepta y envía mensajes instantáneos a bandejas de entrada instantáneas. Cada suscriptor cuenta con una bandeja de entrada para recibir sus mensajes instantáneos. Cuando un mensaje es recibido, llega a la

bandeja de entrada y una notificación es mostrada inmediatamente. La notificación puede mostrar el mensaje por si sola o debe permitir al usuario hacer clic para ver el mensaje. En ocasiones, la notificación es acompañada de un pequeño sonido para captar la atención del usuario. Las respuestas son procesadas y enviadas a su destinatario remoto de la misma manera.

La comunicación instantánea colaborativa entre dos suscriptores es de hecho manejada a través del servicio de presencia.

Lo que permite que el servicio de presencia y de mensajería coexistan y trabajen sincronizadamente, es que ambos operan bajo protocolos altamente estandarizados, tal como se detalla en la siguiente sección.

5.1.4 Protocolos

5.1.4.1. XMPP[8]

XMPP es la formalización hecha por la IETF[9] a los protocolos creados por la comunidad Jabber en 1999.

XMPP es un protocolo abierto basado en XML para MI y otras aplicaciones apoyadas en presencia. Su estándar, se divide en cuatro RFC:

RFC 3920: (XMPP) – Núcleo:

Define las características principales del XMPP, un protocolo para trasmitir elementos XML con el objetivo de intercambiar información estructurada en tiempo real entre cualquier par de puntos de una red.

RFC 3921: (XMPP) – MI y presencia:

---

[8] Extensible Messaging and Presence Protocol - RFC 2778

[9] Internet Engineering Task Force

Describe las aplicaciones y extensiones a las características del núcleo de XMPP que proveen la funcionalidad básica de la MI y la presencia como esta definido en el RFC 2779[10].

RFC 3922: (XMPP) y Common Presence and Instant Messaging (CPIM):

Define la interoperabilidad entre MI y presencia.

RFC 3923: Encripción para XMPP:

Define métodos para encriptar la autenticación y algunos objetos del XMPP.

5.1.4.2 Protocolo Jabber y XMPP

Jabber y la comunidad de MI

El sustantivo Jabber, es generalmente usado en muchas cosas del mundo de la MI. No solo representa a Jabber Inc, la compañía de software comercial ubicada en Denver, Colorado, sino que representa otras áreas relacionadas a la MI. Jabber Inc. entrega una infraestructura de presencia, mensajería y XML para permitir aplicaciones en tiempo real, sistemas y servicios. Como patrocinador de Jabber Software Fundation, Jabber Inc. alimenta el espíritu de la libertad que caracteriza a la comunidad del software libre. Esta fundación sin ánimo de lucro, es la que maneja el protocolo Jabber y es conocida como jabber.org.

Jabber está basado en XMPP. XMPP es el único protocolo aprobado por la IETF para mensajería en tiempo real y presencia. La comunidad global de Jabber está creciendo a nivel global y miles de desarrolladores crean soluciones basadas en Jabber.

[10] Requisitos de la mensajeria instantanea y los protocolos de precencia. http://www.ietf.org/rfc/rfc2779.txt

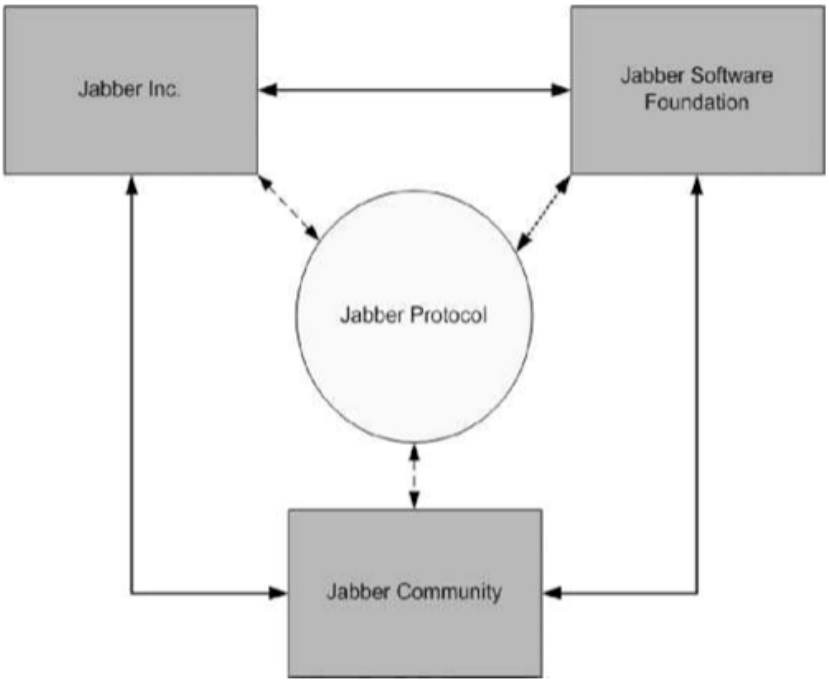


Figura 2: Ilustración del mundo de Jabber. [3]

La arquitectura de los sistemas Jabber de MI es similar a la de el sistema de mensajería mas usado y probado del planeta: e-mail. Aunque hay unas diferencias claves, no es del todo ilógico pensar en Jabber como “e-mail instantáneo”.

Para entender como funciona, pensemos que Juan y Luisa usan Jabber en sus trabajos. Luisa tiene una cuenta en un servidor Jabber, y su identificador se parece mucho a una cuenta de correo. Como Luisa es una empleada de ABC, ella registra el nombre de usuario Luisa dentro del servidor que esta instalado en ABC.com, entonces su id será luisa@abc.com. Similarmente, Juan tiene una cuenta en su empresa XYZ, y su id de Jabber es juan@xyz.net.

Una vez que Luisa entra al servidor ABC.com, puede enviar mensajes a su colega Juan de la compañía asociada XYZ. Cuando Luisa inicia el cliente de mensajería en su computador esto es lo que ocurre:

- Luisa envía un mensaje a juan@xyz.net.
- El mensaje es manejado por el servidor Jabber de ABC.com

- El servidor de ABC.com abre una conexión a XYZ.net si no hay ninguna abierta en el momento.
- Suponiendo que los administradores no han deshabilitado las comunicaciones servidor a servidor entre ABC.com y XYZ.net, el mensaje de Luisa es enrutado al servidor Jabber de XYZ.net.
- El servidor en XYZ.net observa que el mensaje esta dirigido a un usuario llamado “Juan” y lo envía al cliente Jabber de Juan en su PDA. El mensaje aparece en la pantalla de Juan.

Hay muchas cosas que observar aquí, existen varios clientes en diferentes sistemas operativos, múltiples servidores y canales de comunicación entre servidores. Jabber se encarga de todo esto. (Ver figura 3)

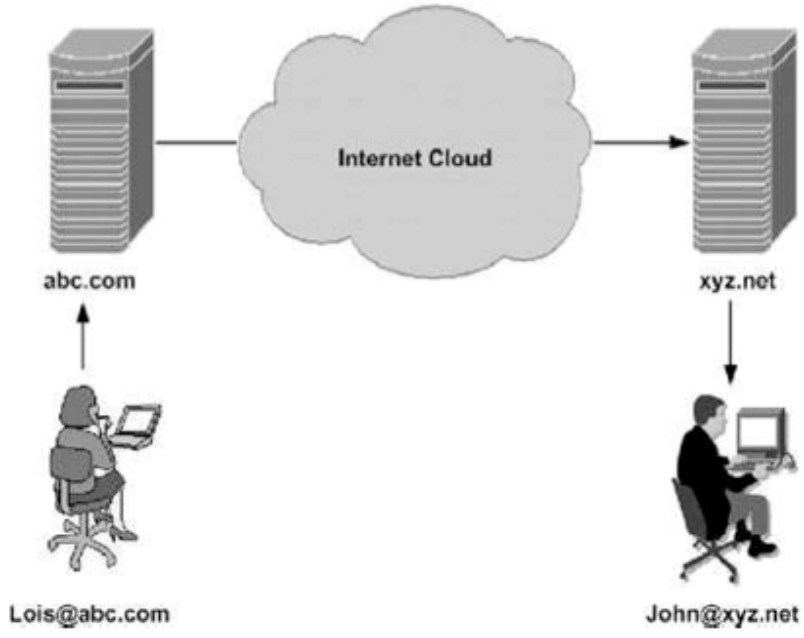


Figura 3: Dos clientes Jabber. [3]

El conocimiento de la presencia es lo que permite que la mensajería sea instantánea. Jabber combina características estándar de la MI con

características adicionales para hacer Jabber único. Estas características incluyen:

1. Jabber usa un conjunto de protocolos abiertos, bien documentados y fáciles de entender.
2. Los protocolos Jabber son basados 100% en XML, lo que permite mensajería estructurada e inteligente entre usuarios humanos y también entre aplicaciones.
3. Jabber usa direcciones iguales a las del correo electrónico (user@host).
4. Jabber utiliza arquitectura cliente servidor, no una arquitectura peer-to-peer como otros sistemas de mensajería.
5. Jabber soporta una arquitectura de red distribuida.
6. Jabber tiene un servidor modular y una arquitectura de cliente simple.

Es importante describir algunos de estos elementos con más detalle:

Protocolos Abiertos

Las tecnologías de Jabber comenzaron en la comunidad del código libre con los servidores y clientes Jabber para Microsoft Windows, MacOS y Linux. Como parte de su trabajo, iniciaron la construcción de un protocolo basado en XML para transmisiones sobre Internet. Su iniciativa posibilitó lo que hoy conocemos como XMPP, tal como se describió anteriormente. Como Jabber usa un protocolo abierto, cualquiera puede implementarlo. Esto ha causado una explosión del software de Jabber, incluyendo servidores completamente libres al igual que clientes.

XML

XML es una parte integral de las tecnologías Jabber. La razón, es que XML posibilita que sus aplicaciones sean extensibles y permite expresar casi cualquier cantidad de datos de forma estructurada. Cuando un cliente se conecta al servidor, este abre una conexión de una vía para la trasmisión de

XML del cliente al servidor, y el servidor responde con una trasmisión de XML de una vía hacia el cliente. Por tanto, cada sesión involucra dos canales de XML. La comunicación entre el cliente y el servidor ocurre a través de estos canales, con pequeños archivos xml como este:

```
<mensaje de='Luisa@ABC.com' para='Juan@XYZ.net'>
     <cuerpo>Donde estas, Juan?</cuerpo>
</mensaje>
```

Arquitectura Cliente/Servidor

Las tecnologías Jabber usan una arquitectura cliente/servidor. Esta arquitectura requiere que toda la información enviada de un cliente a otro, tenga que pasar por el servidor Jabber. Un cliente Jabber se conecta al servidor con un socket TCP por el puerto 5222. Los servidores se conectan entre ellos a través del puerto 5269. Esta conexión es persistente durante toda la vida de la sesión del cliente con el servidor. Esto significa que **el cliente no tiene que preguntar por mensajes al servidor como lo haría un cliente de correos[11].** Cualquier mensaje que se envíe a un cliente Jabber es inmediatamente despachado al cliente si se encuentra conectado. El servidor mantiene un seguimiento de la información de presencia y cuando detecta que un cliente se ha ido para el estado "desconectado", almacena cualquier mensaje enviado a ese cliente para entregarlo la próxima vez que se conecten al servidor Jabber.

La arquitectura cliente/servidor es usada en prácticamente todos los sistemas de MI. Los clientes de MI instalados en los computadores de los usuarios, se comunican con un servidor de MI que provee la infraestructura necesaria para localizar los usuarios e intercambiar los mensajes. Los datos generalmente se envían sin encriptar (Figura 4).

La mensajería peer-to-peer, usada por algunos sistemas de MI, puede resultar mas seguro que la arquitectura cliente servidor en la medida que ambos usuarios están ubicados en la misma red local y los mensajes no

[11] Esta es una de las características más importantes para la construcción del sistema de MI de EAFIT Interactiva pues constituye una forma de atacar los problemas de desempeño que se derivan de realizar preguntas repetitivas al servidor en busca de nueva información.

viajan por la Internet. Los clientes peer-to-peer se conectan al servidor para ubicar a otros clientes. Cuando se ha ubicado el otro peer, la conexión se realiza directamente entre ellas. Obviamente, si un usuario esta ubicado fuera de la red corporativa, el modelo se torna tan inseguro como el cliente/servidor porque el mensaje debe pasar por la Internet para alcanzar su destino (Figura 5).

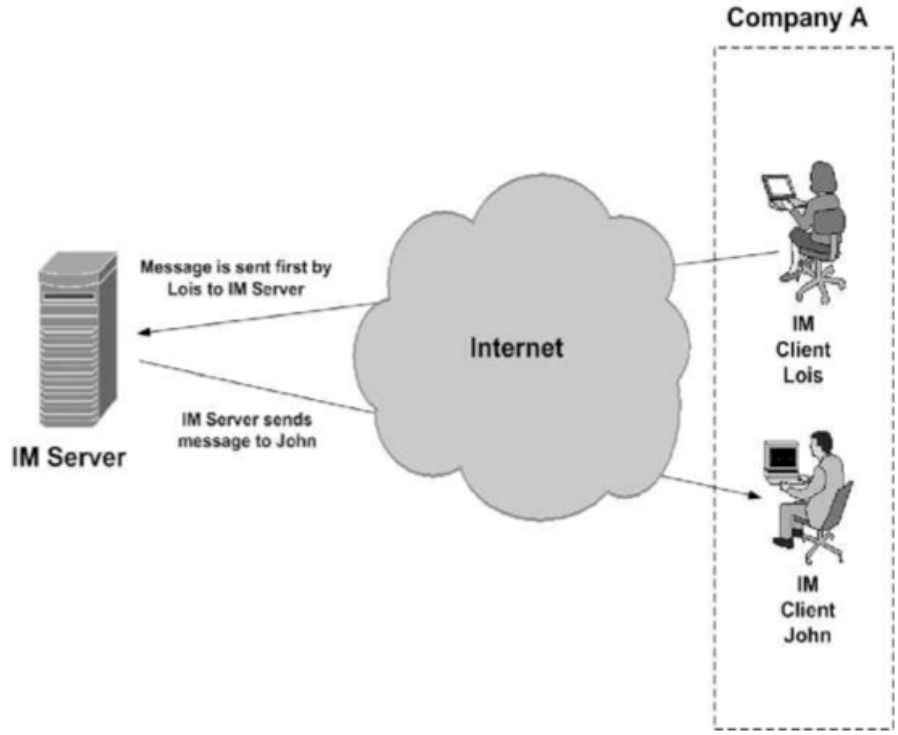


Figura 4: Cliente/Servidor MI [3]

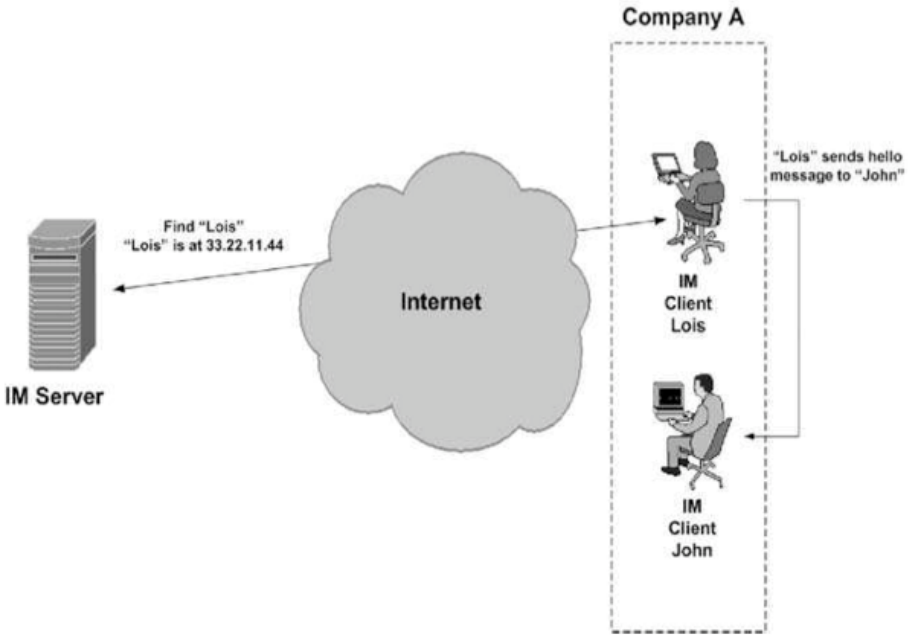


Figura 5: Peer-to-Peer MI [3]

Servidor modular y una arquitectura de cliente simple

Un servidor Jabber ejecuta tres tareas básicas:

1. Maneja las conexiones con los clientes y se comunica directamente con ellos.
2. Maneja las comunicaciones con otros servidores Jabber.
3. Coordina varios componentes asociados con el servidor en si mismo.

Los servidores Jabber están diseñados para ser modulares. Están construidos con paquetes de código, manejando cada uno una funcionalidad específica, como el registro, la autenticación, la presencia, la lista de contactos, el almacenamiento de los mensajes fuera de línea entre muchos otros. Los servidores pueden extenderse con componentes externos que le facilitan a los administradores la tarea de agregar servicios suplementarios a los que tienen por defecto. Las implementaciones de las funcionalidades

pueden ser hechas por cualquier persona o equipo, y no necesariamente la comunidad Jabber debe participar en el desarrollo.

Desarrollar Clientes para Jabber también resulta particularmente sencillo, solo debe tenerse en cuenta la comunicación a través de sockets TCP, reconocer la sintaxis del XML y entender los tipos de datos del núcleo de Jabber.

5.1.4.3 SIP (Session Initiation Protocol)[12]

El protocolo SIP constituye una sólida base que puede ser usada por los proveedores de servicios para fortalecer el poder del protocolo de Internet (IP) y transformar sus formas de transmisión tradicionales. El mundo de las redes esta atravesando un cambio: las redes están convergiendo, las comunicaciones y la computación se están volviendo inseparables. En el centro de esta evolución esta SIP: es el mecanismo que une los servicios entre plataformas creando así una multiplicidad de nuevas posibilidades.

SIP es un protocolo usado para establecer sesiones en una red IP. Una sesión puede ser simplemente una llamada telefónica de dos vías pero también puede ser una sesión de conferencia multimedia colaborativa con muchos participantes. La capacidad de establecer este tipo de sesiones, significa un nuevo mundo de posibilidades y de servicios innovadores como un sistema de comercio electrónico enriquecido por voz, paginas Web con asistencia telefónica, mensajería instantánea con lista de contactos entre muchos otros. Durante los últimos años, la comunidad de voz sobre IP, ha adaptado a SIP como su protocolo principal.

