Proyecto Cupi2	ISIS-1205 Algorítmica y Programación **Descripción**
Ejercicio:	
Autor:	
Fecha:	

Descripción de la Aplicación

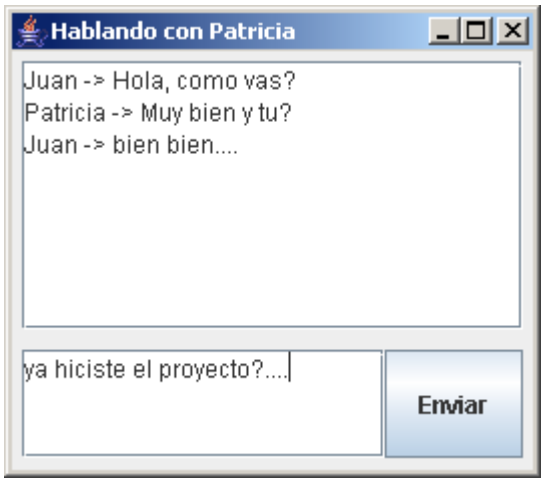
El objetivo de estas aplicaciones es construir un sistema de mensajería al estilo del MSN Messenger, Yahoo Messenger o AIM. Para esto será necesario construir una aplicación que funcione como servidor y un cliente: los usuarios de la aplicación usarán el cliente para conectarse al servidor y este les informará sobre el estado de sus amigos (están conectados o desconectados).

Cada una de los usuarios de la aplicación tiene un nombre con el que se identifica en el sistema y puede construir una lista de amigos. Mientras esté utilizando el sistema, recibirá notificaciones cuando sus amigos se conecten o desconecten. El hecho de que un usuario tenga dentro de su lista de amigos a otro usuario no implica que al revés también sea cierto.

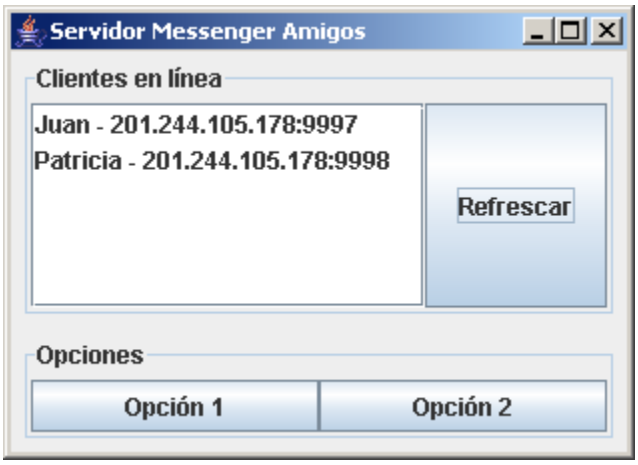
Dos amigos que estén conectados pueden establecer una conversación entre ellos. No es posible establecer conversaciones entre tres personas, pero si se pueden tener varias conversaciones al mismo tiempo.

Finalmente, es importante mencionar que la información sobre los usuarios y sus amigos se almacena en una base de datos relacional (Derby), a la que se tiene acceso desde la aplicación del servidor.

Interfaz del Cliente



Interfaz del Servidor



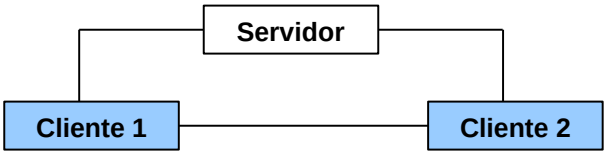
Análisis

Las principales acciones que puede realizar un usuario del sistema son:

- Conectarse al servidor
- Agregar un amigo
- Llevar a cabo una conversación
- Desconectarse

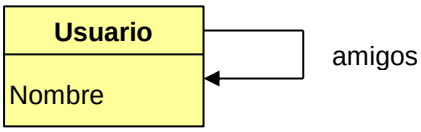
Mientras el usuario esté conectado, debe recibir notificaciones con el estado de sus amigos. Así mismo, cuando se conecte y se desconecte, se deben enviar notificaciones a las personas que lo conocen.

