PSD-ims

Memoria del proyecto

Daniel Pinto Rivero

Javier Bermúdez Blanco

Esta página se ha dejado en blanco intencionadamente

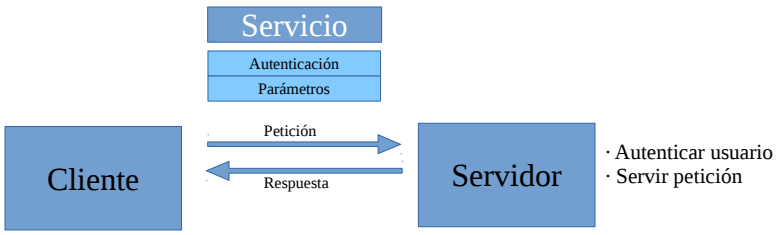
Table of Contents

1 Resumen

El objetivo de este proyecto es diseñar e implementar un Servicio de Mensajería Instantánea (IMS) utilizando Servicios Web. En este caso, se va a utilizar la librería gSOAP para realizar la comunicación entre las distintas partes del sistema.

1.1 Funcionamiento general

Esta aplicación está basada en servicios web **sin sesión**, lo que significa que en todas las llamadas al servidor, el cliente debe incluir la información de autenticación del usuario.



Este modo de trabajar reduce la complejidad del servidor y permite que un usuario se conecte desde varias aplicaciones cliente a la vez sin gestión adicional en el lado del servidor, a cambio de la pequeña carga de trabajo que supone autenticar al usuario en cada llamada. Sin embargo, aunque se utilizaran sesiones, es necesario comprobar que el usuario que hace la llamada es el propietario de la sesión, por lo que la sobrecarga adicional es mínima.

1.1.1 Registro de usuarios

El registro en el sistema se hace a través de un servicio público que no necesita autenticación, mediante el cual se envía al servidor el nombre del nuevo usuario, su contraseña y su información asociada, que por el momento consiste únicamente en una cadena de texto para incluir una descripción.

Cada usuario se identifica unívocamente mediante su nombre de usuario, por lo que este no puede estar repetido, no permitiendo el servidor el registro si no se cumple esto.

La baja se realiza a través de otro servicio público, que, al contrario que el registro, necesita autenticación. Cuando un usuario se da de baja, se marca como inválido, se le elimina de todos los chats en los que formara parte y se eliminan todas las relaciones de amistad en las que apareciera, de modo que toda su información permanece en el sistema, así como los mensajes que haya enviado, pero no se le permite hacer login.

1.1.2 Relación de amistad

Para que un usuario tenga acceso a la información de otro, así como la capacidad de agregarle a chats, deben ser "amigos". Esta es una relación bidireccional, de modo que un usuario no puede ser "amigo" de otro sin que este sea, a su vez, "amigo" del primero.

Para que dos usuarios pasen a ser "amigos", uno de ellos debe enviar una solicitud de amistad, indicando el nombre del usuario al que se la envía. Entonces, este podrá decidir si acepta la solicitud, momento en el que ambos pasarán a ser "amigos", o la rechaza, lo que simplemente borrará la solicitud, pero no impide que el primer usuario vuelva a enviarla. Sin embargo, este no será notificado de tal suceso.

1.1.3 Chats

Los usuarios se comunican a través de chats. Estos chats funcionan como una especie de depósito de mensajes, con una lista de usuarios miembros asociada, y un usuario con permisos especiales, conocido como "administrador del chat".

El administrador del chat será el único que pueda agregar y eliminar miembros. Además, tiene la capacidad de pasar su rol de administrador a otro miembro.

Los miembros pueden enviar mensajes al chat, que serán accesibles para todos los miembros de este que hayan sido agregados antes del envío. Los miembros también pueden abandonar el chat cuando quieran, momento a partir del cual dejarán de tener acceso a los mensajes y la información de este, tal como su descripción y su lista de miembros.

Debido a este diseño, todos los chats son grupales, de modo que si un usuario quiere mandar mensajes a otro, creará un chat del que formarán parte sólo los dos.