SIP implementa sus sistemas de MI a través de los siguientes cuatro pasos (Figura 6):

1. Cliente X y Y usan un SIP registrador para registrar su existencia.
2. Cliente Y se suscribe pidiendo actualizaciones de MI sobre el cliente X.

[12] Definido en el RFC 3261

3. El cliente X hará notificaciones reportando su presencia y disponibilidad a su amigo, el cliente Y.

4. El mensaje será intercambiado a través de los clientes de MI X y Y

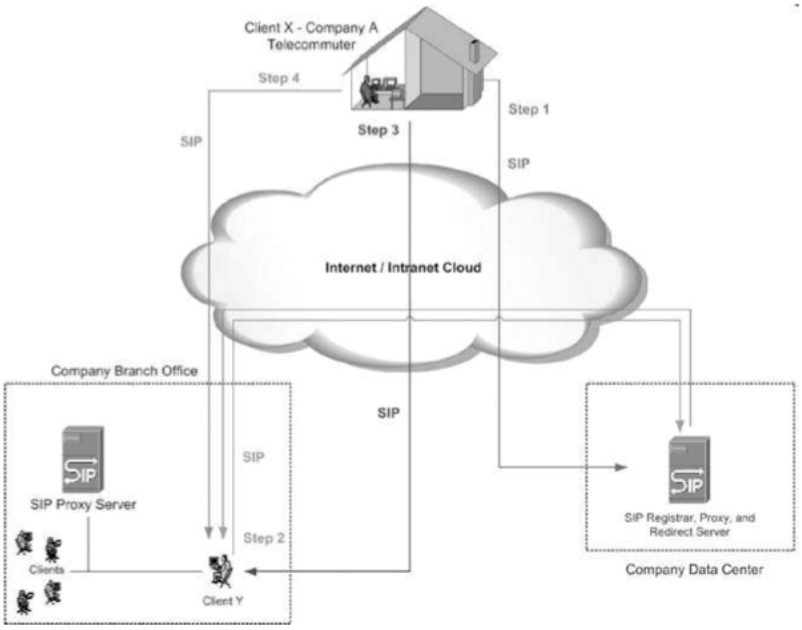


Figura 6: MI con SIP [3]

Hasta ahora, se han descrito, dos de los protocolos mas difundidos en la comunidad de la mensajería instantánea. Sin embargo, existen muchos otros protocolos propietarios entre los cuales pueden destacarse[13]:

Protocolo	Creador	Fecha
MNSP (Windows live Messenger)	Microsoft	1999 Julio
OSCAR (AIM, ICQ)	AOL	1997
YMSG (Yahoo!	Yahoo!	-

[13] Una lista completa de protocolos en http://en.wikipedia.org/wiki/Comparison_of_instant_messaging_protocols (Link revisado el día 7 de septiembre de 2008)

Messenger)		
Skype	Skype	-

Tabla 1: Algunos sistemas de mensajería instantánea. [28]

Hemos revisado unos de los protocolos más difundidos de la mensajería instantánea. Sin duda, Jabber y SIP constituyen dos de las aproximaciones mas aceptadas en todo el mundo, por ser estándares abiertos y estar soportados por una amplia y sólida comunidad de usuarios.

Si bien son protocolos que deben considerarse cuando se quiere incursionar en el mundo de la mensajería instantánea, veremos algunas razones de peso por las cuales su implementación no es posible en la plataforma de EAFIT INTERACTIVA, y analizaremos las salidas más efectivas a este problema.

6. MENSAJERÍA INSTANTÁNEA EN LA PLATAFORMA DE EAFIT INTERACTIVA

6.1 Requisitos (Versión Cliente)

6.1.1. Organizaciones

Organización	Universidad EAFIT
Dirección	Carrera 49 Nº 7 Sur - 50 | Medellín - Colombia - Suramérica
Teléfono	(57) (4) - 2619500
Fax	000
Comentarios	Ninguno

6.1.2. Participantes

Participante	**Santiago Oquendo Solano**
Organización	Estudiante Pregrado
Rol	Ingeniero de Requisitos
Es desarrollador	Sí
Es cliente	No
Es usuario	Sí
Comentarios	Ninguno

Participante	**Carolina Pabón**
Organización	Universidad EAFIT
Rol	Ingeniera EAFIT Interactiva
Es desarrollador	No
Es cliente	Sí
Es usuario	Sí
Comentarios	Ninguno

6.1.3. Actores

ACT-0001	Usuario
Versión	1.1 (25/08/2008)
Autores	Santiago Oquendo Solano
Fuentes	Carolina Pabón
Descripción	Este actor representa a los estudiantes y profesores que tienen acceso a

	EAFIT INTERACTIVA
Comentarios	Ninguno

6.1.4. Objetivos

OBJ-0001	**Comunicar a los usuarios de EAFIT Interactiva**
Versión	1.1 (25/08/2008)
Autores	Santiago Oquendo Solano
Fuentes	Carolina Pabón
Descripción	El sistema deberá comunicar a los usuarios de EAFIT INTERACTIVA a través de la mensajería instantánea
Subobjetivos	Ninguno
Importancia	vital
Urgencia	inmediatamente
Estado	En construcción
Estabilidad	alta
Comentarios	Ninguno

6.1.5 Requisitos funcionales

UC-0001	**Inicio de la Comunicación**	
Versión	1.1 (25/08/2008)	
Autores	Santiago Oquendo Solano	
Fuentes	Carolina Pabón	
Dependencias	Ninguno	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el usuario entre a alguna de las materias registradas en EAFIT INTERACTIVA	
Precondición	El usuario ha entrado a EAFIT INTERACTIVA y ha seleccionado una de sus materias	
Secuencia normal	**Paso**	**Acción**
	1	El sistema muestra una lista de contactos con todos los usuarios conectados de la materia seleccionada. Cada elemento de la lista consta de la foto, el nombre completo y la fecha de ingreso del usuario.
	2	El sistema notifica a los usuarios conectados de la materia que un usuario nuevo acaba de llegar
	3	El actor Usuario (ACT-0001) inicia la comunicación con uno de los usuarios conectados haciendo clic sobre su nombre
	4	El sistema abre una ventana emergente y la ubica en el extremo inferior derecho de la pantalla. La ventana tiene un titulo con el nombre completo del usuario, un espacio para ver los mensajes, una

		caja de texto y un botón de cerrar.
Postcondición	El usuario se encuentra registrado en el sistema de mensajería instantánea y esta listo para comunicarse con los usuarios conectados	
Excepciones	**Paso**	**Acción**
	1	Si el sistema detecta que un usuario nuevo se ha conectado a la materia, el sistema debe actualizar de inmediato la lista de contactos agregando este nuevo usuario, a continuación este caso de uso continúa
	4	Si el Usuario ya tiene alguna ventana emergente abierta y quiere iniciar una comunicación con otro usuario, el sistema debe ubicar la nueva ventana al lado izquierdo de aquella(s) que ya estén en el extremo inferior derecho de la pantalla, a continuación este caso de uso continúa

UC-0002	**Envío de mensajes**	
Versión	1.1 (25/08/2008)	
Autores	Santiago Oquendo Solano	
Fuentes	Carolina Pabón	
Dependencias	Ninguno	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el Usuario se dispone a enviar los mensajes	
Precondición	El usuario ya tiene abierta la ventana emergente del usuario con el que quiere hablar tal como se detalla en el UC-0001	
Secuencia normal	**Paso**	**Acción**
	1	El actor Usuario (ACT-0001) escribe su mensaje dentro de la caja de texto de la ventana del usuario con el que quiere hablar
	2	El actor Usuario (ACT-0001) presiona la tecla ENTER cuando termina su mensaje para enviarlo
	3	El sistema muestra este mensaje en el espacio para mensajes de la forma "yo: mensaje"
	4	El sistema envía el mensaje a su destinatario
Postcondición	El usuario envía el mensaje a su destinatario	
Excepciones	**Paso**	**Acción**
	3	Si el sistema determina que el usuario destinatario ya no se encuentra conectado, el sistema deberá informar este hecho al usuario remitente con un mensaje como: "El mensaje no pudo enviarse porque el usuario ya no esta conectado", a continuación este caso de uso queda sin efecto
	3	Si el sistema encuentra en el texto del mensaje cadenas que puedan ser representadas como imágenes (emoticons), el sistema debe mostrar estas imágenes en el espacio de los mensajes del usuario remitente y del destinatario, a continuación este caso de uso continúa

UC-0003	**Recepción de Mensajes**
Versión	1.1 (25/08/2008)
Autores	Santiago Oquendo Solano
Fuentes	Carolina Pabón

Dependencias	Ninguno	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el usuario recibe un mensaje	
Precondición	El usuario ha ingresado a EAFIT INTERACTIVA y ha seleccionado una de sus materias	
Secuencia normal	**Paso**	**Acción**
	1	Si no existe una ventana de conversación con el usuario remitente, el sistema deberá crear esta ventana con el nombre del usuario remitente y ubicarla tal como se describe en UC-0001. Además en el espacio de mensajes, el sistema debe poner el mensaje de la forma: "Nombre usuario remitente: mensaje"
	2	Si existe una ventana de conversación con el usuario remitente, el sistema debe poner en el espacio de mensajes el mensaje de la forma: "Nombre usuario remitente: mensaje"
Postcondición	El usuario recibe el mensaje de su remitente	

UC-0004	**Cierre de la ventana de conversación**	
Versión	1.1 (25/08/2008)	
Autores	Santiago Oquendo Solano	
Fuentes	Carolina Pabón	
Dependencias	Ninguno	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el usuario quiere cerrar alguna de las ventanas de conversación que tiene activas en el extremo inferior derecho de la pantalla	
Precondición	Ventana de conversación abierta	
Secuencia normal	**Paso**	**Acción**
	1	El actor Usuario (ACT-0001) hace clic sobre el botón cerrar de la ventana que quiere cerrar
	2	El sistema cierra la ventana y la elimina del extremo inferior derecho de la pantalla
	3	Si hay mas de una ventana abierta, el sistema debe reubicar las ventanas en el extremo inferior derecho de la pantalla de tal manera que se llene el espacio vacío de la ventana cerrada
Postcondición	Ventana de conversación cerrada	

UC-0005	**Fin de la comunicación**	
Versión	1.1 (25/08/2008)	
Autores	Santiago Oquendo Solano	
Fuentes	Carolina Pabón	
Dependencias	Ninguno	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el usuario sale de la materia o del sistema	
Precondición	EL usuario se encuentra en alguna de las materias de EAFIT INTERACTIVA	
Secuencia normal	**Paso**	**Acción**
	1	El actor Usuario (ACT-0001) cambia de materia o sale del sistema
	2	El sistema notifica a todos los usuarios conectados de la materia que el usuario ha salido y lo elimina de cada una de las listas de contactos

Postcondición	El usuario sale del sistema de mensajería instantánea

UC-0006	**Ubicación de las ventanas**	
Versión	1.1 (25/08/2008)	
Autores	Santiago Oquendo Solano	
Fuentes	Carolina Pabón	
Dependencias	Ninguno	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el usuario mueve las barras de desplazamiento o cambia el tamaño de la ventana del navegador	
Precondición	Ventanas de conversación abiertas	
Secuencia normal	**Paso**	**Acción**
	1	El actor Usuario (ACT-0001) mueve las barras de desplazamiento o cambia el tamaño de la ventana del navegador
	2	El sistema deberá reubicar las ventanas que se encuentren abiertas de tal forma que siempre permanezcan en el extremo inferior derecho de la pantalla
Postcondición	Ventanas de conversación ubicadas	

6.1.6 Requisitos no funcionales

NFR-0001	**Integración a Eafit Interactiva**
Versión	1.1 (25/08/2008)
Autores	Santiago Oquendo Solano
Fuentes	Carolina Pabón
Dependencias	Ninguno
Descripción	El sistema deberá *ajustarse al diseño actual de la plataforma de EAFIT INTERACTIVA y debe ser diseñado como un modulo independiente que no requiera cambios en la estructura actual de la plataforma*

NFR-0002	**Velocidad de respuesta**
Versión	1.1 (25/08/2008)
Autores	Santiago Oquendo Solano
Fuentes	Carolina Pabón
Dependencias	Ninguno
Descripción	El sistema deberá *enviar y recibir los mensajes en tiempo real*

6.2. Requisitos (Desarrollador)

6.2.1. Requisitos Funcionales

FRQ-0001	**Tiempo real**
Versión	1.0 (06/09/2008)
Autores	Santiago Oquendo
Fuentes	Carolina Pabón
Dependencias	Ninguno
Descripción	El sistema deberá *enviar y recibir los mensajes en tiempo real sin hacer peticiones cada segundo al servidor (Latencia igual a 30 segundos)*

FRQ-0002	**Protocolo basado en eventos**
Versión	1.0 (06/09/2008)
Autores	Santiago Oquendo
Fuentes	Carolina Pabón
Dependencias	Ninguno
Descripción	El sistema deberá *implementar un protocolo basado en eventos para lograr el envío y recepción de mensajes en tiempo real.*

6.2.2. Requisitos no funcionales

NFR-0001	**Lenguaje de programación**
Versión	1.0 (06/09/2008)
Autores	Santiago Oquendo
Fuentes	Carolina Pabón
Dependencias	Ninguno
Descripción	El sistema deberá *desarrollarse en el lenguaje de programación JAVA*
Importancia	vital
Urgencia	inmediatamente
Estado	en construcción
Estabilidad	alta
Comentarios	Ninguno

NFR-0002	**Manejo de Estilos**
Versión	1.0 (06/09/2008)
Autores	Santiago Oquendo
Fuentes	Carolina Pabón
Dependencias	Ninguno

Descripción	El sistema deberá *adoptar los estilos CSS de la plataforma de EAFIT Interactiva*

NFR-0003	**Acceso a base de datos**
Versión	1.0 (06/09/2008)
Autores	Santiago Oquendo
Fuentes	Carolina Pabón
Dependencias	Ninguno
Descripción	El sistema deberá *construirse sin la necesidad de acceder a la base de datos para verificar la llegada de nuevos usuarios o mensajes.*

NFR-0004	**Implementación del protocolo**
Versión	1.0 (06/09/2008)
Autores	Santiago Oquendo
Fuentes	Carolina Pabón
Dependencias	Ninguno
Descripción	El sistema deberá *implementar el protocolo de mensajería sin apoyarse en instalaciones de protocolos existentes como Jabber*

NFR-0005	**Instalaciones en el servidor**
Versión	1.0 (06/09/2008)
Autores	Santiago Oquendo
Fuentes	Carolina Pabón
Dependencias	Ninguno
Descripción	El sistema deberá *funcionar sin requerir la instalación de ningún tipo de software de terceros en los servidores donde reside EAFIT Interactiva*
Importancia	Vital
Urgencia	Inmediatamente
Estado	en construcción
Estabilidad	Alta
Comentarios	Ninguno

NFR-0006	**Patrón MVC**
Versión	1.0 (06/09/2008)
Autores	Santiago Oquendo
Fuentes	Carolina Pabón
Dependencias	Ninguno
Descripción	El sistema deberá *estar construido bajo el patrón MVC (Modelo, Vista, Controlador)*

A partir de este momento, argumentaremos las razones por las cuales la aplicación de mensajería instantánea para EAFIT Interactiva fue construida en el lenguaje de programación JAVA, utilizando específicamente el Google Web Toolkit (GWT).

6.2 Limitando el problema

Como puede verse en la lista de requisitos del apartado 6.1, el dominio del problema se estrecha a unas características particulares, donde las soluciones posibles se van acortando. Por si aun no existe claridad, resumamos de forma global las características mas importantes del sistema de mensajería instantánea para la plataforma de EAFIT Interactiva, pensando sobre todo en aquellos elementos que nos ayudarán a tomar una decisión sobre que tecnología emplear:

Restricciones del servidor

Respetando las políticas propias del Centro de Informática de la Universidad, es imposible instalar en el servidor de la plataforma de EAFIT Interactiva cualquier tipo de software. Es decir, para el caso del protocolo Jabber, discutido en secciones anteriores, es necesario instalar un servidor de MI Jabber dentro del entorno de la plataforma. Este servidor abriría algunos puertos y consumiría recursos permanentemente en la maquina donde se instale. Dada la sensibilidad e importancia de la plataforma, cualquier instalación del tipo Jabber queda totalmente descartada. Por lo general, estas aproximaciones requieren del Servidor y del Cliente para operar correctamente.

Lenguaje de programación

EAFIT Interactiva esta construida enteramente en Java. Los desarrolladores están solo dispuestos, por cuestiones de mantenibilidad y tiempo a soportar algún componente nuevo que se encuentre desarrollado en este lenguaje. Por tanto, nuestro conjunto se ve limitado ahora por aquellas soluciones basadas en JAVA.

Aplicaciones Cliente

Dado que la objetivo del sistema de MI, esta enfocado a entregar una herramienta de comunicación a los usuarios de EAFIT Interactiva, deben descartarse todos los clientes que requieran de un instalador en el sistema operativo de los usuarios. Por el contrario, el enfoque debe extenderse a aquellas aproximaciones que traten de incorporar dentro de una aplicación Web preexistente aquello que sirve de cliente pero que se ejecute a través del navegador y como un agregado de la plataforma sin requerir trabajo alguno por parte de los usuarios.

Antes de continuar, veamos algunas de las soluciones que se descartaron como consecuencia de los puntos anteriores (De entrada descartaremos aquellos que tengan un valor comercial):

Nombre del Servidor	**Lenguaje**	**Sistemas Operativos**
DJabberd: Conjunto de módulos en Perl para construir un servidor Jabber propio.	Perl	Todos Se requiere compilador de Perl
Ejabberd: Servidor Jabber modular, tolerante a fallos con varios servicios y herramientas administrativas	Instalador	Todos
Jabberd: Servidor XMPP	C	Todos Se requiere compilador de C

Tabla 2: Lista de algunos servidores Jabber. [9]

Si bien, no es posible instalar ninguno de los servidores Jabber, los clientes que utilizan este protocolo, quedan automáticamente descartados, además de que algunos de ellos no están diseñados para integrarse en la Web o están construidos en lenguajes diferentes a JAVA. Por mencionar tenemos los siguientes (Figura 7):

Name	Platforms	Comments
Bombus	J2ME (MIDP2.0)/WinCE	
Bombusmod	J2ME (MIDP2.0)	
Coccinella	Cross-platform	Tcl/Tk
Exodus	Windows	
Gabber	Linux/Unix	GTK+
Gajim	Cross-platform	PyGTK
GOIM	Cross-platform	Eclipse Rich Client Platform
Gossip	Linux/Unix	GTK+
JabberMixClient	J2ME (MIDP2.0)	
Jabbim	Cross-platform	PyQt
Jabbin	Cross-platform	Qt
Jeti	Cross-platform	Java, browser-based
MCabber	Cross-platform	Cabber fork, console client (ncurses)
Mobber	J2ME (MIDP1.0)	
MOO-XMPP	MOO	Runs as a MOO object
Psi	Cross-platform	Qt
Spark	Java	
Tapioca		
Tkabber	Cross-platform	Tcl/Tk
wija	Java	

Figura 7: Algunos clientes Jabber, con su nombre, plataformas y comentarios. [29]

6.3 Definiendo la solución

Hasta el momento, hemos dedicado varias páginas a describir los problemas relativos a la adopción de algunas de las tecnologías existentes. Por esto, basado en la investigación y en las necesidades de la plataforma, se decide construir una aplicación que contenga los siguientes elementos:

1. Una arquitectura cliente/servidor

2. Una interfaz en el navegador integrada con la presentación actual de EAFIT Interactiva.

3. Un protocolo de comunicación simple con las operaciones básicas de Logueo (login), envío de mensaje y salida (logout).