Un esquema básico de la comunicación es el siguiente:



Es necesario que el cliente le envíe cierta información al servidor y también es necesario que el servidor le envíe información a los clientes. Cuando dos clientes establezcan una conversación, tendrán que enviarse información entre ellos.

En este Messenger básico, cada usuario tiene muy poca información (solamente su nombre y sus amigos). Los usuarios se pueden modelar entonces de la siguiente manera.



Diseño de la Solución

Hay dos puntos principales a tener en cuenta para diseñar el sistema completo. Por una parte hay que decidir cómo se comunicarán las aplicaciones (clientes y servidor) entre sí y por otra hay que establecer un protocolo de comunicación. Este protocolo garantizará que los mensajes puedan ser interpretados correctamente por quien los recibe.

Arquitectura del Sistema

En este caso hay básicamente dos formas de implementar la comunicación que se descubrió que es necesaria durante el análisis del problema.

1. En la primera alternativa los clientes se comunican únicamente con el servidor. Cuando quiere establecerse una conversación entre dos clientes, esta se lleva a cabo a través del servidor. Esta solución permite que los clientes sean más sencillos, a costa de aumentar la complejidad del servidor.

2. En la segunda alternativa los clientes envían y reciben información y comandos del servidor, pero para llevar a cabo una conversación se establece una comunicación directa entre los dos clientes. Esta solución hace que el servidor sea más sencillo pero aumenta la complejidad de los clientes.

En este caso la alternativa escogida es la segunda. Si se hubiera escogido la primera, el servidor habría tenido que manejar los comandos básicos de los clientes y todas las conversaciones que se llevaran a cabo entre ellos. Con la segunda alternativa el servidor solamente maneja los comandos básicos de los clientes pero a costa de que ahora los clientes tengan que establecer comunicaciones adicionales.

Los clientes deberán comunicarse con el servidor para realizar las siguientes tareas básicas:

- Conectarse / Desconectarse

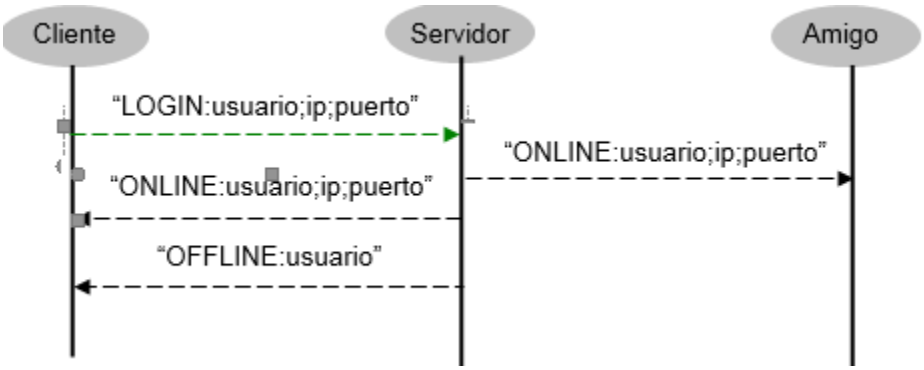
- Agregar un amigo

Adicionalmente, cuando un cliente quiera establecer una conversación deberá enviar un mensaje inicial a través del servidor para que el otro cliente sepa que se debe establecer una comunicación directa. Por ejemplo, si el cliente A quiere establecer una conversación con el cliente B, A enviará un mensaje al servidor que será reenviado a B. Luego A quedará a la espera de B, que con el mensaje enviado por el servidor sabrá que tiene que conectarse directamente al cliente A. Una vez B se haya conectado directamente a A la conversación se realizará solamente entre los dos y hasta que alguno de los dos envíe un mensaje de terminación.