1.1.4 Archivos adjuntos

Los mensajes pueden o no tener archivos adjuntos, lo cual se indicará en el momento en que se envíe el mensaje. Sin embargo, el adjunto no se envía en la misma llamada (servicio web), si no en otra diferente, para dar mayor flexibilidad a las aplicaciones cliente, pudiendo escoger el momento para enviar estos archivos.

Igual que al enviar mensajes, al recibirlos, tendrán un marca que indica si tienen o no archivo adjunto, el cual se descarga mediante otra llamada diferente.

Esto se puede hacer porque cada mensaje está identificado unívocamente[1] del modo que se indica en la sección de Archivos adjuntos.

1 Cada mensaje se identifica mediante el código del chat al que pertenece y su timestamp, ya que el servidor se asegura de que todos los mensajes del mismo chat tengan timestamps distintos.

1.2 Resumen de servicios

A continuación se describen por encima cada uno de los servicios que ofrece el servidor a la aplicación cliente. Estos servicios conforman la funcionalidad completa de la aplicación.

Servicio	Descriptión
user_register	Registrar un usuario en el sistema. El nombre de usuario debe ser único.
user_unregister	Eliminar un usuario del sistema. La información de este usuario no se borra, si no que es invalidado y se le elimina de todos los chats a los que perteneciera.
get_user	Obtener la información asociada al usuario autenticado.
get_friends	Obtener la lista de “amigos”
get_friend_info	Obtener la información asociada a un usuario “amigo”
get_chats	Obtener la lista de chats.
get_chat_info	Obtener la información asociada a un chat al que pertenezca el usuario autenticado.
get_chat_messages	Obtener la lista de mensajes del un chat del que forme parte el usuario autenticado.
get_attachment	Descargar el archivo adjunto a un mensaje concreto.
get_pending_notifications	Obtener las notificaciones pendientes del usuario. Esto incluye: Nuevos amigos, usuarios agregados o eliminados de chats, chats con mensajes, ...
get_all_data	Obtener todas las listas asociadas al usuario, es decir, amigos, chats, mensajes y solicitudes de amistad recibidas sin resolver.
create_chat	Crear un nuevo chat en el sistema con un miembro. (Además del propio usuario, que será el administrador del chat)
add_member	Agregar un miembro a un chat existente del que el usuario es administrador. (El nuevo miembro debe ser “amigo” del usuario que le agrega.
remove_member	Eliminar un miembro de un chat del que el usuario es administrador.
quit_from_chat	Salir de un chat.
send_message	Enviar un mensaje a un chat del que el usuario es miembro.
send_attachment	Subir el archivo adjunto de un “mensaje con adjunto” previamente enviado a un chat.

send_friend_request	Enviar una solicitud de amistad a un usuario existente en el sistema.
accept_request	Aceptar una solicitud de amistad recibida.
decline_request	Rechazar una solicitud de amistad recibida.

Para obtener más información sobre estos servicios, y las estructuras con las que trabajan, consultar el archivo de cabecera “rpc/ims.h”

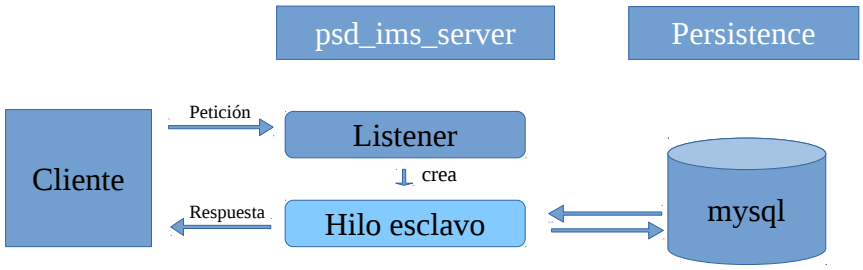
2 Servidor

El servidor de PSD-IMS recibe y sirve las peticiones que le llegan. Esto se hace mediante los manejadores de las rutinas de gSOAP.