La experiencia de conocer los sistemas Jabber fue importante en el sentido de que se ratificó la importancia vital de los tres puntos anteriores para diseñar un sistema de mensajería instantánea de calidad.

6.3.1. Construyendo la solución

La solución será desarrollada utilizando herramientas de desarrollo Web de última generación. Las tecnologías que comenzaremos a analizar a partir de este momento, constituyen una base sólida para el trabajo gracias a su difusión y gran proyección en la comunidad del desarrollo Web.

Si aun no está muy clara la razón por la cual se ha optado por tecnologías Web, las siguientes afirmaciones ayudarán a acreditar un poco la decisión que se tomó frente al tema:

- Las restricciones del problema dan indicios de que es mucho más fácil, para el futuro mantenimiento de la plataforma, y para conservar los niveles de servicio requeridos por la Universidad, realizar un desarrollo utilizando las tecnologías y la arquitectura que constituyen a EAFIT Interactiva.

- EAFIT Interactiva está construida cien por ciento en ambientes Web, en una arquitectura cliente/servidor.

- Los requisitos funcionales apuntan a que el sistema de MI deberá estar integrado a la plataforma de una forma limpia, eficiente y sencilla.

- Con el fin de proyectar el futuro de la plataforma, se incluirá por primera vez dentro del sistema las tecnologías Ajax[14] como base del desarrollo del sistema de MI. Sin este concepto, lograr las funcionalidades requeridas seria imposible, de ahí la importancia de entenderlo a profundidad pues la solución esta basada en principios fundamentales de este tema.

Todas las aplicaciones modernas, que tratan de enriquecer la experiencia de usuario con interfaces cada vez mas parecidas a las de una aplicación de escritorio tradicional, basan su arquitectura en la tecnología Ajax (o conjunto de tecnologías como se verá mas adelante). Sin duda, desde su aparición en el año 2005 con el disparo de aplicaciones como Gmail y Google Maps desarrolladas por el titán informático Google, la fuerza de Ajax viene en asenso y constituye en la actualidad unas de las necesidades más importantes para lograr contenido Web de alta calidad.

[14] En el capítulo 7, sección 7.1.1 en adelante se explica este concepto en detalle.

7. TECNOLOGÍAS

7.1 Ajax

"If anything about current interaction design can be called "glamorous," it's creating Web applications. After all, when was the last time you heard someone rave about the interaction design of a product that wasn't on the Web? (Okay, besides the iPod.) All the cool, innovative new projects are online.

Despite this, Web interaction designers can't help but feel a little envious of our colleagues who create desktop software. Desktop applications have a richness and responsiveness that has seemed out of reach on the Web. The same simplicity that enabled the Web's rapid proliferation also creates a gap between the experiences we can provide and the experiences users can get from a desktop application.

That gap is closing. Take a look at Google Suggest. Watch the way the suggested terms update as you type, almost instantly. Now look at Google Maps. Zoom in. Use your cursor to grab the map and scroll around a bit. Again, everything happens almost instantly, with no waiting for pages to reload.

Google Suggest and Google Maps are two examples of a new approach to web applications that we at Adaptive Path have been calling Ajax. The name is shorthand for Asynchronous JavaScript + XML, and it represents a fundamental shift in what's possible on the Web."[15]

---

[15] Tomado del articulo que da inicio oficial al término AJAX: Ajax: A New Approach to Web Applications, Jesse James Garrett, February 18, 2005

7.1.1 Definiendo Ajax

Ajax no es una tecnología. Realmente es un conjunto de tecnologías que cuando se unen posibilitan poderosas y novedosas alternativas. Recibe su nombre de Asynchronous JavaScript[16] + XML (JavaScript Asincrónico[17] + XML).

La parte "asincrónica" significa que el navegador[18] no esperará a que el servidor le retorne información, pero sí es capaz de manejar la información cuando es enviada de regreso. En otras palabras, la transferencia de datos ocurre internamente, sin hacer que el navegador pare y espere que ocurra algo. Esta es una parte crucial de Ajax: es posible manejar los datos del servidor, cuando éste envía esos datos. No es requerido obligar a que toda la aplicación se pare esperando a que los datos lleguen. Si es necesario esperar por los datos, entonces la aplicación seria sincrónica[19], y para conexiones de Internet de baja velocidad esto constituye un problema.

La parte de JavaScript del término Ajax, es también de vital importancia, porque gracias a esto Ajax ocurre en los navegadores. Ajax delega la responsabilidad a JavaScript de conectarse con el servidor y manejar los datos que éste retorna. JavaScript siempre es usado para subir y bajar información del servidor "detrás de escenas" y cuando los datos son descargados, también es usado para mostrarlos u organizarlos de la manera correcta.

En el mundo informático, XML se ha convertido en la lengua franca de la Web, posibilitando una manera de enviar y recibir datos en forma de texto a través de la Internet. La razón por la cual XML se ha convertido en un fenómeno tan popular es porque la Internet esta diseñada para manejar documentos en formato texto. Por esta razón, las aplicaciones Ajax son

---

16 JavaScript es un lenguaje de programación interpretado, es decir, que no requiere compilación, utilizado principalmente en páginas Web.

17 Tipo de transmisión en el cual cada carácter es transmitido de forma independiente e individual, al margen de cualquier referencia de tiempo.

18 Programa que se usa para entrar a las páginas en Internet, por ejemplo, Internet Explorer, Firefox, Opera.

19 Eventos que ocurren al mismo tiempo.

escritas a menudo para manejar información enviada desde el servidor usando XML. En otras palabras, cuando se hace contacto con un servidor, enviará datos de respuesta en un documento de XML.

Sin embargo, XML es solo una de las formas de manejar la información enviada desde el servidor. Es posible también enviar texto plano, o porciones de código HTML.[20]

A parte de JavaScript y XML, Ajax trabaja también con HTML y CSS[21]. Estas dos tecnologías permiten actualizar la información que es mostrada en una página Web; y dado que con Ajax no se recarga toda la página de nuevo sino una parte de ella, la responsabilidad de HTML y CSS es muy grande porque ambas permiten actualizar partes específicas de la página Web.

La parte de JavaScript que hace posible el uso de Ajax, es el objeto conocido con el nombre de XMLHttpRequest. Éste es un objeto especial incorporado en todas las versiones modernas de JavaScript en los navegadores de la actualidad.

Entonces, Ajax es una colección de tecnologías, no una sola. El objeto XMLHttpRequest es usado para conectarse al servidor y luego para manejar el texto plano o XML que el servidor envía usando JavaScript, por último, HTML y CSS entran a mostrar los resultados en el navegador. Todo esto sin refrescar ni una sola vez la página.

7.1.2 Comparación con el modelo Web Tradicional

Para iniciar la comparación, observemos la figura 8.

---

[20] Es el lenguaje de marcado predominante para la construcción de páginas web.

[21] Las hojas de estilo en cascada (Cascading Style Sheets, CSS) son un lenguaje formal usado para definir la presentación de un documento estructurado escrito en HTML o XML. La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación.

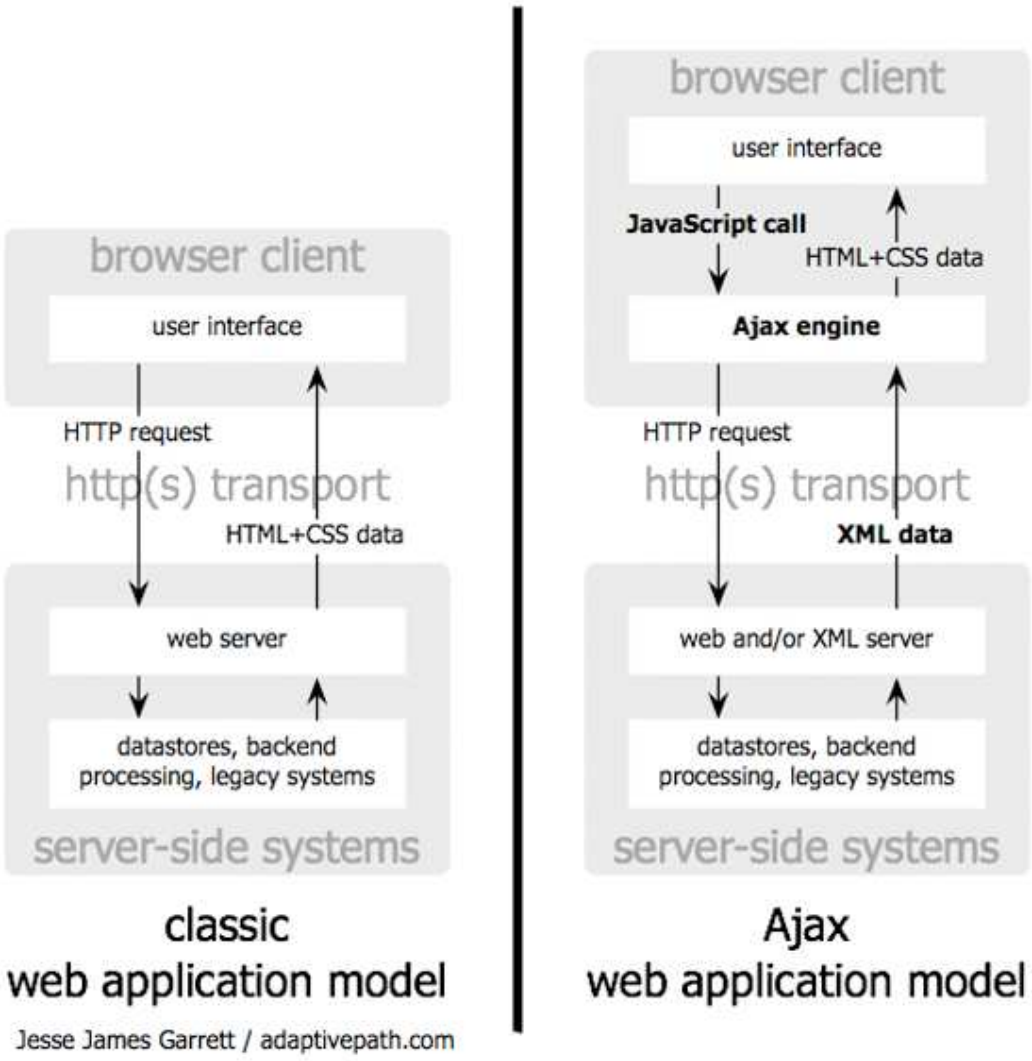


Figura 8: El modelo tradicional para aplicaciones Web (izquierda) comparado con el modelo Ajax (derecha) [12]

Como puede verse en la parte izquierda de la gráfica, en las aplicaciones clásicas de la Web, cualquier acción del usuario en la interfaz dispara una petición HTTP[22] a un servidor Web. El servidor realiza el procesamiento requerido según la acción, bien sea interactuando con una base de datos, realizando un cálculo (o cualquier otra operación) y luego retorna una página HTML al cliente.

Si bien esta aproximación resulta ser bastante lógica en términos técnicos, no posibilita una experiencia de usuario óptima o especial. Mientras el servidor

[22] El protocolo de transferencia de hipertexto (HTTP, HyperText Transfer Protocol) es el protocolo usado en cada transacción de la Web.

esta haciendo lo suyo, el usuario se encuentra esperando. Y con cada paso en una tarea, el usuario espera más y más.

Obviamente, al diseñarse una aplicación, lo menos que se quiere es infringir en tiempo muerto para los usuarios, haciéndoles esperar constantemente en cada interacción con el servidor.

Como lo veíamos en la definición, en la gráfica de la izquierda vemos como cada acción del usuario dispara un llamado JavaScript a un motor Ajax apoyado en el objeto XMLHttpRequest, el cual a su vez, valiéndose de la misma petición http, envía la información al servidor para que realice la tarea respectiva. Luego el servidor devuelve los datos de resultado en forma de XML o texto y es recibido de nuevo por el objeto XMLHttpRequest que se vale de otras facilidades de JavaScript para manipular la página e incorporar de este modo el HTML y CSS necesario.

A simple viste, resulta lógico pensar que el resultado obtenido con ambos modelos es el mismo, y que pese a unos pasos de más en el modelo de Ajax, las ventajas no son visibles.

7.1.3. Cómo se diferencia Ajax

Una aplicación Ajax elimina la naturaleza “inicie-pare-inicie-pare” de la interacción Web agregando un motor Ajax intermediario entre el usuario y el servidor. Parecería que agregar una nueva capa a la aplicación la haría menos eficiente, pero ocurre exactamente lo contrario.

En la figura 9 observaremos la gran diferencia entre el modelo tradicional y el modelo Ajax para el desarrollo Web. Esta diferencia se encuentra enmarcada en dos conceptos, lo sincrónico, y lo asincrónico.

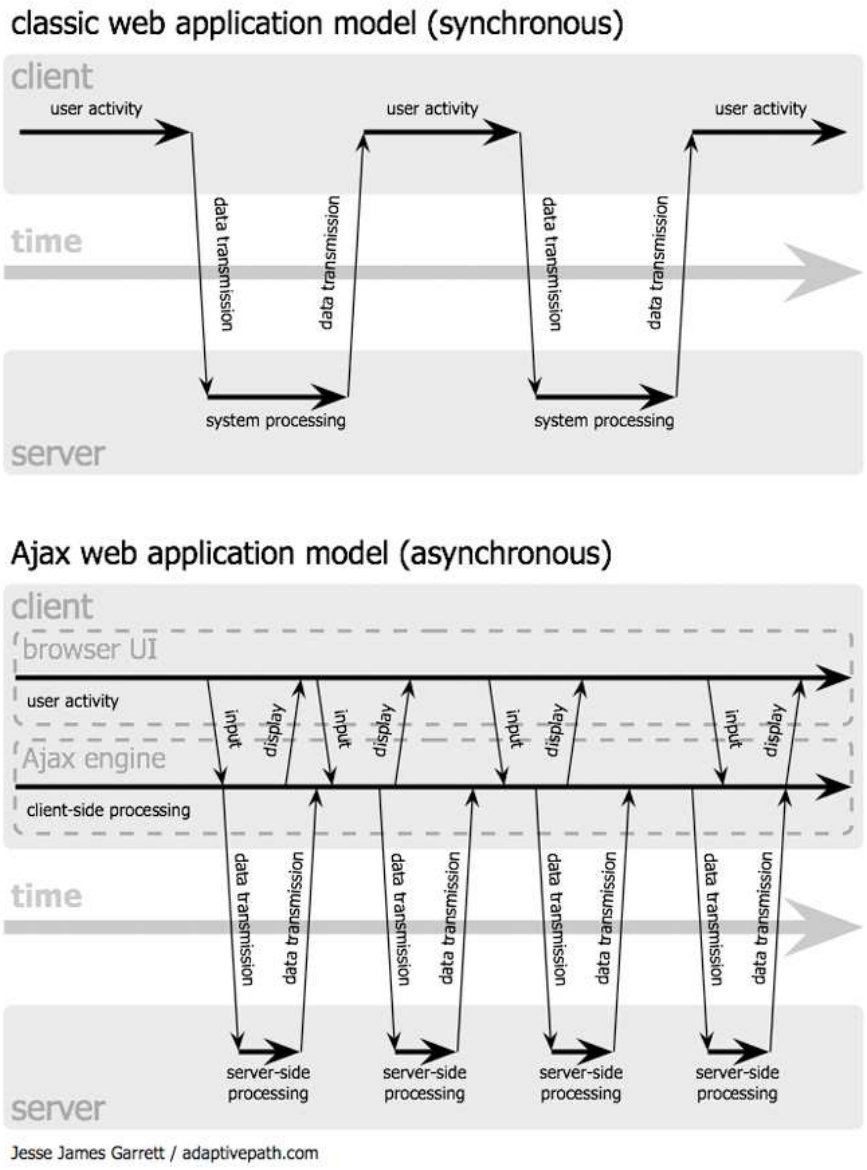


Figura 9: El patrón sincrónico del modelo tradicional (arriba) y el patrón asincrónico del modelo Ajax (abajo). [12]

Obsérvese en la parte de arriba de la grafica, cómo cada vez que hay un procesamiento en el servidor la línea de actividad de los usuarios se ve interrumpida. Es en estos momentos cuando vemos las pantallas blancas en los navegadores cada vez que realizamos alguna acción. Es por esto que hablamos de sincronía, pues es necesario suspender una actividad para dar paso a la siguiente, y la continuidad de las actividades depende de las respuestas que cada una vaya dando.

En contraparte, en el modelo de Ajax, se ve claramente la continuidad de la línea de actividad del usuario aun cuando se están realizando procesamientos en el servidor. Se habla de asincronía, porque las peticiones al servidor pueden realizarse de forma independiente y en cualquier momento de la línea del tiempo sin interrumpir las actividades normales del usuario dentro de la interfaz gráfica de usuario.

Pese a todo lo anterior, Ajax tradicional no puede obtener información en tiempo real directamente desde el servidor. Si cualquier clase de información llega después de que el usuario ha realizado una acción, solo podrá verla hasta que él solicite de nuevo la página. Para atacar esta limitante, surge lo que se conoce como Reverse Ajax.

Para entender un poco la diferencia entre ambas tendencias de Ajax, supongamos que tenemos tres clientes y un servidor, y que el cliente uno envía el mensaje ¨Hola¨ al resto de los clientes.

Con Ajax Tradicional

- El cliente 1 envía el mensaje "Hola".
- El servidor recibe el mensaje "Hola".
- El cliente dos hace una petición al servidor.
- El cliente dos recibe el mensaje “Hola”.
- El cliente tres hace una petición al servidor.
- El cliente tres recibe el mensaje “Hola”.