Protocolo de Comunicación

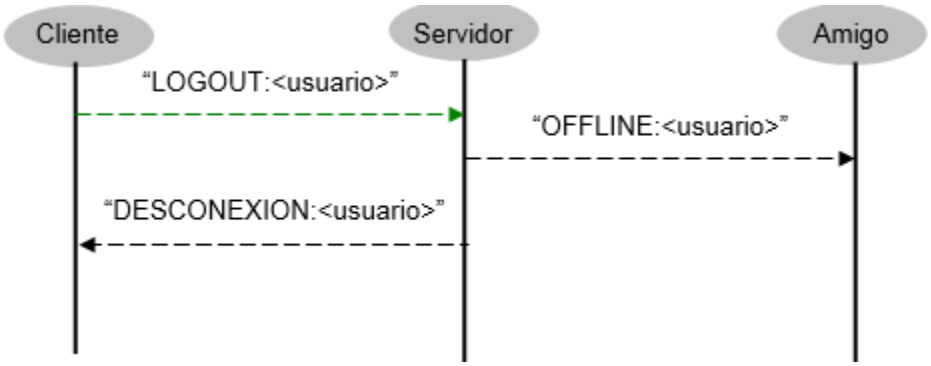
A continuación se presenta el protocolo de comunicación que establece cuales mensajes (y en qué orden) se deberán enviar para realizar cada una de las tareas del sistema. Es responsabilidad tanto del servidor como de los clientes ser capaz de interpretar este protocolo y realizar las tareas necesarias.

Conexión de un Cliente



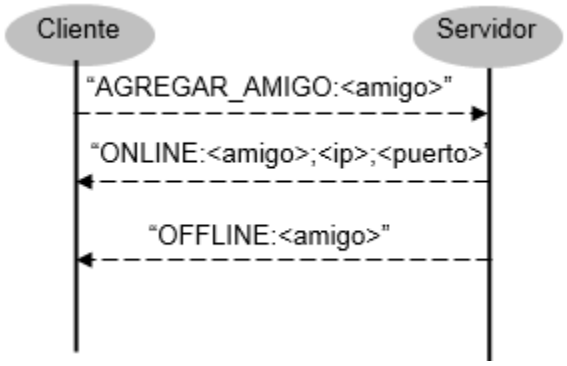
En el primer mensaje que le envía el cliente al servidor le indica cuál es su nombre, la dirección ip de su máquina y el puerto en el que recibirá las conexiones de los otros clientes. El servidor envía un mensaje a las personas que conocen al usuario recién conectado, indicando que ahora está ONLINE. Luego el servidor envía un mensaje al cliente recién conectado indicando el estado de cada uno de sus amigos: pueden ser mensajes ONLINE (para los que están conectados) y OFFLINE (para los que están desconectados).

Desconexión de un Cliente



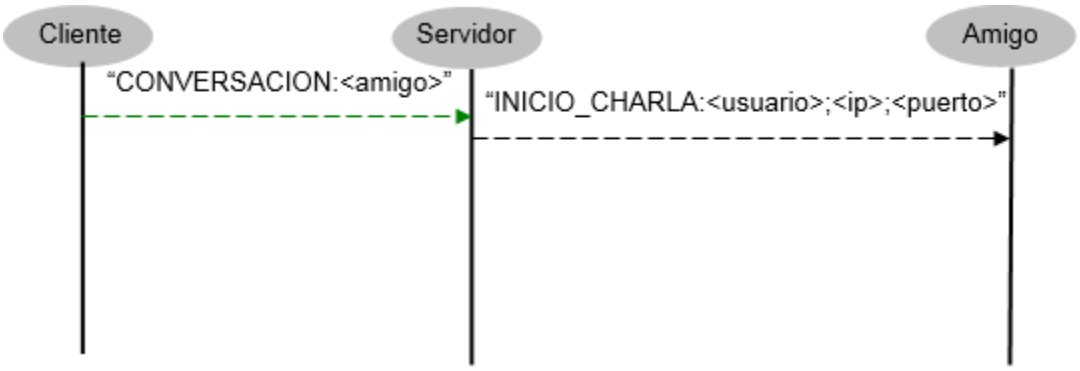
Cuando un cliente va a desconectarse, envía un mensaje de LOGOUT al servidor. Este notifica su desconexión a todas las personas que lo conocen y envía como respuesta al cliente un mensaje de DESCONEXION, que sirve como confirmación para saber que todo se realizó sin problemas.