Gestiona la lista de usuarios registrados en el sistema, las relaciones de amistad existentes entre ellos, las peticiones de amistad, los chats, listas de miembros y mensajes, y los archivos adjuntos.

2.1 Implementación

El servidor permite su ejecución en modo mono o multihilo, dependiendo de si se cumplen o no los requisitos necesarios para ejecutarse en modo multihilo.



Por el momento solo existe un requisito, y es que mysql haya sido conpilado en modo thread-safe. Si es así, se escojerá automáticamente el modo multihilo, manteniendo el hilo principal para escuchar nuevas conexiones entrantes. Una vez que llega una nueva conexión, este hilo principal creará un hilo esclavo, con una copia de la estructura gsoap (tal como se indica en la documentación) y una copia de la estructura mysql, que básicamente consiste en un conector nuevo con la misma información de autenticación que el original.

Como cada hilo usa un conector diferente a la base de datos, y el servidor se ha asegurado de que mysql es thread-safe, no debería haber problemas con acceso a secciones críticas en mysql, sin embargo existen problemas potenciales en el acceso a información de distintas tablas, que podría quedar inconsistente en

algunos casos. Dicho esto, aún no se ha encontrado ningúno de estos casos.

Los servicios se sirven realizando consultas directamente a mysql, es decir, el servidor no mantiene estructuras de ningún tipo (excepto la estructura principal de servidor, que incluye el conector principal a mysql y la estructura soap)

2.1.1 psd-ims-server.c

Este archivo contiene la lógica principal del servidor. Consiste en la estructura de servidor, las rutinas para inicializar y liberar dicha estructura, esperar y servir una nueva conexión,… Además de los manejadores de las rutinas de gSOAP.

2.1.2 persistence.c

La persistencia se lleva a cabo mediante una base de datos mysql, sin embargo, toda esta gestión queda asbtraida en el archivo persistence.c, de modo que se podría cambiar tan solo modificando este archivo.

2.1.3 Gestión de usuarios

2.1.3.1 Usuarios

Un usuario registrado es aquel que tiene una entrada en la tabla de usuarios, marcada como válida.

Los usuarios registrados tienen la capacidad de enviar y recibir solicitudes de amistad, así como aceptarlas o rechazarlas, crear chats, agregar y eliminar miembros de chats de los cuales sean administradores, enviar mensajes a chats de los cuales sean miembros, ser agregados y abandonar chats, y darse de baja del sistema.

Cuando un usuario se da de baja, su información se mantiene almacenada en el sistema por coherencia de los datos, pero se le elimina de todos los chats de los que fuera miembro, así como todas las relaciones de amistad en las que apareciera, y se marca su entrada como inválida, de modo que no se pueden utilizar sus credenciales para invocar servicios, no se le pueden enviar solicitudes de amistad, ni agregarle como miembro de un chat.

2.1.3.2 Relaciones de amistad

Para que un usuario pueda agregar a otro como miembro de un chat, estos deben ser amigos, es decir, es necesario que exista una relación de amistad para poder iniciar una comunicación, sin embargo, no es estrictamente necesario para poder comunicarse, ya que un tercer usuario, amigo de ambos, podría crear un chat al que los agregara, de modo que podrían enviarse mensajes sin ser amigos. De hecho, si este usuario abandonara el chat, se le pasaría el papel de administrador a uno de los otro usuarios, y estos se escribirían sin la presencia del amigo común. Esto no es un problema, ya que el requisito para que un usuario forme parte de un chat, es que le agrege un “amigo”, que debe ser administrador del chat, no es necesario que todos los miembros del chat sean amigos, y eventualmente, podría haber miembros que no sean amigos ni del administrador.

Para crear una nueva relación de amistad, un usuario debe enviar una solicitud de amistad al servidor, indicando el usuario al que se la envía. En principio, los usuario no tienen manera de saber los nombres de otros usuarios del sistema, a no ser que se lo hayan comunicado entre ellos fuera de la aplicación, o sean miembros de algún chat común.