Con Reverse Ajax

- El cliente uno envía el mensaje “Hola”.
- El servidor recibe el mensaje “Hola”
- El servidor envía el mensaje al cliente dos y al tres.

Como puede observarse, con reverse Ajax se genera muchos menos tráfico y los mensajes son transferidos con menor retardo. Esto es un elemento clave para nuestra aplicación de mensajería instantánea.

7.1.4. Reverse Ajax

Como se dijo anteriormente, surge ante la limitante que tiene Ajax para manejar de forma instantánea información que llegue al servidor en un momento del tiempo. El término Reverse Ajax también recibe el nombre de programación Cometa, web de dos vías (Two-way-web), http Streaming, http Server push, entre otros.

Este problema, generalmente se resuelve con un código JavaScript que automáticamente refresque la página cada cierto tiempo, habitualmente segundos para lograr el efecto de tiempo real. Pero refrescar una página entera en un intervalo de segundos es mortal para el servidor. Si los millones de usuarios de un sitio refrescan la página cada segundo el consumo de ancho de banda se vería incrementado dramáticamente. En realidad, lo único que se necesita es actualizar una pequeña cantidad de datos en tiempo real para que el navegador sea actualizado cuando estos datos llegan al servidor. La forma mas sencilla es hacer que el servidor avise a todos los clientes que los datos han cambiado pero el problema es que los servidores no tienen forma de iniciar contacto con los navegadores. De ahí salen algunas de las soluciones que serán explicadas mas adelante.

7.1.4.1. Polling

Es la repetición de peticiones hecha por un cliente al servidor. La diferencia, es que en vez de refrescar toda la página, solo se modifican los datos que deben ser actualizados (**Figura 10).**

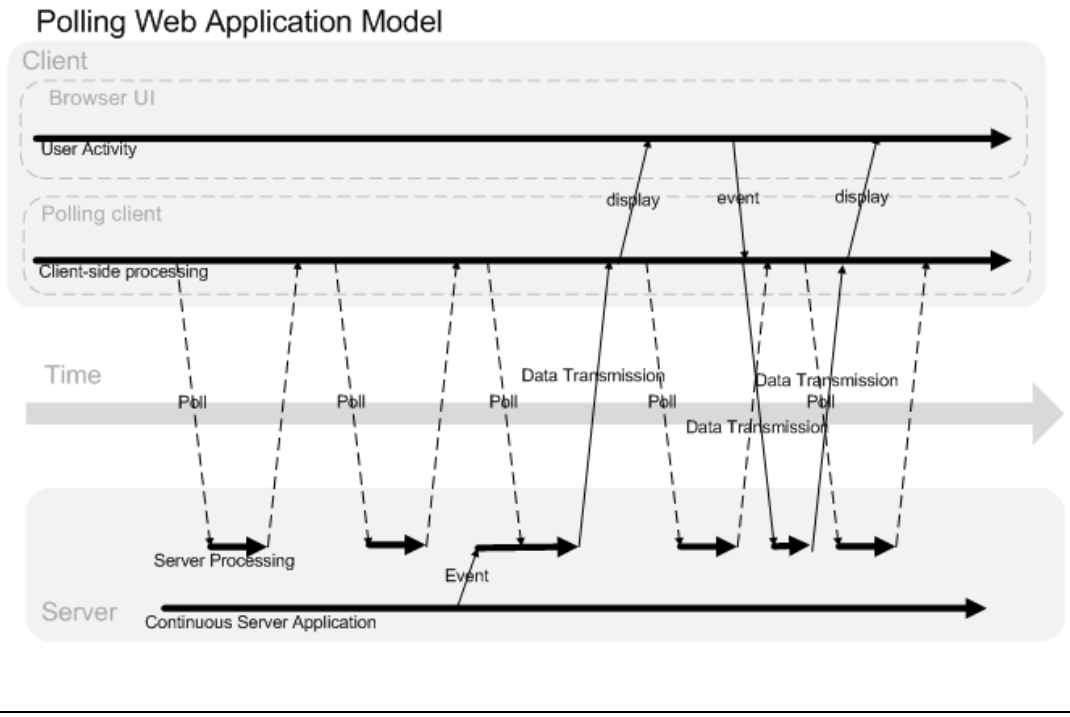


Figura 10: Poolling [13]

En la gráfica puede observarse una línea llamada Aplicaciones continuas en el servidor, que representan procesos de actividad en tiempo real que se llevan a cabo en el servidor, como por ejemplo, actualización del valor de las acciones de la bolsa de Nueva York o de los resultados de un partido de football. Cuando la información llega al servador este dispara un evento que es entregado al Procesamiento del servidor que es el área donde se manejan las peticiones recibidas de los clientes.

Una vez que la página es cargada, un código JavaScript en el cliente se encarga de hacer peticiones al servidor en un intervalo de tiempo programado (unos pocos segundos). Estas peticiones continuas se muestran en la gráfica como líneas punteadas hacia abajo. Si no hay información nueva en el servidor, este retorna una respuesta vacía representada por líneas punteadas hacia arriba. Si el servidor tiene datos nuevos, estos son almacenados hasta la próxima vez que el cliente haga una petición al servidor. Las líneas gruesas hacia arriba representan las ocasiones en las que el servidor envío datos al cliente.

El problema de esta técnica, es que el servidor deberá cargar de nuevo con millones de peticiones (aunque no sean páginas completas) de todos los usuarios cada segundo si se quiere una respuesta rápida que genere los efectos de tiempo real.

7.1.4.2. Cometa

Ésta técnica requiere que la conexión entre el cliente y el servidor nunca se cierre. Con esto lo que se busca, es eliminar los problemas del Polling en cuanto a la frecuencia en la que se tienen que hacer las peticiones al servidor. En vez de preguntar al servidor constantemente, lo que permite el hecho de tener una conexión HTTP duradera es enviar datos a través de ese canal al cliente sin que este haya realizado una petición previa.

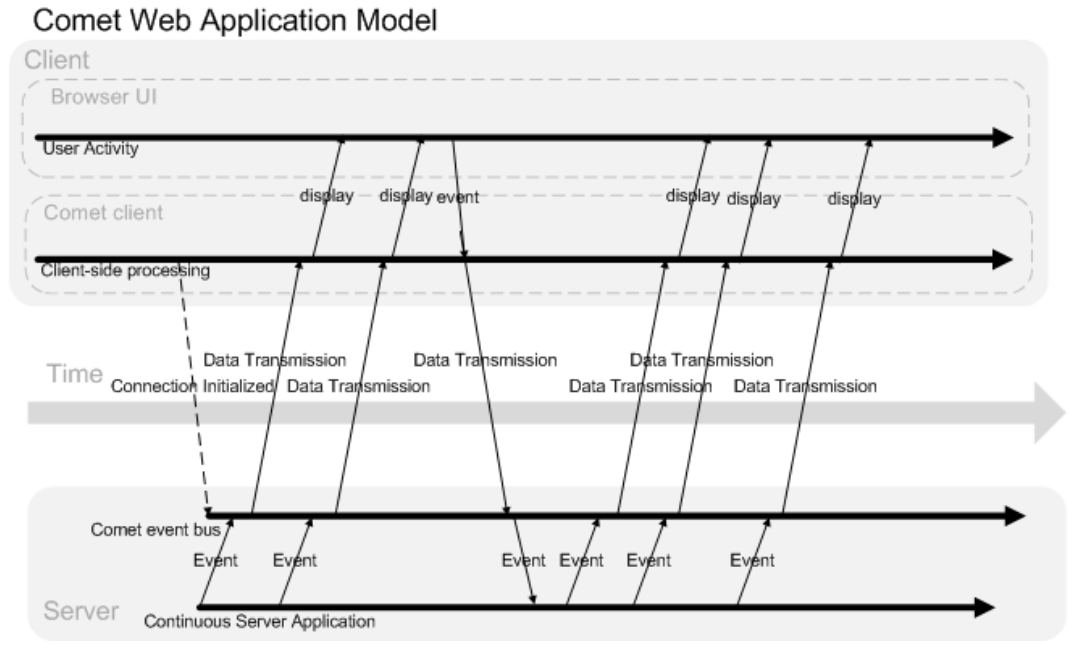


Figura 11: Cometa [14]

Los datos son enviados a través de una conexión HTTP única. Esto reduce la latencia para el envío de datos de forma considerable. El hecho de que puedan informar de cambios de estado con una latencia casi imperceptible, la programación cometa es especialmente ideal para herramientas de monitoreo, ambientes de colaboración multi-usuario y otro tipo de

aplicaciones de este estilo que serian inmanejables sin un agregado especial en el navegador.

Observado la Figura 11 de izquierda a derecha, puede verse en forma de línea punteada hacia abajo la petición inicial del cliente al servidor. Esto abre el bus de eventos, permitiendo que los eventos del lado del servidor viajen hasta el cliente y actualicen la interfaz de usuario del navegador.

7.1.4.2.1. Algunas técnicas de la programación Cometa

7.1.4.2.1.1. Marco Oculto

Una técnica básica para aplicaciones web dinámicas es el uso de los llamados IFrames de forma oculta. Un IFrame (inline frame) es una etiqueta que permite insertar un documento HTML dentro de otro. Cuando un evento ocurre, el Iframe es rellenado con código JavaScript que es ejecutado en el navegador. Como los navegadores procesan las páginas de forma incremental, cada código es ejecutado en la medida que va llegando.

Uno de los beneficios de los marcos ocultos, es que trabaja en todos los navegadores comunes. La parte en contra, es la falta de confiabilidad en la parte del manejo de errores y la imposibilidad de hacer un seguimiento al estado en el que se encuentra una petición.

7.1.4.2.1.2. XMLHttpRequest

El objeto básico de Ajax también puede usarse para la comunicación servidor-navegador con cometa de varias formas. El problema de esta aproximación es que no es soportada por todos los navegadores modernos.

7.1.4.2.1.3. Ajax con peticiones largas

Con esta técnica el cliente hace una petición al servidor por un evento o una serie de eventos. Esta petición estilo Ajax al servidor, se mantiene abierta hasta que el servidor tenga datos nuevos que enviar al navegador. Luego el

navegador inicia una nueva petición larga con el fin de obtener los eventos siguientes.

Hasta el momento, se han presentado una serie de elementos importantísimos para entender como funciona la solución del sistema de MI para EAFIT Interactiva. Ajax es sin duda, uno de sus componentes esenciales, mas aún, la parte que acaba de estudiarse de programación cometa. Si observamos, las aplicaciones que en la actualidad son el pilar o la muestra de la forma en que esta tecnología debe emplearse, encontraremos que la gran mayoría entran al conjunto de aplicaciones de mensajería instantánea en la Web. Algunos ejemplos de estas aplicaciones son:

- La mensajería instantánea integrada en el correo electrónico GMAIL[23] de Google.
- Meebo[24]: Una de las comunidades de mensajería instantánea mas grande de la Web.

A continuación observaremos algunos de los problemas que trae consigo el uso de tecnologías Ajax, argumentos que también nos ayudaran a seleccionar la tecnología con la cual se hará el desarrollo de la aplicación de MI para EAFIT Interactiva.

7.1.5. Los problemas más comunes de Ajax

La página creada dinámicamente no se registra a sí misma en el historial del navegador, así que al dar clic en "Atrás" en las opciones del navegador no va a dar el resultado deseado.

---

[23] http://www.gmail.com

[24] http://www.meebo.com

Los desarrolladores han implementado varias soluciones para este problema. Entre ellas está el utilizar IFRAMES invisibles para invocar cambios que alimenten el historial usado por el botón Atrás de un navegador. Por ejemplo Google Maps realiza búsquedas en un IFRAME invisible y luego jala los resultados en un elemento sobre la página web visible. Otro problema es que las actualizaciones de la página Web dinámica hace difícil para un usuario marcar como favorito un estado particular del aplicativo. Existen soluciones para este problema, muchas de las cuales, utilizan el identificador del fragmento URL (es decir, la porción del URL después del #) para poder guardar registro y permitir a los usuarios volver al aplicativo en un estado determinado. Esto es posible porque muchos navegadores permiten que JavaScript actualice dinámicamente, el identificador de fragmento de URL, para que aplicativos en Ajax puedan mantener ese identificador mientras que el usuario cambia el estado en el aplicativo. Esta solución también mejora el soporte del botón de atrás. Pero esta no es sin embargo, una solución completa.

Problemas con Tiempo de Respuesta

El intervalo entre la solicitud del usuario y la respuesta del servidor, necesita ser cuidadosamente analizado durante el desarrollo en Ajax. Sin una retroalimentación clara al usuario, predescarga inteligente (smart preloading) de datos y manipulación adecuada del objeto XMLHttpRequest, los usuarios pueden experimentar demoras que no esperaban o que no entenderán. Adicionalmente, cuando una página entera se abre, existe un instante de re-ajuste para el ojo, cuando el contenido cambia. El uso de retroalimentación visual (así como throbbers[25]) para alertar al usuario de la actividad que se está llevando a cabo en el “background” son con frecuencia sugeridos como

[25] Un throbber es una imagen normalmente localizada en la esquina superior derecha de la interfaz gráfica de usuario de un programa informático. Particularmente de los navegadores web muestran un throbber animado para informar al usuario de que el programa está realizando una acción (como descargar una página web).

soluciones para estos problemas de latencia en la red referente a la espera que realiza el usuario.

Optimización del Motor de Búsqueda

Los sitios Web que usan Ajax para cargar datos que deberían ser indexados por motores de búsqueda deben tener cuidado de proporcionar datos equivalentes en un link de URL público y en un formato que el motor de búsqueda pueda leer, ya que los motores de búsqueda, generalmente no ejecutan el código JavaScript requerido para la funcionalidad Ajax. Este problema NO es específico para Ajax, pues el mismo problema ocurre con sitios que proveen datos dinámicos mientras que una página completa se refresca como respuesta a una forma o información enviada (el problema general es llamado algunas veces la Web escondida o profunda “hidden or deep web”).

Dependencia en JavaScript

Ajax depende de JavaScript, que puede ser implementado de diferentes maneras por diferentes navegadores o versiones de un navegador específico. Debido a esto, es necesario probar sitios que usen JavaScript en múltiples navegadores para chequear compatibilidad. No es inusual ver código de JavaScript escrito dos veces, uno para Internet Explorer, otro para Mozilla, etcétera. Hoy, esto es menos cierto con la nueva versión de Internet Explorer 7 y con el uso más frecuente de librerías de abstracción de JavaScript como Prototype o Mootools. Estas librerías abstraen diferencias específicas para navegadores, y el desarrollador puede despreocuparse en cierta forma de realizar códigos independientes para las distintas versiones.

El nivel de soporte IDE para JavaScript es pobre, aunque esto ha cambiado con el uso mas frecuente de herramientas como Firebug, la barra de herramientas IE Developer, y Venkman.

El problema también surge cuando el usuario ha apagado el soporte de JavaScript en su navegador, desactivando así la funcionalidad de la página.

Web Analytics

Muchas soluciones de análisis de Web están basadas en el paradigma de cargar una nueva página cada vez que nuevo contenido va a ser presentado al usuario, o para mantener rastreados una serie de pasos tales como un check-out cuando se realiza una compra por Internet. Debido a que Ajax altera este proceso, se debe tener cuidado en cómo manejar una página o una porción de la misma para poder guardar registro de las acciones hechas en la página. Sistemas analíticos que permiten guardar registro de los eventos fuera de la simple visualización de la página, tales como el clic de un botón o de un link, son los que más probablemente pueden acomodarse en un sitio que utilice mucho Ajax.

7.1.6. Consideraciones finales

En este capitulo, se ha realizado un recorrido general de los aspectos más importantes de Ajax. Comenzamos describiendo su funcionamiento haciendo un paralelo con los modelos Web tradicionales, inspeccionamos más a fondo el concepto de la programación cometa y su importancia para lograr que las aplicaciones Web puedan basarse en eventos bidireccionales entre el cliente y el servidor para acabar con la restricción del protocolo Http referente a la incapacidad del servidor de enviar un evento sin una petición previa del navegador cliente, y por último, se describen algunas de las realidades a las que deben enfrentarse los desarrolladores cuando utilizan la tecnología Ajax.

Con toda esta información, es posible notar la importancia que tiene JavaScript para la implementación de las soluciones basadas en Ajax. Sin embargo, también es cierto que las dificultades que se presentan en el

momento del desarrollo para verificar las compatibilidades entre los navegadores modernos son bastante altas y se necesita de una basta experiencia en las particularidades de los navegadores o la utilización de las librerías de abstracción como las que se mencionaron previamente.

A parte de esto, si se concluye que la aplicación necesita tener respuestas en tiempo real, como en el caso del sistema que vamos a incorporar en EAFIT Interactiva, también es necesario definir cual de las aproximaciones es la mas apropiada para implementar el manejo de eventos entre el cliente y el servidor, de tal manera que no se infrinjan las restricciones de rendimiento requeridas para mantener el nivel de servicio de la plataforma.

También es necesario definir el lenguaje con el cual se enviarán las respuestas desde el servidor (recuérdese XML); para de acuerdo a esto realizar los desarrollos respectivos a nivel de cliente que interpreten el conjunto de mensajes.

Por último, es necesario utilizar los estándares HTML y CSS para la parte visual de la aplicación de mensajería instantánea, todo esto sin interferir en el diseño actual de EAFIT Interactiva.

Como lo hemos visto desde el principio, Ajax involucra un conjunto de tecnologías bastante amplias que deben tenerse en cuenta a la hora de realizar un desarrollo. Si bien esta es una realidad, existen alternativas para alivianar un poco la carga a la que se ven sometidos los desarrolladores. Dichas alternativas, lo que pretenden, es abstraer todos estos elementos e incorporarlos en una sola herramienta. Existen muchísimas alternativas, que combinan Ajax con diversos lenguajes a nivel de servidor, pero dado que como se enunció en los requisitos no funcionales de la aplicación, nuestra solución debe construirse en Java, entonces deben explorarse este tipo de alternativas.

El proceso de investigación nos lleva a seleccionar una herramienta donde todo el desarrollo se realiza en Java. Se trata de GWT (Google Web Toolkit), o Herramientas para la Web de Google. Gracias a este conjunto de herramientas, ya no es necesario conocer a profundidad todas las

tecnologías que se requieren para el desarrollo Ajax, y un programador Java habilidoso, teniendo conciencia de las arquitecturas cliente servidor, podría desarrollar aplicaciones grandiosas sin mucho conocimiento previo en tecnologías Web y con la confianza que los problemas de compatibilidad con navegadores, código HTML y CSS, manejo de respuestas y hasta soluciones limpias para aplicaciones basadas en eventos son contemplados. A continuación describiremos esta grandiosa herramienta y concluiremos con el modelo que se empleó para el sistema de MI de EAFIT INTERACTIVA.