Agregar un Amigo



Cuando un cliente quiere agregar a un amigo, envía un mensaje AGREGAR_AMIGO al servidor. Este recibe el mensaje, lo procesa y envía un mensaje en el cual informa sobre el estado actual del amigo (ONLINE u OFFLINE).

Iniciar una Conversación



Para iniciar una conversación un cliente envía un mensaje al servidor diciendo CONVERSACION y el nombre del amigo con el que quiere iniciar la conversación. El servidor a su vez envía un mensaje al amigo indicando INICIO_CHARLA y los datos del usuario al cual debe conectarse para establecer la conversación. Una vez que los dos clientes se han conectado, pueden enviarse todos los mensajes que quieran siempre y cuando estos empiecen por "MENSAJE:"

Cuando se quiere terminar la conversación uno de los clientes debe enviar un mensaje TERMINAR y el otro confirmará el fin de la conversación con un mensaje CONVERSACION_TERMINADA.

Diseño de los Clientes

Para diseñar los clientes es necesario tener en cuenta los siguientes puntos.

- Tiene que mantenerse una conexión permanente con el servidor.
- Podrían recibirse mensajes del servidor en cualquier momento.
- Se deben poder establecer conexiones adicionales para las conversaciones.
- Se pueden recibir mensajes de las conversaciones en cualquier momento.

Estas condiciones establecen restricciones importantes para la solución que será diseñada.

- Se deben esperar permanentemente mensajes del servidor.
- Se deben esperar permanentemente mensajes en las conversaciones activas.
- La interfaz no sólo se deberá actualizar cuando el usuario realice ciertas acciones, sino también cuando se reciban algunos mensajes del servidor o de otro cliente.

El diseño de los clientes que se plantea a continuación implementa los requerimientos funcionales de la aplicación y cumple con las restricciones planteadas y con las restricciones que establece el protocolo de comunicación.

Estructura de la Solución

El diseño de los clientes tiene que tener en cuenta tanto el mundo de la aplicación como la interfaz y sus relaciones. En el mundo tenemos principalmente la clase ClienteAmigos, que se encarga de los requerimientos funcionales básicos, y la clase Conversacion, que se encarga de mantener las conversaciones. A estas dos clases del mundo corresponden dos clases en la interfaz: a través de la clase InterfazClienteMessengerAmigos se realizan las principales tareas de la aplicación, delegando su realización a la clase ClienteAmigos; además, cuando se deben mostrar cambios que fueron informados por el servidor, la clase ClienteAmigos debe informarlea la clase InterfazClienteMessengerAmigos; la otra clase muuy importante de la interfaz es VentanaConversacion: para cada objeto conversación que haya en el mundo debe haber una ventana asociada, desde la cual se podrán enviar mensajes y en la cual se mostrarán los mensajes enviados y recibidos.

Además de estas las clases del mundo descritas, se tienen la clase ThreadRecibirMensajesServidor y ThreadRecibirMensajesConversacion. Cada una de estas clases realiza tareas en un hilo de ejecución aparte del principal. Así, con un objeto de la clase ThreadRecibirMensajesServidor es posible recibir permanentemente los mensajes enviados por el servidor para luego delegar la realización de alguna tarea a la clase ClienteAmigos. De forma similar, para cada conversación activa habrá un objeto de la clase ThreadRecibirMensajesConversacion, que se encargará de recibir los mensajes enviados por el otro cliente y delegar su procesamiento a la clase Conversacion.

Finalmente la clase Usuario se usa para mantener la información sobre un amigo (su estado, dirección y puerto).