Una vez el servidor recibe la solicitud de amistad, se le notifica al usuario receptor para que pueda aceptarla o rechazarla. Si la acepta, ambos pasarán a ser amigos, y el primer usuario será notificado. Si la rechaza, la solicitud se elimina del sistema, pero no se notifica al primer usuario.

2.1.4 Gestión de chats

Las comunicaciones en PSD-ims se llevan a cabo mediante “chats”. Estos chats son basicamente contenedores de mensajes con una información y una lista de miembros asociada, además de un miembro especial, el administrador del chat.

2.1.4.1 Miembros

Los miembros de un chat son todos aquellos usuarios que aparezcan en la lista de miembros del mismo, estos tienen acceso a los mensajes del chat que hayan sido enviados después de que se les agregara como miembros, así como a los nombres de todos los miembros, incluido el administrador.

Para ser miembro de un chat, el administrador de este, que debe ser amigo del usuario, debe agregarle explícitamente.

Una vez que el usuario es agregado a un chat, se le notificará, y empezará a recibir las notificaciones correspondientes al chat. Además, podrá obtener los mensajes que se envíen a partir de ese momento, enviar mensajes, salirse del chat y, potencialmente, convertirse en el administrador del chat.

El administrador del chat, es un miembro más, con todas las capacidades de estos, además del poder de agregar amigos suyos al chat, y eliminar miembros del mismo.

2.1.4.2 Mensajes

Los mensajes son porciones de comunicaciones entre usuarios, que constan de una cadena de texto, el nombre del emisor, el momento exacto en que el servidor lo recibió, y una marca indicando si tienen o no un archivo adjunto.

Estos mensajes no se envían directamente entre usuarios, si no que se envían a chats, donde quedan almacenados, para que tengan acceso a ellos todos los usuarios que sean miembros del chat en el momento del envío.

2.1.4.3 Archivos adjuntos

Para enviar archivos entre usuarios, estos deben adjuntarse a mensajes, lo que no quiere decir que se envíen juntos, primero se envía el mensaje, y se indica que tiene un archivo adjunto, entonces, utilizando el código del chat y la marca de tiempo (identificación unívoca del mensaje) se envía el archivo con una llamada diferente. El servidor identifica el mensaje al que se pretende adjuntar el archivo, y si esto es posible, almacena el archivo, y lo mantiene disponible para su descarga por parte de usuarios autorizados

(aquellos con acceso al mensaje)

Cuando otros usuarios reciban el mensaje, les aparecerán con la marca de archivo adjunto, y podrán descargar el archivo con una llamada diferente a la usada para obtener el mensaje.

Este gestión de los archivos, separándolos de los mensajes en los que están adjuntos, permite a las aplicaciones cliente gestionar el momento más oportuno para la carga o descarga del archivo, una flexibilidad muy conveniente en entornos con la red limitada.

2.1.5 Mecanismos de sincronización con el cliente

La información existente en el servido es la que prima; si dos usuarios aparecen como amigos en el servidor, lo serán a todos los efectos, aunque los clientes no tengan constancia de ello. Si no lo hacen, no lo son, aunque así aparezcca en los clientes. Es importante que los clientes tengan las herramientas necesarias para mantener su información sincronizada con la del servidor, de modo que puedan exponer al usuario su realidad dentro del sistema.

Para ello se utiliza las llamadas *get_all_data* y *get_pending_notifications*.

2.1.5.1 get_all_data

Este servicio permite obtener toda la información pertinente al usuario autenticado, es decir, su lista de amigos con la información de estos, su lista de chats, los miembros de estos, …

Se espera que se la llame sólo una vez, al arrancar la aplicación cliente, si es que no tenía ninguna información previa del usuario.

2.1.5.2 get_pending_notifications

Este servicio es el que permite hacer eco a las aplicaciones cliente de los cambios registrados en el servidor. Devuelve la lista de miembros que han sido agregados o eliminados a algún chat, incluido el propio usuario, la lista de nuevos amigos, la lista de chats con mensajes nuevos de los que el usuario sea miembro, …

Es la llamada que más se utilizaría por parte de los clientes, se espera que sea invocada de forma periódica, para mantener la sincronización.