8. HERRAMIENTAS PARA LA WEB DE GOOGLE (GWT)

Google[26] es el titán de las búsquedas en la Internet, y en la actualidad posee el mayor número de páginas indexadas por cualquier buscador. Sin embargo, paralelamente ha sido uno de los pioneros en el desarrollo de tecnologías Ajax. De hecho, el artículo que dio origen al término Ajax, tal como se describe en el capítulo 7, se basó en gran medida en la aparición de aplicaciones Web tan impresionantes como Google Maps y Gmail, el correo de Google.

La primera versión del GWT fue lanzada el 16 de Mayo de 2006. Google anunció el GWT en la conferencia JavaOne en el 2006.

Actualmente se encuentra en la versión 1.5, lanzada el 27 de agosto de 2008.

La motivación de la herramienta, es facilitar la construcción, la reutilización y el mantenimiento de códigos JavaScript largos y de componentes Ajax que generalmente pueden resultar difíciles y frágiles. GWT le permite a los desarrolladores Web construir y mantener aplicaciones complejas de alto desempeño en el lenguaje de programación Java.

8.1. Flujo del desarrollo

Editar código Java y ver los cambios inmediatamente sin recompilar

Durante el desarrollo, es posible ver los cambios inmediatamente sin necesidad de recompilar de nuevo la aplicación. Esta es una ventaja que le agrega el GWT al desarrollo Java tradicional, gracias a un servidor que tiene instalado para realizar pruebas. No es necesario compilar ni trasladar la aplicación a un servidor para poder probar.

---

[26] Información de Google puede encontrarse en http://www.google.com/intl/es/corporate/

Paso a paso en el código Ajax con el Debugger de Java

Cuando se va a pasar a producción, el código es compilado y transformado a JavaScript, pero en la fase del desarrollo todo el código corre en la maquina virtual de Java. Esto significa que los métodos de debugging tradicionales se aplican al código de GWT. Como ejemplo, podemos mencionar los tan útiles puntos de parada (breakpoints) y la ejecución paso a paso.

Compilación y despliegue optimizados, JavaScript multinavegador

Cuando el desarrollador esta listo para el despliegue, GWT compila el código Java y lo transforma en archivos JavaScript que pueden ser manejados por cualquier servidor. Además, las aplicaciones GWT automáticamente soportan Internet Explorer, Firefox, Mozilla, Safari y Opera sin ninguna operación de detección de código de navegador adherida al código. El desarrollador escribe el código una sola vez, y GWT lo transforma en el código JavaScript mas eficiente posible para cada navegador en particular.

8.2 Características

8.2.1 Comunicación con el servidor a través de RPC[27] simples

Los RPC del GWT convierten todas las comunicaciones Java en algo particularmente simple y eficiente. Simplemente se crea una interfaz que especificará los métodos remotos que podrán llamarse. Cuando se llama uno de esos métodos remotos del Servidor, GWT inmediatamente serializa[28] los

---

[27] Remote Procedure Call o Llamada a Procedimiento Remoto: es un protocolo que permite a un programa de ordenador ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos. El protocolo es un gran avance sobre los sockets usados hasta el momento. De esta manera el programador no tenía que estar pendiente de las comunicaciones, estando éstas encapsuladas dentro de las RPC.

Las RPC son muy utilizadas dentro del paradigma cliente-servidor. Siendo el cliente el que inicia el proceso solicitando al servidor que ejecute cierto procedimiento o función y enviando éste de vuelta el resultado de dicha operación al cliente.

[28] Serialización: Transmitir un objeto a través de una conexión de red en formato binario. Cuando llega a su destino, este código binario permite crear una copia exacta del objeto original.

argumentos, invoca el método apropiado en el servidor, luego deserializa el valor de retorno para el cliente.

8.2.2. Optimizar la descarga de JavaScript basado en perfiles de usuario

Aunque es posible descargar el código JavaScript para un navegador particular y de este modo evitar el sobre peso de la página incluyendo funciones innecesarios, GWT también permite que archivos diferentes a los del navegador, también sean parametrizables y se pueda controlar su descarga. Por ejemplo, si la aplicación soporta varios idiomas, GWT se encargará de verificar en el navegador del usuario qué leguaje se registra para de este modo cargar la aplicación en el lenguaje requerido, empleando solo el código fuente requerido para este idioma.

8.2.3. Reutilizar componentes de interfaz gráfica en varios proyectos

Es posible generar vistas de usuario particulares llamadas Widgets, que pueden ser integradas en varios proyectos, agilizando el desarrollo a través de la reutilización de código.

8.2.4. Utilizar librerías o código nativo de JavaScript

Pese a que GWT proporciona muchísimas herramientas para desarrollar las aplicaciones, es posible que se requiera un componente particular que debe implementarse usando otra Liberia de JavaScript o código nativo. GWT proporciona esta interfaz para facilitar este proceso.

8.2.5 Soporte del botón de atrás del navegador

Como se describió en la parte de los problemas de Ajax, GWT permite que el botón de atrás siga siendo válido aun en las aplicaciones Ajax.

8.2.6 Internacionalización

Permite que las aplicaciones desarrolladas en GWT tengan soporte a múltiples idiomas.

8.2.7 Productividad.

Gracias a que esta basado enteramente en Java, el desarrollador tendrá la opción de seleccionar su herramienta de trabajo favorita, como Eclipse o IntelliJ. Gracias a esto, la tarea de descubrir los errores en el código Ajax se hace muchísimo más rápida, pues estas herramientas poseen capacidades de detección de errores y ayudas casi en tiempo real.

8.3. Paquetes de GWT

Veamos una gráfica de la estructura de GWT (Figura 12)

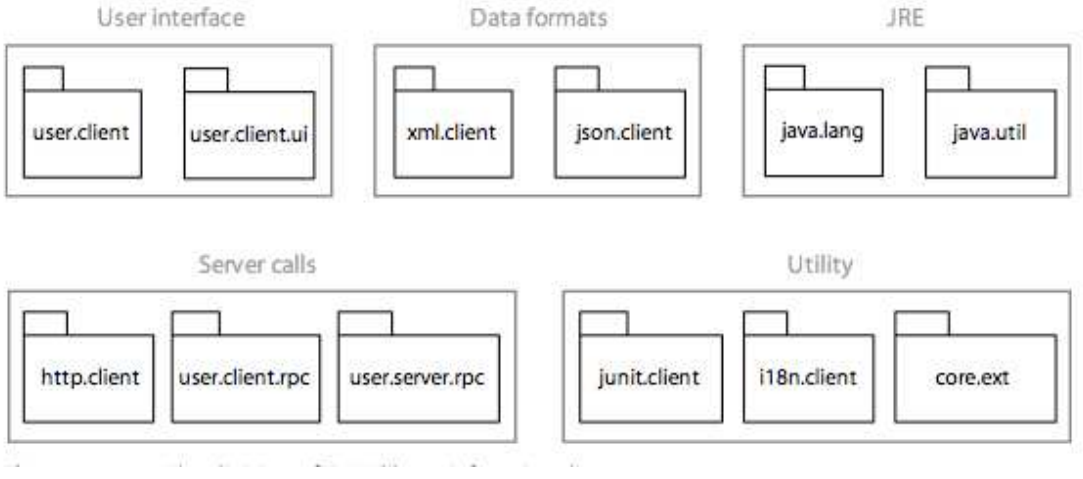


Figura 12: División de paquetes de GWT [4]

Los paquetes pueden dividirse en cinco categorías: Interfaz de usuario, llamadas al servidor, formato de datos, Emulación JRE, y Utilidades.

8.3.1. Interfaz de usuario

Es el paquete más grande y usado, y se encarga de construir y manejar una interfaz de usuario dinámica.

Contiene dos paquetes:

- **com.google.gwt.user.client**

  Se comunica con el los navegadores para construir las páginas dinámicamente.

- **com.google.gwt.user.client.ui**

  Entrega las herramientas necesarias para construir las interfaces de usuario similar al proporcionado por Java (AWT). Es muy útil para crear y usar componentes de interfaz dinámicos y reutilizables.

8.3.2. Llamadas al servidor

Contiene lo necesario para hacer llamadas y comunicaciones con el servidor.

Contiene tres paquetes:

- **com.google.gwt.http.client**

  Crea un poderoso objeto XMLHttpRequest independiente a las versiones de los navegadores.

- **com.google.gwt.user.client.rpc y com.google.gwt.user.server.rpc**

  Entrega la opción a las aplicaciones de comunicarse con el servidor a través de llamadas a procedimientos remotos (RPC). Recordemos que esto permite comunicarnos con el servidor sin preocuparnos por los

detalles del protocolo. También se apoya en el objeto XMLHttpRequest para su comunicación. Esta será la herramienta a utilizar en el sistema de MI.

8.3.3. Formato de datos

Permite la construcción y el procesamiento de formatos de datos (XML y JASON).

- **com.google.gwt.xml.client**

  Permite procesar documentos XML e iterar sobre sus contenidos. También permite construir documentos XML. Utilizando este paquete con el de HTTP permite enviar y recibir documentos XML a y desde el servidor.

- **com.google.gwt.json.client**

  El paquete JSON es similar al paquete XML, excepto que el formato manejado es JSON en vez de XML. JSON es un formato jerárquico similar al XML pero mucho más liviano en peso.

8.3.4. Emulación

Este paquete contiene un subconjunto de la librería de Java para asistir en tareas comunes de programación incluyendo tipos básicos de Java, excepciones y colecciones. Se considera una emulación, porque las clases de Java no son usadas cuando una aplicación GWT es desplegada. En vez de esto, el compilador reemplaza estas clases con código JavaScript equivalente.

- **java.lang**

Contiene clases básicas de Java como Integer y String.

- **java.util**

  Provee muchas de las colecciones de Java, incluyendo mapas (maps), listas (lists) y conjuntos (sets).

8.3.5. Utilidades

Contiene utilidades muy importantes como la internacionalización, las unidades de prueba y las clases para extender el compilador del GWT. Estos paquetes son: com.google.gwt.i18n.client, com.google.gwt.junit.client, com.google.gwt.core.ext.

Aunque existen muchas librerías de JavaScript excelentes para ayudar a construir aplicaciones Ajax, GWT se diferencia de todas ellas en el sentido de entregar un conjunto de herramienta de ingeniera de software de Java para ser usadas en aplicaciones Ajax, en vez de ser una librería llena de funcionalidades preconstruidas. Sin duda, constituye una excelente elección para el desarrollo de aplicaciones web robustas, mantenibles y con la confiabilidad que han mostrado todos los productos de Google.

Con toda la información proporcionada hasta el momento, es posible comenzar a enunciar los detalles técnicos de la solución al problema de la mensajería instantánea para la plataforma de EAFIT Interactiva.

9. EL SISTEMA

9.1. El patrón de colaboración

Para construir una aplicación usando el patrón de Colaboración es necesario definir un modelo centralizado al que se accede a través de un controlador como se muestra en la figura 13. Cada cliente colaborativo interactúa con el controlador para operar el modelo. El controlador es el responsable de informar al resto de clientes cualquier cambio sobre el modelo.

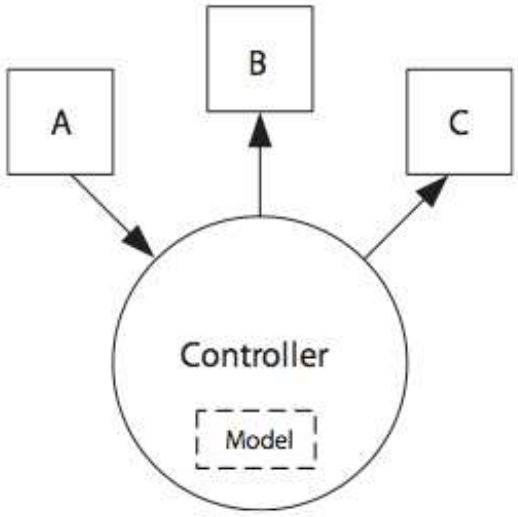


Figura 13: El patrón de colaboración. [4]

Como se ha dicho reiteradamente, en los sistemas de mensajería instantánea los clientes no se conectan directamente entre ellos, sino que lo hacen a través de un servidor. Esto es precisamente lo que hará la aplicación para la plataforma de EAFIT Interactiva (Figura 14).

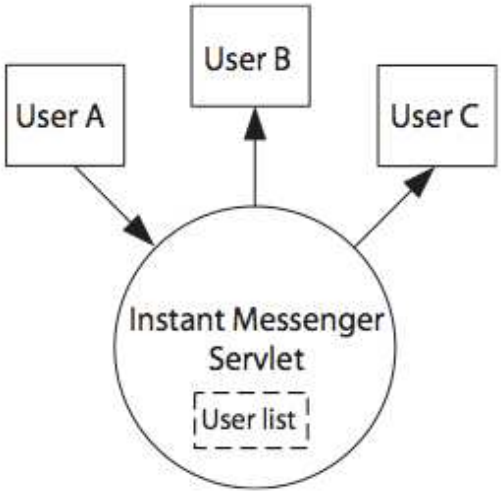


Figura 14: Patrón de colaboración aplicado a la mensajería instantánea. [4]

El servidor de MI correrá como un servlet GWT con una interfaz GWT-RPC. Los clientes se conectarán y harán llamadas a los métodos del servlet para soportar la funcionalidad del sistema de MI. El servlet implementará la interfaz RPC y se encargará de hacer la propagación de los eventos a cada cliente. Lo anterior será descrito con mas profundidad en la sección 9.2.4.

La aplicación se integrará en la página Web de EAFIT Interactiva como un widget[29] embebido que complementará el funcionamiento normal de la plataforma con el servicio de presencia y la comunicación instantánea entre los usuarios.

9.2 MVC (Modelo-Vista-Controlador)

El diseño interno de la aplicación sigue el patrón de arquitectura Modelo-Vista-Controlador. Este patrón separa la lógica del negocio de las consideraciones de interfaz de usuario, posibilitando una aplicación donde es mucho más fácil modificar bien sea la apariencia visual o las reglas de negocio sin ningún tipo de conflicto entre interfaz y negocio.

[29] Componente de interfaz gráfica de usuario con el cual un usuario interactúa.

Por lo general, el control de flujo en una aplicación MVC funciona de la siguiente manera:

1. El usuario interactúa con la interfaz de usuario de alguna manera.
2. Un controlador maneja el evento de entrada de la interfaz de usuario, generalmente a través de un manejador previamente registrado para el evento en particular.
3. El controlador notifica al modelo la acción del usuario, resultando posiblemente en un cambio de estado dentro del modelo.
4. Una vista usa el modelo (indirectamente) para generar una interfaz de usuario apropiada. La vista obtiene los datos del modelo. El modelo no tiene conocimiento directo de la vista.
5. La interfaz de usuario espera por nuevas interacciones y se repite el ciclo.

Para el caso del sistema de MI, el modelo representa los objetos en los cuales la vista opera, incluyendo los contactos, la lista de contactos y los mensajes. La vista representa el objeto gráfico de la lista de contactos y otras ventanas requeridas (ventanas de conversaciones individuales) para el sistema de MI dentro de la página. El controlador implementa la comunicación con el servidor y maneja los objetos del modelo para ser desplegados en la vista.

9.2.1 El modelo

El modelo de la aplicación de MI tiene solo tres clases: ListaContacto, Contacto y Mensaje. Estas clases son muy simples y podría sugerirse que no son necesarias en el sistema, pero en caso de que vaya a ampliarse alguna funcionalidad es muy importante el orden y tener separados los componentes de la aplicación.

Otro beneficio de tener el modelo para la aplicación, es que cuando se usa GWT para codificar el lado del cliente podemos compartir los objetos del modelo con código del servidor. Gracias a que implementaremos el código

del servidor como un servlet, su desarrollo también se hace en el lenguaje Java. Es importante recordar en este momento que GWT es un conjunto de herramientas desarrolladas en el lenguaje de programación Java, y puede usarse, bien sea para la construcción de componentes para sitios Web sin interacción con el servidor (solo del lado del cliente), o por el contrario, como una aplicación completa que realice operaciones en el servidor y en el cliente conjuntamente. Para nuestro caso, utilizaremos GWT-RPC que nos permite transferir objetos de Java entre el cliente y el servidor, lo que posibilita que usemos los objetos del modelo para la construcción del protocolo de datos.

Puede verse una representación básica del modelo en la figura 15:

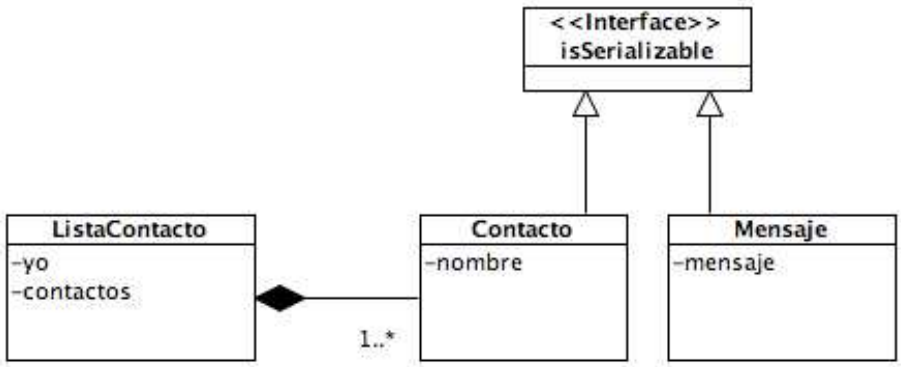


Figura 15: Modelo de la aplicación [Traducido de 4]

La clase ListaContacto tiene un atributo “yo” que hace referencia a una instancia de Contacto que representa al usuario actual. Tiene también una lista de contactos. La clase Contacto tiene el nombre del contacto, y la clase Mensaje tiene básicamente un atributo mensaje que representa el cuerpo del mensaje. Es muy importante notar, que tanto Contacto como Mensaje, implementan la interfaz de GWT “IsSerializable”. Implementar esta interfaz solo implica la creación de un constructor sin argumentos en cada una de las clases, gracias al cual las instancias de las clases pueden ser transmitidas a través del protocolo GWT-RPC. Recordemos entonces que gracias al

modelo, cualquier cambio en estas clases estará disponible en el controlador, en el servidor (gracias al RPC) y en la vista.