Descripción de las Clases Principales

Nombre:	**ClienteAmigos**
Descripción:	Esta clase es la que permite la comunicación del cliente con el servidor e implementa las acciones que se deben realizar cuando se reciben mensajes del servidor. Cuando se quiere establecer una conversación con un amigo, esta es la clase que debe crear el objeto Conversacion. Esta clase tiene además la responsabilidad de mantener la colección de amigos con su estado actualizado, tanto en la representación interna como en la interfaz.
Asociaciones	
Usuario	La clase ClienteAmigos mantiene la información de los amigos del usuario que está conectado. Cada vez que el servidor envíe información sobre el estado de un amigo, se debe actualizar y guardar.
Conversacion	La clase ClienteAmigos debe saber cuáles conversaciones se están llevando a cabo en un momento dado. De esta forma puede cerrar estas conversaciones si se inicia el proceso de desconexión.
InterfazClienteMessengerAmigos	La clase ClienteAmigos tiene que poder avisarle a la interfaz cuando esta tenga que actualizarse. Los motivos para desencadenar una actualización pueden ser el cambio en el estado de uno de los amigos o la creación de una nueva conversación.
Responsabilidades:	Establecer y mantener la comunicación con el servidor. Realizar las acciones necesarias cuando se reciban mensajes del servidor a través de la clase ThreadRecibirMensajesServidor. Informar a la interfaz de los cambios en el estado de los amigos. Crear una conversación cuando el cliente local así lo requiera. Crear una conversación cuando se reciba una solicitud del servidor e informarle a la interfaz de la necesidad de crear una ventana para la conversación.

	Realizar las tareas necesarias para desconectar al cliente (cerrar la conexión al servidor y terminar las conversaciones en curso). Mantener actualizada la información sobre los amigos del usuario conectado.

Nombre:	**Conversacion**
Descripción:	Esta clase representa una conversación que se está llevando a cabo entre dos usuarios. A través de esta clase se pueden enviar mensajes al otro amigo; cuando la clase ThreadRecibirMensajesConversacion recibe mensajes, esta clase es la encargada de procesarlos.
Asociaciones	
ClienteAmigos	Es el cliente de la cual hace parte esta conversación. Cuando se termina una conversación es necesario informarle al cliente para que sepa que esta conversación ya no estará activa.
VentanaConversacion	La conversación necesita conocer la ventana de la conversación para poder actualizar los mensajes mostrados. Además, si el otro cliente termina la charla, la clase Conversacion tendrá que cerrar la ventana en consecuencia.
Responsabilidades:	Mantener el canal de comunicación con el otro cliente: si la conversación se inició localmente, al principio debe esperar a que el otro usuario se conecte; si la conversación la inició el otro cliente, esta clase debe establecer la conexión con él. Procesar los mensajes enviados y recibidos y actualizar la ventana en consecuencia. Terminar la conversación cuando sea necesario e informar tanto a la VentanaConversacion como al ClienteAmigos.

Nombre:	**InterfazClienteMessengerAmigos**
Descripción:	Esta es la clase principal de la interfaz. A través de ella se deben realizar las peticiones para conectar, desconectar, agregar un amigo o iniciar una conversación. Esta clase tiene que mantener la lista de amigos con su estado actualizado.
Asociaciones	
ClienteAmigos	La interfaz delegará a esta clase del mundo la realización de tareas como conectar, desconectar, agregar un amigo e iniciar una conversación.
Responsabilidades:	Delegar a la clase ClienteAmigos las tareas que requieran de comunicación con el servidor. Mantener actualizada la representación del estado de los clientes. Crear las ventanas para las conversaciones que establezca el cliente.

Nombre:	**VentanaConversacion**
Descripción:	Esta ventana muestra todos los mensajes de la conversación.
Asociaciones	
Conversacion	Es la conversación para la que se muestran los mensajes. Para enviar mensajes es necesario hacerlo a través de este objeto. Esta conversación le informará a la ventana cuando se reciban mensajes que deban ser mostrados.
Responsabilidades:	Enviar mensajes a través del objeto Conversación. Mostrar los mensajes enviados y recibidos. Solicitar la terminación de la conversación cuando la ventana es cerrada.

Verificación del Diseño de los Clientes

A continuación se mostrará cómo el diseño presentado permite cumplir con los requerimientos funcionales principales.