Probablemente es el servicio que provoque una mayor sobrecarga en el servidor, ya que debe realizar varias consultas a la base de datos, para rellenar una estructura con cierta complejidad. Como se han hecho las pruebas con un pequeño número de clientes conectados concurrentemente, y en una red con muy poca latencia, esto no ha sido un problema, pero sería muy conveniente llevar a cabo pruebas de estrés para determinar si este servicio supone un cuello de botella y, de ser el caso, buscar soluciones tales como mantener los últimos cambios cacheados en nuevas estructuras en el servidor.

2.1.5.3 Timestamps

Sería muy poco conveniente que los clientes tuvieran que descargar todas las listas para mantenerse sincronizados. Por ejemplo, no esperaríamos tener que descargar la lista de amigos completa para saber si

el usuario tiene algún amigo nuevo, o lo que sería peor, descargar toda la lista de chats, o todos los mensajes de un chat… Para evitar esto, a casi toda la información que registra el servidor, le asigna una marca de tiempo, o timestamp, que devuelve al cliente en los casos en que sea necesario. Por ejemplo, cuando un cliente envía un mensaje, el servidor obtiene el tiempo actual, que asigna al mensaje y devuelve al cliente una vez lo ha registrado correctamente.

Esto permite que el cliente solicite los elementos de una determinada lista a partir de un timestamp concreto, obviando todos los anteriores.

Esta marca de tiempo, por limitaciones de mysql, tiene una precisión de segundos, lo cual es un problema en un sistema que sirve más de una petición por segundo, como es el caso de cualquier máquina actual en la que corriera el servidor. De momento, el cliente debe asumir que es posible que reciba alguna notificación repetida, y gestionarlo convenientemente.

2.1.5.4 Sincronización de mensajes

Los mensajes, al identificarse mediante el identificador del chat y el timestamp, no se puede permitir que tengan timestamps repetidos dentro del mismo chat, lo cual es gestionado por el servidor.

Esta gestión debe considerarse como un parche, ya que la solución ideal sería cambiar los timestamps por otros con una precisión de, almenos, milisegundos.

Para garantizar esto, el servidor comprueba que el chat que recibe el mensaje no tenga ningún mensaje con el mismo timestamp, si no es así, esperará un pequeño tiempo (una fracción conveniente de segundo) para volver a obtener el momento actual e intentar la inserción. Se intuye que esto podría bloquear la llamada el suficiente tiempo para causar un timeout innecesario en el cliente, un problema a tener en cuenta.

Cuando un usuario envía un mensaje a un chat, este mensaje será notificado de vuelta al usuario emisor igual que al resto de miembros, esta es una consideración importante a la hora de diseñar un cliente, para evitar que los mensajes enviados por el propio usuario aparezcan duplicados en la lista de mensajes del cliente.

Este comportamiento, además, sirve como confirmación de recepción.

Los mensajes, como el resto de listas del sistema, se proveen al cliente a partir del timestamp que este especifique, sin embargo, es deseable que un usuario sólo pueda obtener los mensajes de un chat enviados a partir del momento en que se le agregó al mismo. Por ello, se mantiene también un timestamp con el momento en que el usuario es agregado al chat.

2.1.5.5 Doble check

Es interesante que los miembros de un chat puedan saber cuando todos los demás han leído algún determinado mensaje, por ello, se mantiene también un timestamp que indica cual es el último mensaje que ha leído cada miembro. Es decir, por donde van en la conversación.

Esto se expone a los cliente notificando el máximo timestamp al que han llegado todos los miembros de cada chat del que forme parte el usuario, pudiendo considerar que todos los miembros han leído los mensajes anteriores a dicha marca de tiempo.

3 Cliente

El cliente envía peticiones al servidor (invoca sus servicios) usando el stub generado por gSOAP

3.1 Implementación

El cliente gestiona todas las listas locales, implementadas como listas enlazadas genéricas (common/list.h). Estas listas son: la lista de amigos, de solicitudes de amistad y de chats. Además, cada chat tiene una lista de miembros y otra de mensajes.