9.2.2 La vista: Construyendo una interfaz complementaria

Como se dijo en el capitulo seis, los requisitos nos llevan a desarrollar una interfaz que no interactúa con el escritorio del usuario, o con una barra de herramientas, sino que debe compartir espacio con otra aplicación en una ventana de navegador. Por esto, debe seleccionarse una ubicación apropiada para que el sistema de MI no interfiera con el uso principal de la plataforma. Para este fin, el administrador de EAFIT Interactiva debe crear un elemento HTML con un id especifico, que luego será utilizado para informarle a nuestro sistema donde deberá ubicar la vista. Por fortuna, en la versión actual de EAFIT Interactiva existe el espacio apropiado para nuestra vista, pues ya hay una sección dedicada a mostrar los usuarios que se encuentran conectados. En este espacio, será mostrada entonces la información de los contactos que se encuentran en línea y tienen disponibilidad para recibir mensajes instantáneos. Sin embargo, también es necesario contar con una vista encargada de enviar mensajes y notificar al usuario sobre la llegada de nuevas comunicaciones. Para este fin, se utilizarán ventanas emergentes, ubicadas en el extremo inferior derecho de la pantalla cuando lleguen mensajes nuevos, sin que ésta interfiera con la actividad que el usuario esté realizando. La misma ventana aparecerá cuando el usuario decida iniciar una conversación haciendo clic en alguno de los contactos en línea (Figura 16).

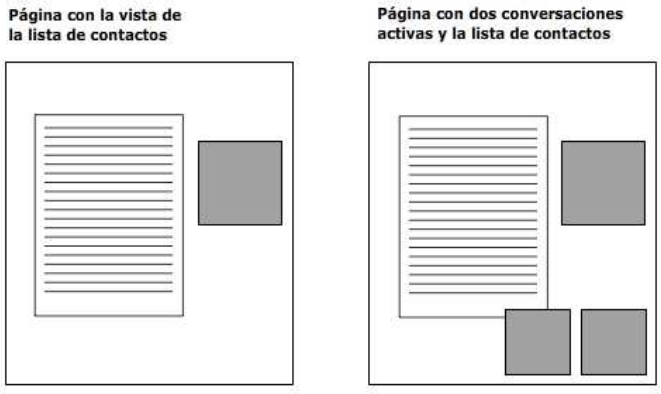


Figura 16: Vistas de la aplicación. [Construida a partir de 4]

Para ofrecer esta funcionalidad, necesitamos cuatro clases como se ve en la figura 17.

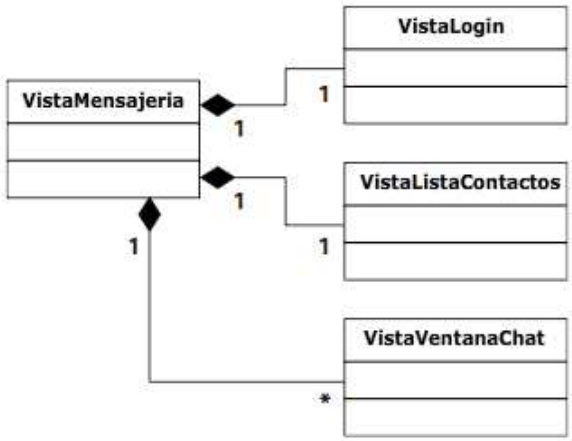


Figura 17: Clases que implementan la vista de la aplicación [Traducido de 4]

La clase VistaMensajeria maneja todas las vistas de la aplicación, la clase VistaLogin muestra un mensaje que notifica al usuario que se está ingresando al sistema de MI, la clase VistaListaContactos muestra una lista de Contactos en la página que le permite al usuario abrir ventanas de conversación con cada uno. La clase VistaVentanaChat genera las ventanas que se ubican en la parte inferior derecha de la pantalla, donde además se van mostrando la lista de mensajes entre dos contactos y les permite a los usuarios enviar mensajes.

La clase VistaMensajeria es muy importante. Gracias a ella entenderemos muchas de las ventajas que ofrece el GWT. A continuación veremos una porción de código con la cual se explicaran varias cosas.

```
public class VistaMensajeria
   implements WindowCloseListener, WindowResizeListener{
   private DeckPanel mainPanel = new DeckPanel();
   private VistaListaContactos vistaListaContactos;
   private VistaLogin vistaLogin;
   private ListaContacto listaContacto;
   private List ventanasAbiertas = new ArrayList();
   private Map todasVentanas = new HashMap();
   private VistaMensajeriaListener listener;
   public VistaMensajeria( VistaMensajeriaListener listener ){
      initWidget( mainPanel );
      this.listener = listener;
      vistaLogin = new VistaLogin ( this );
```

```
        mainPanel.add(vistaLogin);
        mainPanel.showWidget(0);
        Window.addWindowCloseListener(this);
        Window.addWindowResizeListener(this);
    }
```

Como puede verse, la clase implementa las interfaces WindowCloseListener y WindowResizeListener. Estas dos interfaces son requeridas por la clase para implementar métodos que recibirán eventos desde el navegador para manejar los eventos de cambio de tamaño de la ventana, o cierre de la ventana. Gracias a estas interfaces, podemos recibir estos eventos desde cualquier navegador, sin importar su naturaleza, y como puede verse, lo único que tenemos que hacer es programar en Java. En el momento de la compilación, GWT transformará este código en JavaScript, cuidando los detalles y las diferencias entre todos los navegadores, para que la captura y el manejo de eventos arrojen los mismos resultados. El evento de cambio de tamaño de la ventana, sirve para reordenar las ventanas abiertas en caso de que el usuario le cambie el tamaño a la ventana. Esto tiene sentido si recordamos que la ubicación de las ventanas siempre debe ser la parte inferior derecha de la página. Por su parte, capturar el evento de cierre de la ventana, que en términos reales implica, bien sea una actualización de la página, o cerrarla completamente, sirve para realizar la operación de logout en el servidor.

Continuando, vemos también un atributo privado que instancia una clase DeckPanel. Esta clase, es una de las muchas que contiene GWT para el desarrollo de interfaces Web. Para este caso, el DeckPanel es un widget que puede contener varios widgets, pero mostrar solo uno a la vez. Es útil para cambiar, en nuestra aplicación, de la vista de login, hacia la vista de la lista de contactos. En la figura 18 pueden verse todas las clases, por categoría, que nos ofrece el GWT para la construcción de interfaces gráficas en la Web. Estas clases reciben el nombre de Widgets.

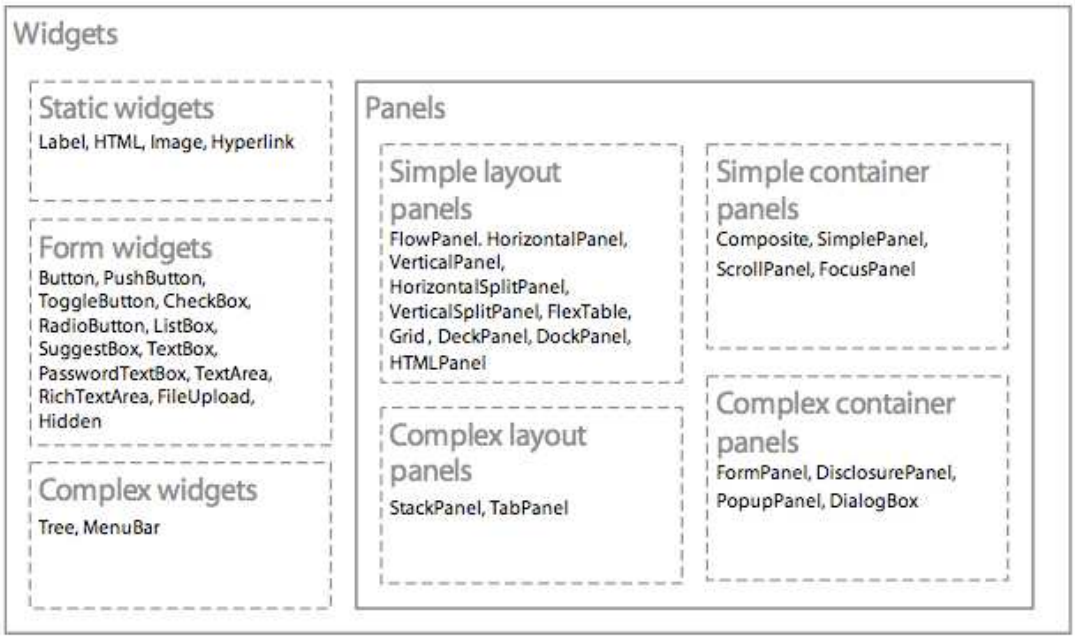


Figura 18: Lista de widgets de GWT por categoría. [4]

De nuevo otra de las ventajas. No tenemos que conocer la sintaxis de HTML para construir nuestras aplicaciones. Mas aun, en algunos casos, muchos de los Widgets de GWT necesitan de la utilización en conjunto de HTML, CSS y JavaScript si se deseara crearlos de forma manual. Sin embargo, aunque GWT encapsula todos estos componentes, permite que los desarrolladores experimentados definan CSS, funcionalidades JavaScript que no posee el toolkit o que construyan sus propios Widgets, entre otros.

Otro elemento vital que podemos encontrar en esta pequeña porción de código es un atributo llamado listener de tipo VistaMensajeriaListener. Puede verse que una referencia a la instancia de VistaMensajeriaListener es necesaria para construir la clase VistaMensajeria, y el controlador implementa esta clase para recibir los eventos de la interfaz. La siguiente es la interfaz VistaMensajeriaListener:

```
public interface VistaMensajeriaListener {
   void onLogIn();
   void onLogOut();
   void onEnvioMensaje( Contacto para, Mensaje mensaje );
}
```

El controlador de la aplicación debe implementar el método onLogIn para manejar cuando un usuario llega a una materia para que sea registrado en el Servidor, el método onLogOut para eliminar al usuario de los usuarios activos cuando deja la página o una materia y onEnvioMensaje para manejar el envío de un mensaje por parte de un usuario a otro contacto (Más detalles en la sección 9.X).

De aquí podemos entonces concluir, que gracias al Listener, la vista se comunica con el controlador, para que éste a su vez, realice las operaciones necesarias en el servidor. En el siguiente código, estamos implementando uno de los métodos de la interfaz WindowCloseListener.

```
public String onWindowClosing(){
   listener.onLogOut();
   return null;
}
```

Obsérvese, que la vista, lo único que dice, es que al cerrar la página, debe ejecutarse el método onLogOut que es implementado por el controlador para acceder al servidor.

Por último, la clase VistaMensajeria también incluye un atributo de tipo ListaContacto con lo que puede notarse su interacción con el modelo.

Dado que en el resto de clases solo se utilizan un conjunto de Widgets para construir las interfaces de la aplicación, continuaremos con el controlador, otro de los elementos importantes del sistema de MI.

9.2.3 El controlador

El controlador de la aplicación de MI tiene el trabajo de recibir eventos de la vista, comunicarse con el servidor y operar en los objetos del modelo de la aplicación. La interacción con la vista y el modelo, ya la hemos visto de forma parcial cuando hablamos del listener que debía ser implementado por el controlador. La vista solo sabe que existe una clase proporcionada por el controlador para hacer las comunicaciones con el servidor (sección 9.2.2). Precisamente la parte mas compleja de implementar en el controlador será la creación de la interfaz RPC para comunicarse con el servidor de MI. Adicional

a esto, tenemos que crear un protocolo basado en eventos que esta por encima de la tecnología que estamos empleando pues ésta no soporta directamente éste tipo de protocolos. Afortunadamente, usaremos la implementación de GWT-RPC, que nos ahorra mucho tiempo liberándonos de la necesidad de desarrollar el protocolo base de comunicación con el servidor que resulta ser bastante intuitivo para los desarrolladores de JAVA.

Para comenzar a entender el controlador, observemos la secuencia de eventos que maneja durante el tiempo de vida de la aplicación. El controlador inicia la aplicación construyendo las vistas principales (a través de la clase VistaMensajería). En ese momento el controlador va al servidor para registrar al usuario nuevo a través de unas variables de sesión que crea la plataforma de EAFIT Interactiva cada vez que un usuario ingresa. De esta forma, el servidor puede enviarle a todos los clientes conectados de una materia en particular que éste usuario ahora está en línea. Cuando se completa éste paso, el controlador le informa a la vista y esta a su vez muestra la lista de contactos. En este momento, el controlador queda listo para aceptar eventos del servidor a medida que usuarios se conectan o salen de una materia en específico, entonces, recibe los eventos y los procesa para ser reflejados en la vista. Como se sabe, el usuario también puede enviar mensajes. En este caso, el controlador recibe el evento de la vista de la aplicación y transmite el mensaje al servidor, el cual lo envía luego a su destinatario. Lo contrario ocurre cuando un cliente envía un mensaje al usuario actual: el controlador recibe el mensaje como un evento y lo pasa a la vista para que se refleje en la interfaz del usuario.

Implementar el controlador puede dividirse en tres partes:

1. La inicialización, donde se crea la conexión con el servidor y se inserta la vista en el documento HTML de EAFIT Interactiva.

2. La implementación de la interfaz RPC del lado del cliente para las comunicaciones remotas en respuesta a eventos de la vista.

3. La implementación del lado del servidor de la interfaz RPC, que maneja las conexiones de muchos clientes y trasmite los eventos.

La estructura de clases para el controlador es muy sencilla pero requiere algún conocimiento de RPC para ser entendida en su totalidad. Antes de hablar detalladamente de RPC, observemos un diagrama de las clases del controlador en la figura 19.

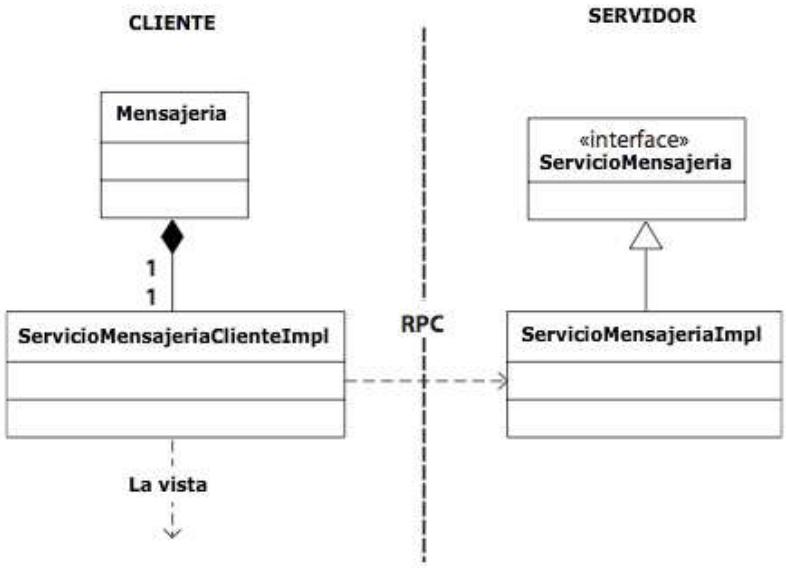


Figura 19: Las clases involucradas en el controlador de la aplicación. [Traducido de 4]

La clase mensajería simplemente crea una instancia de la clase ServicioMensajeriaClienteImpl e inserta la vista dentro de la página de EAFIT Interactiva usando el elemento HTML que se creó para este fin:

```
public class Mensajeria implements EntryPoint{
   public void onModuleLoad() {
 ServicioMensajeriaClienteImpl servicioMensajeria =
 new M ServicioMensajeriaClienteImpl
(GWT.getModuleBaseURL()+"mensajeria");
      RootPanel.get("id_del_elemento_html").add(
servicioMensajeria.getVista() );

   }
}
```

Ésta clase implementa la interfaz GWT EntryPoint y su método onModuleLoad. Cuando GWT carga la aplicación, este método es llamado de primero. La instancia de ServicioMensajeriaClienteImpl tiene el trabajo de interactuar con el servidor de MI y con la vista de la aplicación. La comunicación con el servidor es ejecutada con la librería GWT-RPC. La instancia de VistaMensajeria, responsable de manejar la vista de la aplicación

(sección 9.2.2), es instanciada como un miembro de la clase ServicioMensajeriaClienteImpl. Entonces, la vista es traída de la instancia de ServicioMensajeriaClienteImpl e insertada en la página de EAFIT Interactiva utilizando la clase de GWT RootPanel (donde se especifica el id del elemento HTML que alojará el sistema de MI).

9.2.4 GWT-RPC

GWT ofrece muchas herramientas para hacer llamadas al servidor. Todas ellas siguen los estándares tecnológicos de la Web y son lo suficientemente abiertas como para que el desarrollador decida la forma en que puede usarlas. GWT no pretende prescribir como usar las tecnologías que están disponibles en el navegador, pero GWT proporciona RPC (llamadas de procedimientos remotos) como una tecnología adicional para comunicaciones de red. Construida con las funcionalidades estándar de los navegadores, entrega un alto nivel de abstracción que nos aleja de la complejidad de los protocolos de red y que es exclusivo de GWT. El logro de las implementaciones de RPC es proveer métodos que se llaman en el cliente, los cuales a su vez llaman un método similar en el servidor. Esto facilita tanto las comunicaciones de red que los detalles del protocolo permanecen ocultos a los desarrolladores. Como los desarrolladores están acostumbrados a llamar métodos en su código, no tienen que aprender una tecnología nueva y no incurrirán en ninguno de los errores que podrían cometerse si usaran un protocolo directamente.

La implementación de GWT-RPC funciona generando un objeto Proxy automáticamente para una interfaz en el servidor. La aplicación en el cliente usa este objeto Proxy para comunicarse con el servidor a través del llamado de métodos. El servidor maneja las llamadas del Proxy y las envía a las implementaciones de los métodos Java correspondientes. Cualquier valor de retorno es enviado de regreso al Proxy del cliente. La aplicación en el cliente que hace la llamada entrega al Proxy un objeto llamado callback que recibe cualquier valor de retorno o mensajes de error del servidor. La figura 20 muestra como funciona GWT-RPC.