Cada una de estas listas tiene un archivo fuente (.c) y uno de cabecera (.h) para describir e implementar las estructuras y funciones específicas de la lista.

También existen otros archivos fuente, como network.c o persistence.c, para implementar el resto de la funcionalidad.

Todo esto queda abstraido bajo la API de cliente, definida en psd_ims_client.h, la única cabecera que se espera que la aplicación cliente tenga que incluir de este código.

3.1.1 Common/list.h

Todas las listas del cliente están implementadas con esta lista genérica. Consiste en una lista doblemente enlazada con nodo fantasma a la que se puede indicar el máximo número de elementos.

La estructura principal de la lista consiste en un puntero al nodo fantasma, un puntero genérico para cualquier información que sea necesario mantener, y punteros a funciones para liberar la información genérica de la lista, el elemento genérico de la lista, comparar dos elementos de la lista, y comparar un elemento de la lista con un valor genérico. Las primeras dos funciones es obligatorio definirlas en el momento en el que se crea la lista, las otras dos se deben asignar a mano. Si alguna no se define, parte de la funcionalidad no estará operativa, pero no crasheará.

Cada nodo es simplemente un puntero al nodo anterior, un puntero al nodo siguiente y un puntero genérico para el elemento de la lista.

Se definen también, como macros, un iterador y varios métodos para acceder a los elementos de la lista y de los nodos.

3.1.2 API del cliente

La API está enteramente incluída en psd_ims_client.h, esta cabecera incluye la estructura "psd_ims_client", con toda la información del cliente, como: nombre de usuario, contraseña, información del usuario, timestamp de las notificaciones, la estructura de network, las listas de amigos, solicitudes de amistad y chats, y los cerrojos para el acceso seguro a la estructura (estos cerrojos se gestionan internamente, así que el implementador del cliente no debe preocuparse por ello).

Todas las funciones y macros de psd_ims_client.h son thread-safe.

Aunque se podría acceder directamente a los elementos de "cliente", e incluir las cabeceras de las listas para modificarlas directamente, lo recomendable es incluir únicamente psd_ims_client.h y utilizar las macros y funciones de esta para el acceso a los miembros.

Entre las macros disponibles, hay iteradores thread-safe para todas las listas, incluidas las de miembros y mensajes, métodos para acceder a los elementos de la lista a través del iterador y métodos para modificar algunos de ellos.

Entre las funciones, está toda la funcionalidad que esperamos para implementar un cliente, como: login, logout, user_register, user_unregister, send_message, create_chat, recv_notifications,...

3.1.3 Comunicación con el servidor

Para comunicarse con el servidor, se emplea el stub creado por gSOAP, pero este no se accede directamente en psd_ims_client, si no que su funcionalidad está encapsulada en el archivo de fuentes network.c.

"network.h" define la estructura "network", con la información de login tal y como está definida por el rpc, un booleano para indicar si se ha podido hacer el login correctamente, la url del servidor, y la estructura soap.

Están definidas también todas las funciones que esperamos para comunicarnos con el servidor.

3.1.3.1 Login

Aunque se utiliza el término "login", por el diseño del servidor no hay un lógin clásico, ya que el servidor no tiene constancia de esto. Sin embargo, esto es algo que queda abstraido en "network", que lo que hace es comprobar que las credenciales del usuario sean correctas, invocando el servicio *get_user_info*, para obtener la información asociada al usuario.

3.1.3.2 Invocación de servicios

Como se ha indicado anteriormente, los servicios del stub no se invocan directamente, si no que quedan encapsulados por "network", donde se definen las funciones necesarias para tal propósito.

Estas funciones, en ocasiones, devuelven estructuras del rpc, que han sido previamente desenlazadas de soap, por lo que deben ser liberadas explícitamente fuera de "network" mediante los métodos "net_free_", también definidos en "network".

Como devuelve directamente las estructuras del rpc, psd_ims_client debe incluir, además de network.h, soapH.h, para conocer su implementación.