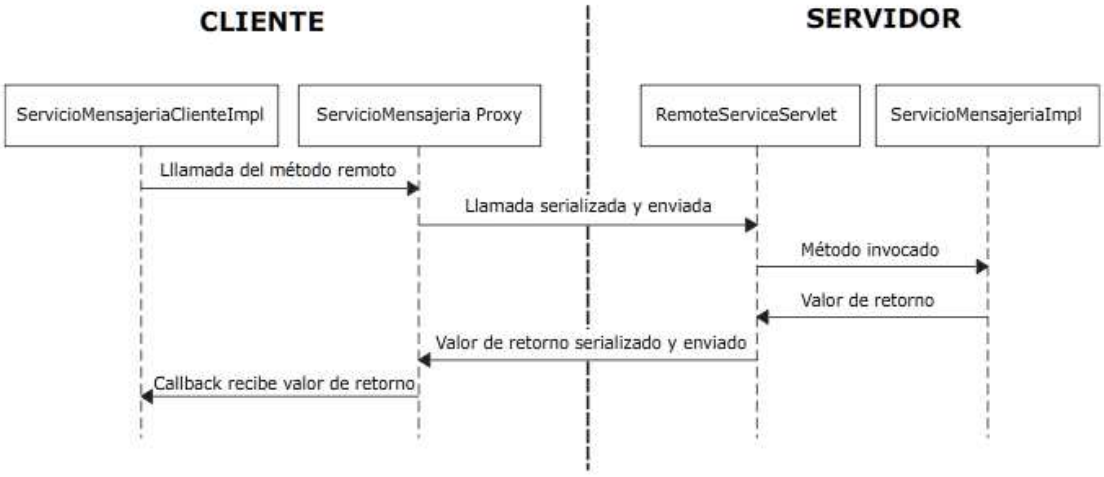


Figura 20: Cómo funciona GWT-RPC [Traducido de 4]

El ServicioMensajeriaClienteImpl es nuestro código controlador que hace llamadas de métodos RPC en el servidor y maneja los resultados en un callback. El Proxy de SerivicioMensajeria es un objeto que GWT genera para implementar una versión asincrónica de la interfaz ServicioMensajeria y también la interfaz de GWT llamada ServiceDefTarget (Mas adelante se describe la versión asincrónica de la interfaz ServicioMensajeria, llamada ServicioMensajeriaAsync). En el servidor, una instancia de la clase de GWT llamada RemoteServiceServlet maneja las peticiones serializadas e invoca el método requerido en la instancia de ServicioMensajeriaImpl que usa la clase RemoteServiceServlet como su súper clase. En la figura 21 se muestran las relaciones que existen entre interfaces y clases en el sistema RPC.

En el lado del cliente, el objeto de Proxy generado implementa las interfaces ServicioMensajeriaAsync y ServiceDefTarget. En el lado del servidor, la instancia de ServicioMensajeriaImpl implementa la interfaz ServicioMensajeria y hereda de la clase GWT RemoteServiceServlet.

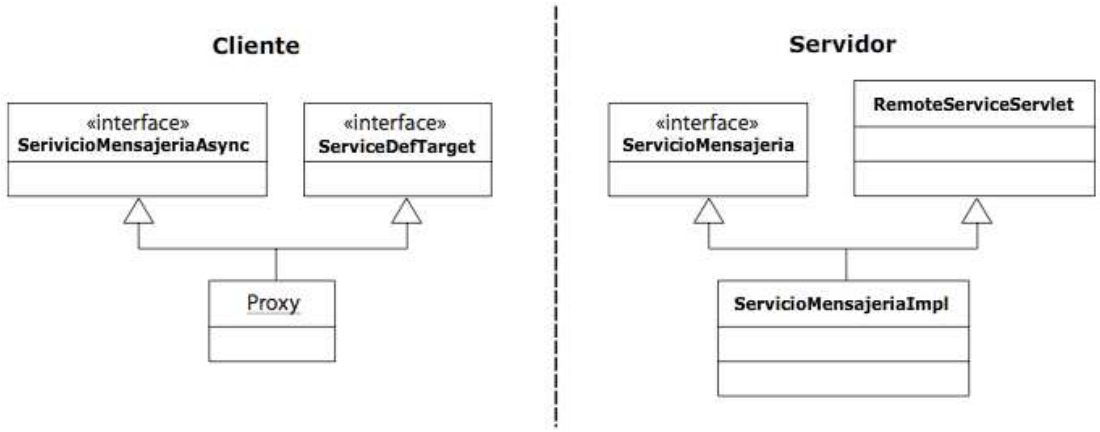


Figura 21: Relación entre interfaces RPC y clases [Traducido de 4]

9.2.4.1 La interfaz RemoteService

SerivicioMensajeria es una interfaz de Java tradicional que define los métodos que vamos a usar para las comunicaciones remotas con el servidor. La siguiente es la definición de la interfaz:

```
public interface ServicioMensajeria extends RemoteService {
   void logIn();
   void logOut();
   void enviarMensaje( Contacto para, Mensaje m );
}
```

Las interfaces remotas GWT deben heredar de la interfaz GWT llamada RemoteService. Ésta es una interfaz vacía que simplemente le informa al compilador GWT que la interfaz ServicioMensajeria será usada para comunicaciones RPC.

Para nuestro caso, vemos que la interfaz define tres métodos que serán usados como la firma de tres diferentes llamadas al servidor. El controlador llamará logIn(), cuando el usuario ingrese a alguna de las materias que tiene activas. El servidor implementará éste método y cuando se invoque, anunciará a los demás clientes que el usuario se ha conectado. El segundo método, logOut(), realiza la operación contraria. Cuando un usuario sale de la materia o de la plataforma, todos los demás usuarios son notificados de esto. El tercer método, enviarMensaje(Contacto para, Mensaje m), será implementado en el servidor para enviar un mensaje al contacto especificado.

Puede verse que todos los métodos reciben parámetros regulares de Java, bien sean objetos o valores primitivos. Esta es una de las características más atractivas de GWT-RPC, podemos enviar objetos Java y la traducción de estos a un formato apropiado para la Web es realizada por GWT. Para enviar objetos Java, lo único que debe tenerse en cuenta, es que las clases que queramos enviar deben implementar la interfaz IsSerializable de GWT o Serializable de Java con un constructor sin argumentos. Por esta razón, las clases del modelo Contacto y Mensaje, implementan la interfaz IsSerializable y ambas poseen un constructor sin argumentos.

9.2.4.2 La clase RemoteServiceServlet

Una clase del lado del servidor debe implementar la interfaz ServicioMensajeria descrita en la sección 9.2.4.1. En el caso del sistema de MI, tenemos un servlet llamado ServicioMensajeriaImpl que implementa la interfaz ServicioMensajeria, y en vez de heredar de la clase Java HTTPServlet para obtener la funcionalidad de ejecutarse en un contenedor de servlets, hereda de la clase GWT RemoteServiceServlet. Ésta clase hace algún trabajo adicional en el sentido de traducir las llamadas RPC de un cliente para invocar las implementaciones del método en la clase del servidor. Posibilita además que la implementación de la interfaz ServicioMensajeria en el servidor sea bastante limpia sin ningún tipo de código RPC. A continuación se muestra una implementación vacía de RPC del lado del servidor:

```
public class ServicioMensajeriaImpl
extends RemoteServiceServlet implements ServicioMensajeria {
   public void logIn(){
   }
   public void logOut(){
   }
   public void enviarMensaje( Contacto para, Mensaje m ){
   }
}
```

Cuando ésta clase es desplegada en un contenedor de servlets, queda lista para recibir peticiones GWT-RPC.

9.2.4.3 La interfaz asincrónica

Existe una limitante por la cual no podemos usar la interfaz ServicioMensajeria para hacer nuestros llamados al servidor desde el cliente. Si la utilizáramos, todas las peticiones al servidor bloquearían el único hilo usado por JavaScript en el navegador y la aplicación aparecería como si estuviera congelada. Además, es necesario tener un objeto callback para informarle al cliente si el método falló o fue exitoso con su respectivo valor de retorno en caso de tenerlo. Entonces, la interfaz asincrónica, es idéntica a la interfaz ServicioMensajeria exceptuando el hecho de que ninguno de los métodos tiene un valor de retorno y deben aceptar como último parámetro un objeto callback. Cuando el cliente hace una llamada al servidor sobre uno de los métodos de la interfaz asincrónica, el método retorna inmediatamente. El valor de retorno es pasado como un objeto al callback. Para implementar un callback para un método RPC es necesario hacer una implementación de la interfaz GWT llamada AsyncCallback:

```
public interface AsyncCallback {
   void onFailure(Throwable caught);
   void onSuccess(Object resultado);
}
```

El parámetro del método onSuccess representa la respuesta del método llamado. El método onFailure recibe una excepción lanzada bien sea por GWT debido a un error de comunicación o desde el código en nuestro servidor.

Así se ve la implementación de la interfaz Asincrónica ServicioMensajeriaAsync, obsérvese que reciben un parámetro adicional de tipo AsyncCallback y todos los métodos son void:

```
public interface ServicioMensajeriaAsync {
   void logIn( AsyncCallback callback );
   void logOut( AsyncCallback callback );
   void enviarMensaje(Contacto para, Mensaje mensaje, AsyncCallback
callback );
}
```

El cliente puede usar esta interfaz para hacer llamadas al servidor, pero antes, debe crearse el objeto Proxy que correrá en el cliente y traducirá

automáticamente estas llamadas a peticiones HTTP para enviarlas al servidor. Un objeto Proxy puede ser creado de la siguiente manera:

```
ServicioMensajeriaAsync servicioMensajeria =
(ServicioMensajeriaAsync) GWT.create( ServicioMensajeria.class );
```

De esta forma, GWT se da cuenta que ServicioMensajeria está implementando la clase RemoteService, entonces sabe que debe crear una instancia del Proxy implementando la interfaz ServicioMensajeriaAsync. En este punto, el Proxy no esta conectado al servidor, y aun no se le ha especificado la URL del servlet. Para conectarlo con el servidor, basta con las siguientes líneas:

```
ServiceDefTarget endpoint = (ServiceDefTarget) servicioMensajeria;
endpoint.setServiceEntryPoint( GWT.getModuleBaseURL() + "mensajeria"
);
```

Como se vio en la figura 21, el Proxy también implementa la interfaz ServiceDefTarget, que provee un método para establecer la URL remota del servlet. Una vez que la URL esta establecida, el cliente puede hacer llamadas al servidor usando la interfaz asincrónica.

9.2.4.4 La conexión con el servidor

La clase ServicioMensajeriaClienteImpl maneja casi todo el trabajo del controlador. La clase maneja la interacción con las vistas y hace las llamadas al servidor a través de GWT-RPC. Para interactuar con la vista implementa la interfaz VistaMensajeriaListener y sus correspondientes métodos. También crea una instancia de VistaMensajeria y guarda su referencia. Veamos el siguiente cuadro donde se explican algunos elementos importantes que faltan para entender el funcionamiento de GWT-RPC en su totalidad:

Tabla 3: La clase ServicioMensajeriaClienteImpl

`public class ServicioMensajeriaClienteImpl implements VistaMensajeriaListener{`	La clase implementa la interfaz VistaMensajeriaListener para comunicarse con la vista

`private ServicioMensajeriaAsync servicioMensajeria;`
`private ListaContactos listaContactos;`
`private vistaMensajeria vista = new vistaMensajeria( this );`
	Se crean instancias de la interfaz asincrónica, de una clase del modelo (ListaContactos) y de la vista (VistaMensajeria)

`public ServicioMensajeriaClienteImpl( String url ){`
`servicioMensajeria = (ServicioMensajeriaAsync) GWT.create( ServicioMensajeria.class );`
`ServiceDefTarget endpoint = (ServiceDefTarget) servicioMensajeria;`
`endpoint.setServiceEntryPoint( url );`
`}`
	El constructor de la clase, recibe la URL del servlet y crea el objeto Proxy del cliente, creando posteriormente la conexión con el servidor gracias a la dirección del servlet que se recibe como parámetro.

`Public VistaMensajeria getVista(){`
`return vista;`
`}`
	Un método get que retorna la referencia a la clase VistaMensajeria.

`public void onLogIn(){`
`servicioMensajeria.logIn(new LogInCallback() );`
`}`
	Uno de los métodos de la interfaz VistaMensajeriaListener. Puede verse la ejecución de un método en el servidor con la instancia de la interfaz asincrónica. Recibe un objeto AsyncCallback.

`public void onLogOut(){`
`servicioMensajeria.logOut(new EmptyCallback());`
`}`
	Otra implementación de la interfaz VistaMensajeriaListener. También recibe un objeto AsyncCallback para hacer el llamado al servidor.

`public void onEnvioMensaje( Contacto para, Mensaje m ){`
`servicioMensajeria.enviarMensaje( para, m, new EmptyCallback());`
`}`
	Otra muestra de cómo la interfaz interactúa con el servidor a través del controlador. Esta vez ejecutando el método en el servidor para el envío de mensajes.

`Private class LogInCallback implements AsyncCallback{`
`public void onFailure(Throwable throwable){`
`GWT.log("error al login",throwable);`
`}`
`public void onSuccess(Object obj){`
`vista.setListaContactos( listaContactos );`
`}`
`}`
	Implementación de una clase AsyncCallback. Los dos métodos ya estudiados. Obsérvese como en el método OnSuccess se realizan operaciones directamente sobre la vista, gracias a una respuesta del servidor.

Recordemos la clase Mensajería que analizamos en el inicio de la sección 9.2.3:

```
public class Mensajeria implements EntryPoint{
   public void onModuleLoad() {
 ServicioMensajeriaClienteImpl servicioMensajeria =
 new                                  ServicioMensajeriaClienteImpl
(GWT.getModuleBaseURL()+"mensajeria");
      RootPanel.get("id_del_elemento_html").add(
servicioMensajeria.getVista() );

   }
}
```

Recordemos que ésta clase es la que llama GWT al inicio de la ejecución. Obsérvese que se crea la instancia de la clase ServicioMensajeriaClienteImpl, se pasa como parámetro la URL del servlet que se definió en la configuración del contenedor de Servlets, y luego se utiliza su instancia para invocar la vista, gracias a la referencia que tiene esta clase a la instancia de VistaMensajeria.

Nuestro sistema se encuentra en este punto casi listo. Falta solo un elemento para terminar el engranaje definitivo. Se trata de añadir a esta estructura RPC, la posibilidad de manejar eventos. Recordemos, que como se discutió en la sección de AJAX, el hecho de realizar esta aproximación constituye una ganancia innegable en términos de rendimiento, y formaliza una de las soluciones mas sofisticadas para entregar a las aplicaciones la habilidad de respuestas en tiempo real.

9.3 Agregando Eventos RPC

La mensajería instantánea requiere un protocolo basado en eventos. Esto es algo sencillo si se piensa en una aplicación de escritorio pues estas están basadas en los sockets de TCP/IP, pero las aplicaciones Ajax recaen en HTTP, el cual no soporta la propagación de eventos. HTTP solo soporta enviar respuestas a un cliente cuando éste ha hecho una petición. Es decir, si no hay petición, no se puede enviar la respuesta.

Como se vio en las secciones dedicadas a Ajax (7.1 en adelante), las aplicaciones Ajax pueden comunicarse de forma asincrónica con el servidor utilizando una facilidad del navegador llamada XMLHttpRequest. Por desgracia, este objeto solo soporta el protocolo HTTP, y como este está basado estrictamente en peticiones y respuestas, debemos sobreponernos a esta limitante para lograr las mismas ventajas de una aplicación de escritorio.

La implementación de RPC que viene en GWT no provee ningún mecanismo para el manejo de eventos. Para notificar sobre la presencia de los usuarios y para el envío de los mensajes, necesitamos indudablemente que nuestra aplicación soporte eventos. Podríamos utilizar GWT-RPC para chequear periódicamente el servidor para verificar nuevos mensajes o contactos en un intervalo corto de tiempo, pero obviamente esta solución trae consigo las limitantes vistas previamente en cuanto a costo de recursos en el servidor y anchos de banda. Debido a la restricción de HTTP, no podemos implementar una solución pura basada en eventos, sin embargo, podemos obtener una aproximación cercana.

La solución es una combinación entre polling (ver sección 7.1.4.1) y eventos. Tiene la ventaja de soportar eventos instantáneos desde el servidor sin sufrir los mismos problemas de rendimiento que una implementación polling tendría.

La solución se resume en los siguientes pasos:

1. La aplicación llama un método clave llamado getEventos.

2. El servidor maneja esta llamada, identificando el usuario y verificando si hay algún evento pendiente para él. Si lo hay, el servidor retorna una lista de eventos, pero si no, la aplicación bloquea el hilo por un máximo de 30 segundos. Si no hay eventos nuevos durante esos 30 segundos, el método retorna vacío y el cliente llama getEventos de nuevo. Si un evento llega mientras el hilo esta dormido, éste es despertado para que los eventos sean retornados al usuario inmediatamente.

La solución entrega notificaciones instantáneas de eventos y reduce la frecuencia de peticiones al servidor a 30 segundos.

La implementación de nuestro protocolo de eventos, tiene dos componentes fundamentales, uno desarrollado en el cliente, que es bastante sencillo, y el otro, desarrollado en el servidor.

Para el cliente, debemos agregar un nuevo método en ServicioMensajeria y ServicioMensajeriaAsync pues éste método se comunicará con el servidor.

```
List getEventos() en ServicioMensajeria
List getEventos(AsyncCallback callback) en ServicioMensajeriaAsync
```

Luego en el método OnSucess de la LogInCallback agregamos

```
servicioMensajeria.getEventos( new GetEventosCallback() );
```

Con esto, ya estamos llamando por primera vez el método getEventos en el servidor después de que el usuario haga su login en el chat, para de esta forma recibir los eventos correspondientes a los nuevos mensajes y a la presencia de los contactos.

Con el objeto GetEventosCallback, le estamos diciendo al cliente qué tiene que hacer con la respuesta del servidor.

La clase se define de esta manera:

```
private class GetEventosCallback implements AsyncCallback {
   public void onFailure(Throwable throwable){
      GWT.log("error al obtener los eventos",throwable);
   }
   public void onSuccess(Object obj){
      List eventos = (List)obj;
      for( int i=0; i < eventos.size(); ++i ){
```

```
            Object evento = eventos.get(i);
            manejarEventos( evento );
         }
         servicioMensajeria.getEventos( this );
      }
}
```

Si la llamada del método fue exitosa, lo único que hacemos es recorrer la lista de eventos que viene desde el servidor y ejecutar el método manejarEventos con cada uno de ellos. Éste método se encarga de realizar las operaciones específicas en la interfaz del usuario de acuerdo al tipo de evento que reciba.

Posteriormente, se ejecuta él método getEventos para simular un ciclo de eventos que permite la recepción de los nuevos eventos que se generen para el usuario.