3.1.3.3 Notificaciones

Las notificaciones es uno de los puntos más importantes del cliente, ya que permite al cliente actualizar sus estructuras acordes al servidor.

Esto se hace mediante la funcion de "network", *net_recv_notifications*, que queda expuesta en la api, como *psd_recv_notifications*, que invoca la función de network y actualiza las listas correspondientes, sin más gestión necesaria por parte del programador del cliente.

Se espera que esta función se llame de manera periódica, para mantener el cliente actualizado.

3.1.4 Persistencia

El diseño de la parte de cliente incluye una abstracción que permite la persistencia de las listas y demás información relativa al cliente, de modo que sólo sea necesario emplear el servicio *get_all_data* la primera vez que se hace login con cada usuario, a partir de entonces, al hacer login de nuevo, se cargaría la información del disco, y se mantendría la sincronización con *psd_recv_notifications*. Sin embargo, esto aún no ha sido implementado, de modo que es necesario llamar a *get_all_data* cada vez que se hace login.

3.1.5 Gestión de amigos

3.1.5.1 Amigos

Los amigos del usuario se mantienen en la lista "friends" de la estructura "psd_ims_client".

La API proporciona un iterador con los métodos correspondientes para el acceso a los elementos de la lista.

Esta lista no se modifica directamente por el cliente, si no que lo hace psd_ims_client.c en las llamadas *psd_recv_notifications* y *psd_recv_all_data*.

Es decir, cuando se acepta una solicitud de amistad, no se agrega el amigo localmente, si no que se espera a que el servidor lo notifique. Esto es así porque al funcionar mediante timestamps, al haberse agregado al amigo después de la última petición de notificaciones, el servidor va a notificarlo igualmente, asíque se evita la gestión necesaria para evitar entradas repetidas y además sirve como comprobación de que el servidor ha registrado la relación de amistad correctamente.

Aunque no está implementado, sería conveniente una función para eliminar amigos de la lista desde la aplicación cliente, ya que el servidor no notifica cuando elimina una relación de amistad.

3.1.5.2 Solicitudes de amistad

Las solicitudes de amistad se mantienen en la lista "request" de la estructura "psd_imc_client".

La API proporciona un iterador con los métodos correspondientes para el acceso a los elementos de la lista.

Esta lista también se modifica mediante las llamadas *psd_recv_notifications* y *psd_recv_all_data*, sin embargo, cuando aceptamos una solicitud de amistad, mediante la llamada p*sd_send_request_accept*, la entrada en la lista de solicitudes se borra.

3.1.6 Gestion de chats

Los chats se mantienen en la lista “chats” de la estructura “psd_ims_client”.

La API proporciona un iterador con los métodos correspondientes para el acceso a los elementos de la lista, además de los iteradores para el acceso a los mensajes y miembros de cada chat.

Las listas de chats y miembros se modifican a través de *psd_recv_notifications* y *psd_recv_all_data*, ya que el servidor notifica cuando ocurre algún evento que necesite actualización.

3.1.6.1 Miembros

Los miembros se mantienen en una lista de miembros dentro de cada chat de la lista de chats.

Sólo el administrador del chat podrá agregar a sus amigos como miembros del chat, y borrar a miembros, pero esta es una restricción del lado del servidor, de modo que al invocar las llamadas correspondientes de psd_ims_client, se invocará el servicio, pero el servidor lo rechazará si no se cumple esta condición.

La lista de miembros no se actualizará nunca directamente desde la aplicación cliente, si no mediante las llamadas *psd_recv_all_data* y *psd_recv_notifications*.

3.1.6.2 Mensajes

Los mensajes se mantienen en una lista de mensajes dentro de cada chat de la lista de chats.

El servidor notifica cuando algún chat tiene mensajes nuevos, pero no los envía. Para ello se proveen las funciones *psd_recv_messages*, *psd_recv_pending_messages*, *psd_recv_all_messages* y *psd_recv_all_pending_messages*, para recibir los mensajes de un chat, los mensajes de un chat sólo si tiene la marca de mensajes pendientes, recibir los mensajes de todos los chats y recibir los mensajes de todos lo chats que tengan la marca de mensajes pendientes, respectivamente.

A los chats se les marca si tienen mensajes pendientes en la llamada *psd_recv_notification*s.

3.1.6.3 Archivos adjuntos

Para enviar archivos, deben adjuntarse a mensajes. Para ello, en el momento de enviar el mensaje (llamada *psd_send_message*), se debe indicar la ruta al archivo y la información asociada al mismo. Esta información aún no se utiliza para nada, pero podría ser útil.

Aunque se utilizan dos servicios distintos para enviar el mensaje y para enviar el adjunto, la API lo hace todo, de momento, en *psd_send_message*. Sin embargo, a la hora de recibir los mensajes, no recibe también el adjunto. Para ello se provee la función *psd_recv_attachment*, a la que habrá que indicar el

identificador del chat y el timestamp del mensaje.

Los archivos enviados adjuntos a mensajes se copian en el directorio ATTACH_FILES_DIR_SND, definido como “attached_files_snd” por defecto en psd_ims_client.h. Los archivos adjuntos se copian en la llamada *psd_send_message*, justo antes de enviarlos al servidor.

Los archivos recibidos adjuntos se copian en el directorio ATTACH_FILES_DIR_RCV, definido como “attached_files_rcv” por defecto en psd_ims_client.h.

El nombre con el que se crearán los archivos adjuntos, tanto los enviados como los recibidos será _<chatid><msgtimestamp>, siendo responsabilidad del programador del cliente proveer al usuario de la aplicación un identificador más apropiado para ocultar estos nombres y facilitar su uso.

4 Ejemplo de cliente (client_graphic_v2)

La aplicación cliente incluída (client_graphic_v2) es un ejemplo del uso de la API “psd_ims_client” totalmente funcional.

Este cliente funciona completamente en consola, de modo que que todo funciona con comandos. Los comandos de cada pantalla aparecen en la cabecera y empiezan siempre con “/”.

4.1 Pantallas

La primera pantalla que aparece es la de **login**, que permite además registrar nuevos usuarios. Una vez autenticado en el sistema, aparece la pantalla principal, con los **chats** del usuario, indicando el número de mensajes no leídos de cada uno de ellos. Si introducimos un código de chat, aparece la pantalla del chat, con los mensajes del mismo, aquí podremos enviar un mensaje simplemente escribiéndolo. Si la cadena de texto empieza con “/” se interpretará como un comando.

Las pantallas no se refrescan automáticamente, ya que el eco de la entrada de usuario aparece debajo de la misma, por lo que el comportamiento del refresco automático sería indeseable al dar la sensación de que se borra la entrada del usuario constantemente.

4.2 Notificaciones

Para obtener las notificaciones, después del login se crea un thread que llama continuamente a la función de la API *psd_get_notifications* con un intervalo de tiempo conveniente entre cada llamada.

5 Compilación

Para compilar psd-ims, hay que situarse en el directorio raíz de proyecto, y ejecutar "*make*".

El makefile incluído tiene otras opciones de compilación para mostrar información adicional durante la ejecución del programa. Las más interesantes son force_debug y force_trace, La primera muestra los mensajes de info y failures, la segunda, además de infos y failures, muestra el trace de llamadas a funciones de psd-ims.

Estas funciones de debug, sobretodo force_trace puede hacer inusable el cliente de ejemplo, ya que el hilo de notificaciones satura la pantalla con mensajes de trace.

Para compilar el proyecto, hay que cumplir algunas dependencias, entre las cuales están: tener instalado mysql5, sus headers, gsoap y las build tools de gnu.

Una vez compilado el proyecto, los ejecutables para el cliente y el servidor se crearán en el directorio bin/ y los archivos objeto en build/

Se incluyen también en el proyecto algunos scripts para la creación de la base de datos. script/runtest.sh creará la base de datos descrita en script/sql/db-psd.sql, usando el usuario psd-server. Si dicho usuario no existe en la base de datos local, con permisos suficientes, deberá crearse o modificar el script.