Para poder lograr que los eventos se envíen como parámetros de las llamadas GWT-RPC, debemos hacer que implementen la clase IsSerializable. Los tipos de eventos que se recibirán son los siguientes:

```
public class Evento implements IsSerializable{
}
public class LogInEvent implements Evento{
   public Contacto contacto;
}
public class LogOutEvent implements Evento {
   public Contacto contacto;
}
public class EnvioMensajeEvent implements Evento {
   public Contacto remitente;
   public Mensaje m;
}
```

Los atributos de cada uno de estos objetos sencillos de Eventos actúan cómo parámetros para el evento en particular. El servidor genera estos eventos como respuesta a acciones en el cliente. Cuando la instancia de ServicioMensajeriaClienteImpl los recibe, son manejados para invocar las acciones apropiadas en la vista. Este trabajo se hace a través del método manejarEventos:

```
protected void manejarEventos( Object evento ){
   if( evento instanceof EnvioMensajeEvent ){
      EnvioMensajeEvent envioMensajeEvent = (EnvioMensajeEvent)event;
      vista.getVistaVentanaChat(envioMensajeEvent.remitente
).agregarMensaje (
 envioMensajeEvent.mensaje );
   }
   else if( evento instanceof LogInEvent ){
```

```
      LogInEvent logInEvent = (LogInEvent)event;

vista.getVistaListaContacto().agregarContacto(logInEvent.contacto);
   }
   else if( evento instanceof LogOutEvent ){
    LogOutEvent logOutEvent = (LogOutEvent)event;
  vista.getVistaListaContacto().quitarContacto(logOutEvent.contacto);
   }
}
```

Como puede verse, en cada caso, dependiendo del tipo de evento que se reciba, se ejecuta un método de la vista que refleje la acción específica, bien sea, agregar o quitar un contacto de la lista de contactos, o notificar sobre la llegada de un nuevo mensaje.

Ahora miremos la implementación del método getEventos, del lado del servidor:

```
public ArrayList getEventos(){
   /*

     OBTENEMOS UNA REFERENCIA DEL USUARIO ACTUAL Y LO
     ASIGNAMOS EN user

   */

   if( user != null ){
      if( user.eventos.size() == 0 ){

     try{
            synchronized( user ){
               user.wait( 30*1000 );
            }
         }
         catch (InterruptedException ignored){}
      }

      synchronized( user ){
         eventos = user.eventos;
         user.eventos = new ArrayList();
      }
   }
   return events;
}
```

El objetivo de este método es retornar una lista de eventos si estuvieran disponibles. Si no hay eventos disponibles, es necesario llamar el método wait en el objeto user. Éste método es una sincronización de hilos que está disponible para todos los objetos Java. Lo que hace es parar el procesamiento en el hilo actual por un máximo de tiempo especificado como parámetro en milisegundos, en este caso 30.000 milisegundos, o 30

segundos. Si hay eventos, entonces son retornados y se genera un nuevo arreglo vacío para el próximo conjunto de eventos.

Sin embargo, dado que es necesario que los usuarios sean notificados de eventos, aun cuando su hilo se encuentre dormido en el intervalo de los 30 segundos, el hilo continúa si cualquier otro llama el método notifyAll en el objeto. De ésta manera, el método notifyAll será llamado en el objeto de un usuario cuando un evento ha sido agregado a su lista.

Por ejemplo, cuando un usuario entra al sistema de MI a través de una materia de EAFIT Interactiva, el resto de los usuarios que se encuentran conectados en la materia deben ser notificados. Es posible que ellos tengan un hilo inactivo cuando ocurre este evento, entonces, todos ellos deben ser notificados como se muestra a continuación:

Para cada usuario activo en la materia del usuario actual:

Usuario.eventos.agregar (evento de LogIn)

Usuario.notifyAll()

De esta forma los hilos son despertados y los eventos se retornan al cliente para realizar el respectivo cambio en la vista.

Hemos visto entonces, como implementar un protocolo de eventos funcional dentro de la aplicación de mensajería instantánea. Con este manejo simple de conceptos, es posible construir aplicaciones que proporcionen respuestas en tiempo real.

Lo único que queda faltando en este momento, es desplegar la aplicación. Para ello, utilizamos el script que genera el GWT para la compilación y al realizar este proceso, copiamos los archivos generados en el servidor web. La única clase que hay que trasladar al contenedor de nuestro servidor, hablando en términos de código Java, es la que implementa el Servlet, en nuestro caso ServicioMensajeriaImpl. El resto de código Java es traducido por el GWT a código HTML y JavaScript totalmente optimizado, cuidándose

de las diferencias existentes entre todos los navegadores, encargándose de realizar las peticiones Ajax y demás consideraciones de interfaz que hallamos programado. Como vimos, todo fue hecho en Java, sin necesidad de conocer ningún detalle de la programación Web.

10. Recomendaciones Futuras

La implementación de nuestro protocolo está posibilitado para recibir eventos del servidor sin tener que hacer peticiones repetidas en intervalos de tiempo muy pequeños. Esto se logra haciendo que cada petición espere hasta 30 segundos antes de intentar de nuevo, y aun así, pese a este retardo, los clientes continúan recibiendo respuestas instantáneas. De no ser así, tendríamos que ir al servidor como mínimo cada segundo, para verificar si hay eventos nuevos.

Aunque la ganancia en rendimiento es bastante alta con esta solución, no es la ideal. El problema es que cada cliente esperando por un evento bloquea un hilo en el servidor. Los contenedores de Servlets siempre tienen un número máximo de hilos concurrentes que pueden correr al mismo tiempo, lo que significa que este valor es el número máximo de clientes que pueden usar la aplicación de MI a la vez.

Un par de soluciones a estos problemas han sido desarrolladas, pero son en general muy nuevas. La solución requiere de técnicas avanzadas sobre las conexiones realizadas del lado del servidor para hacer llamadas que no bloqueen hilos, permitiendo al servidor liberarlos aunque una conexión esté pendiente.

Cuando estas tecnologías maduren y las librerías incluyan soporte de forma nativa a eventos sería interesante explorar esta solución para las futuras versiones del sistema de Mensajería Instantánea.

11. Conclusiones

La mensajería instantánea, constituye una de las formas de colaboración más explotadas y con mayor proyección para apoyar y en algunas circunstancias reemplazar sistemas colaborativos tradicionales como el correo electrónico. Por esta razón, organizaciones de todo tipo han optado por implementar dentro de sus ambientes productivos, estas herramientas como facilitadores del conocimiento y de la comunicación.

Existen numerosos esfuerzos mundiales por difundir los estándares de la mensajería instantánea. Sin duda, uno de los mas importantes, es Jabber, que gracias a su inmensa comunidad de usuarios y colaboradores han forjado los estándares de estos protocolos instantáneos, y han servido como base para la creación de soluciones propietarias de alta calidad y para la difusión del espíritu de comunicación instantánea a nivel mundial.

El desarrollo Web constituye uno de los grandes desafíos del mundo informático en la actualidad. Con la explosión de la Internet se han desarrollado tecnologías impresionantes para hacer sitios Web cada vez más dinámicos, con interfaces de usuario similares a las de escritorio y con niveles de respuesta óptimos donde el tiempo de los usuarios empieza a ser importante y donde cada vez es menos común el desarrollo de sitios donde sea necesario esperar para recibir una respuesta.

Uno de los grandes adelantos tecnológicos, sin duda es lo que hoy se conoce con el término Ajax. Este conjunto de tecnologías han posibilitado una expansión de las aplicaciones Web respaldadas e impulsadas por gigantes informáticos como Google, gracias a los cuales esta tendencia se ha venido desarrollando fuertemente de la mando de lo que se conoce desde hace algún tiempo como la Web 2.0. Ajax está al alcance de cualquier desarrollador que tenga conocimientos fuertes en las tecnologías que envuelve.

El desarrollo Web de última generación puede resultar difícil si se tienen en cuenta todas las tecnologías que se ven implicadas en el proceso. Esto se ve

agravado por la falta de herramientas consolidadas que enmarquen el proceso de los desarrolladores y por las diferencias existentes entre los diferentes navegadores modernos, lo que dificulta el seguimiento al código y el desarrollo de las pruebas.

GWT, un conjunto de herramientas de Google para el desarrollo Web, buscan encapsular todos los elementos necesarios para que el desarrollo de aplicaciones Web no sea una tarea traumática y pueda compararse en términos de herramientas, a los desarrollos tradicionales. Gracias a su arquitectura, el GWT permite la creación de aplicaciones Web de última generación gracias a una abstracción total del desarrollo al lenguaje de programación Java. De esta forma el control a particularidades de los navegadores, la optimización de código, las técnicas tradicionales de seguimiento al código, los casos de prueba, la ingeniería de software, y demás elementos de los ciclos de desarrollo particulares, pueden ser implementados con toda naturalidad y fluidez.

Para la implementación de los protocolos basados en eventos para arquitecturas Web, es necesario tener claro que HTTP, el protocolo de la Web, está limitado a respuestas desencadenadas por una petición desde un cliente. Es por esto que pensar en que el servidor enviará información al cliente sin que sea solicitada, resulta un poco difícil de imaginar. Sin embargo, vimos como, combinando un par de tecnologías fue posible construir una solución funcional, donde no fue necesario conocer técnicas avanzadas de desarrollo Web, y gracias a las herramientas de GWT, conseguimos que el sistema de MI para la plataforma de EAFIT Interactiva burlara las restricciones del protocolo HTTP.

Una solución cien por ciento basada en eventos requiere de técnicas avanzadas que apenas están siendo incluidas en algunos de los contenedores o servidores Java modernos, por lo cual, un trabajo futuro innegable es explorar las nuevas herramientas que se están desarrollando para continuar con la expansión de las posibilidades en la Web, ahora, en el tema de las respuestas en tiempo real.

12. Bibliografía

Referencias Bibliográficas

[1] HOLZNER, Steven – Ajax Bible – United States of America: Wiley Publishing, 2007. ISBN 9780470102633.

[2] KEITH, Jeremy – Bulletproof Ajax – United States of America: New Riders Press, 2007. ISBN 9780321472663

[3] RITTINGHOUSE, Jhon W, RANSOME, James F – IM Instant Messaging Security – United States of America Elsevier Digital Press, 2005. ISBN 1555583385

[4] DEWSBURY, Ryan – GOOGLE Web Toolkit Applications – United States of America, Prentice Hall, 2008. ISBN 9780321501967

Referencias tomadas de Internet

[5] Wikis en educación: sus múltiples usos. [En línea]. [Citado el 09 de abril de 2008]. Disponible en Internet en:

http://www.slideshare.net/educablog/wikis-en-educacin-sus-mltiples-usos/

[6] IM and collaboration [En línea]. [Citado el 4 de octubre de 2008]. Disponible en Internet en:

http://findarticles.com/p/articles/mi_m0CMN/is_9_44/ai_n21053169

[7] Instant Messaging for Collaboration: A Case Study of a High-Tech Firm. [En línea]. [Citado el 4 de octubre de 2008]. Disponible en Internet en:

http://jcmc.indiana.edu/vol10/issue4/quan-haase.html

[8] Instant Messengers: A Brief History . [En línea]. [Citado el 4 de octubre de 2008]. Disponible en Internet en:

http://im.about.com/od/imbasics/a/imhistory.htm

[9] Jabber Servers. [En línea]. [Citado el 4 de octubre de 2008]. Disponible en Internet en:

http://www.jabber.org/servers

[10] Asincrónico. [En línea]. [Citado el 4 de octubre de 2008]. Disponible en Internet en:

http://www.natureduca.com/tecno_gloselec_a02.php

[11] Sincrónico. [En línea]. [Citado el 4 de octubre de 2008]. Disponible en Internet en:

http://club.telepolis.com/geografo/glosario/s.htm

[12] Ajax: A New Approach to Web Applications. [En línea]. [Citado el 4 de octubre de 2008]. Disponible en Internet en:

http://www.adaptivepath.com/ideas/essays/archives/000385.php

[13] Exploring Reverse AJAX. [En línea]. [Citado el 4 de octubre de 2008]. Disponible en Internet en:

http://gmapsdotnetcontrol.blogspot.com/2006/08/exploring-reverse-ajax-ajax.html

[14] Comet: Low Latency Data for the Browser. [En línea]. [Citado el 4 de octubre de 2008]. Disponible en Internet en:

http://alex.dojotoolkit.org/2006/03/comet-low-latency-data-for-the-browser/

[15] GWT - Product Overview. [En línea]. [Citado el 4 de octubre de 2008]. Disponible en Internet en:

http://code.google.com/webtoolkit/overview.html

Referencias a Wikipedia

[16] [En línea]. [Citado el 4 de octubre de 2008]. Disponible en Internet en:

http://en.wikipedia.org/wiki/Instant_messaging#IM_language

[17] [En línea]. [Citado el 4 de octubre de 2008]. Disponible en Internet en:

http://en.wikipedia.org/wiki/List_of_XMPP_client_software

[18] [En línea]. [Citado el 4 de octubre de 2008]. Disponible en Internet en:

http://es.wikipedia.org/wiki/CSS

[19] [En línea]. [Citado el 4 de octubre de 2008]. Disponible en Internet en: http://es.wikipedia.org/wiki/HTML

[20] [En línea]. [Citado el 4 de octubre de 2008]. Disponible en Internet en: http://es.wikipedia.org/wiki/JavaScript

[21] [En línea]. [Citado el 4 de octubre de 2008]. Disponible en Internet en: http://es.wikipedia.org/wiki/HTTP

[22] [En línea]. [Citado el 4 de octubre de 2008]. Disponible en Internet en: http://en.wikipedia.org/wiki/Reverse_Ajax

[23] [En línea]. [Citado el 4 de octubre de 2008]. Disponible en Internet en: http://en.wikipedia.org/wiki/Comet_(programming)

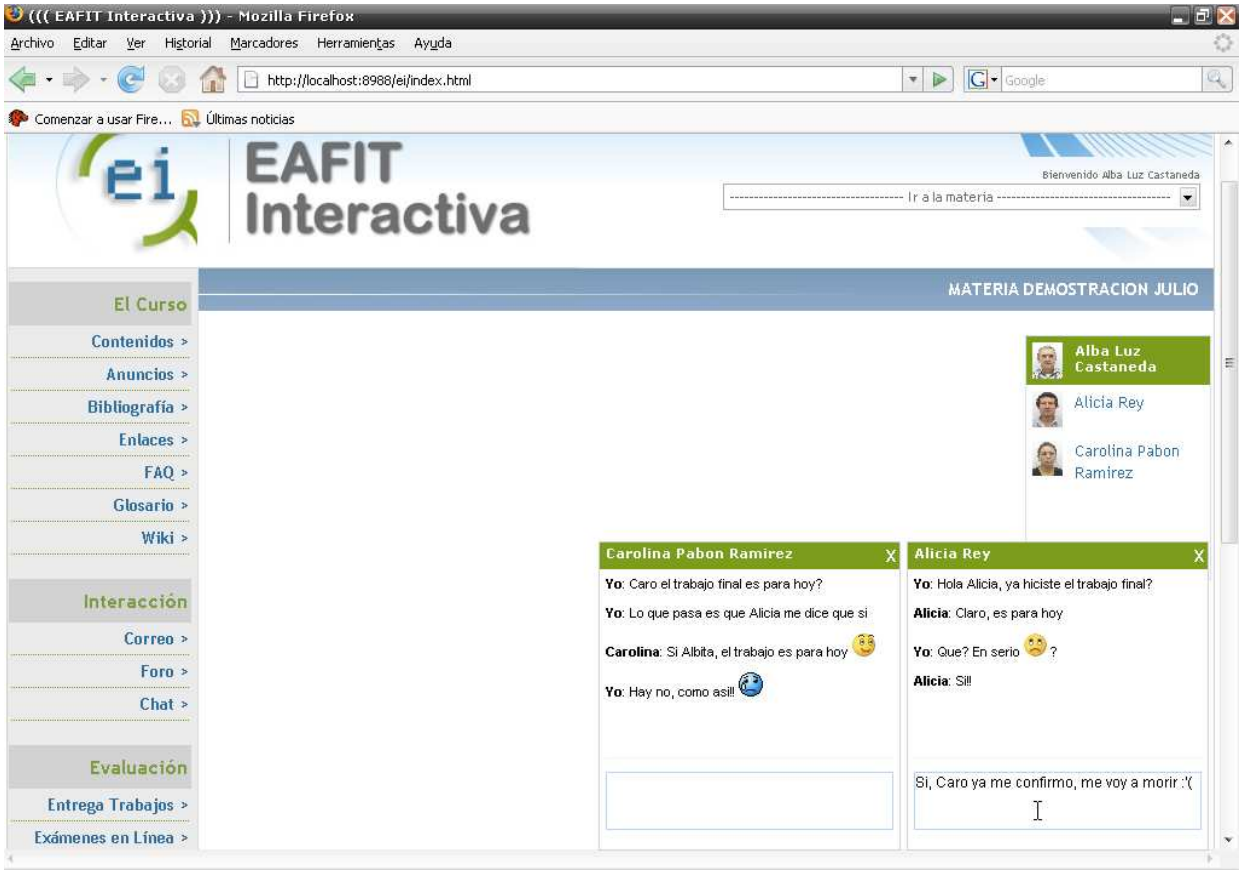
[24] [En línea]. [Citado el 4 de octubre de 2008]. Disponible en Internet en: http://en.wikipedia.org/wiki/Serialization

[25] [En línea]. [Citado el 4 de octubre de 2008]. Disponible en Internet en: http://es.wikipedia.org/wiki/Remote_procedure_call

[26] [En línea]. [Citado el 4 de octubre de 2008]. Disponible en Internet en: http://code.google.com/webtoolkit/overview.html

[27] [En línea]. [Citado el 4 de octubre de 2008]. Disponible en Internet en: http://en.wikipedia.org/wiki/Model-view-controller

13. Anexo


(((EAFIT Interactiva))) - Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas Ayuda
http://localhost:8988/ei/index.html
Comenzar a usar Fire... Últimas noticias
EAFIT Interactiva
Bienvenido Alba Luz Castaneda
Ir a la materia
El Curso
Contenidos >
Anuncios >
Bibliografía >
Enlaces >
FAQ >
Glosario >
Wiki >
Interacción
Correo >
Foro >
Chat >
Evaluación
Entrega Trabajos >
Exámenes en Línea >
MATERIA DEMOSTRACION JULIO
Alba Luz Castaneda
Alicia Rey
Carolina Pabon Ramirez
Carolina Pabon Ramirez
Yo: Caro el trabajo final es para hoy?
Yo: Lo que pasa es que Alicia me dice que si
Carolina: Si Albita, el trabajo es para hoy
Yo: Hay no, como asi!!
Alicia Rey
Yo: Hola Alicia, ya hiciste el trabajo final?
Alicia: Claro, es para hoy
Yo: Que? En serio ?
Alicia: Si!!
Si, Caro ya me confirmo, me voy a morir :'(