Requerimientos Funcionales Principales

Nombre:	**Conectar al servidor**
Clases Involucradas	
InterfazClienteMessengerAmigos	A través de esta clase el usuario hace la solicitud para conectarse.
ClienteAmigos	Esta clase establece la conexión al servidor y crear el objeto ThreadRecibirMensajesServidor que se encarga de recibir los siguientes mensajes del servidor. Desde que se establece la conexión, para cada mensaje que informa del estado de un amigo la clase ClienteAmigos le envía una notificación a la interfaz para que se actualice la información mostrada.
Secuencia de Acciones	
1. InterfazClienteMessengerAmigos.iniciarConexion() 2. ClienteAmigos.conectar(String) 3. ThreadRecibirMensajesServidor.start();	
Comentarios:	La responsabilidad de crear y mantener la conexión está en la clase ClienteAmigos, así como la responsabilidad de informar a la interfaz de los cambios que haya en el estado de los amigos. No se sabe en qué momento el servidor enviará mensajes al cliente. La existencia de una clase especializada únicamente en recibir los mensajes hace mucho más sencilla a la clase ClienteAmigos. Si el cliente no pudiera comunicarse con la interfaz, sería necesario que periódicamente esta averiguara acerca de los cambios que se pudieran haber presentado.

Nombre:	**Desconectar del Servidor**
Clases Involucradas	
InterfazClienteMessengerAmigos	A través de esta clase el usuario indica que quiere desconectarse.
ClienteAmigos	Esta clase le indica a todas las conversaciones que deben terminar y termina la conexión con el servidor.
Conversacion	La conversación debe terminar, así que se envía un mensaje a la otra persona que participa en ella.
VentanaConversacion	Cuando se termina la conversación también debe cerrarse la ventana asociada.
Secuencia de Acciones	
1. InterfazClienteMessengerAmigos.terminarConexion() 2. ClienteAmigos.enviarDesconexion()	

3. Cuando se recibe el mensaje DESCONEXION se llama el método ClienteAmigos.desconectar( )
   Para cada conversación activa.

4. Conversacion.terminar()

5. Cuando se recibe el mensaje CONVERSACION_TERMINADA entonces se llama terminarConversacion() y se le informa a la ventana que debe cerrarse (VentanaConversacion.dispose( ))

6. Se termina la conexión con el servidor.

Comentarios:	La iniciativa de terminar cada conversación viene del objeto Conversacion, y no de la interfaz: por esto es necesario que cada objeto Conversacion pueda informarle a su respectiva VentanaConversacion que es necesario cerrarse. Cuando se inicia la desconexión se envía un mensaje (LOGOUT) al servidor y ya. Cuando se recibe el mensaje DESCONEXION es que se inician las tareas de terminar las conversaciones y cerrar la conexión con el servidor.

Nombre:	**Iniciar una conversación (local)**
Clases Involucradas	
InterfazClienteMessengerAmigos	A través de este objeto el usuario hace la solicitud para iniciar una conversación.
ClienteAmigos	Este objeto crea un objeto de tipo Conversacion para que maneje la nueva conversación y envía un mensaje al servidor solicitándole que le envíe un mensaje (INICIAR_CHARLA) a otro cliente.
Conversacion	Cuando se construye este objeto queda lista para recibir la conexión del cliente con el que se va a establecer una conversación.
VentanaConversación	Esta ventana se encargará de enviar y mostrar los mensajes de la conversación.
ThreadRecibirMensajesConversacion	Este objeto se encarga de recibir los mensajes enviados por el otro participante en la conversación y delegar su procesamiento a la clase Conversacion.

Secuencia de Acciones

1. La interfaz recibe una solicitud para crear una nueva conversacion y la delega al ClienteAmigos.

2. El ClienteAmigos envía un mensaje al servidor con la solicitud, crea el objeto Conversacion y lo retorna.

3. El objeto Conversacion creado espera la conexión del otro cliente. Cuando la recibe crea una instancia de ThreadRecibirMensajesConversacion para que se encargue de esperar los mensajes que el otro usuario envíe.

4. Con el objeto Conversacion retornado, la Interfaz crea una nueva VentanaConversacion y la asocia a la nueva conversación.

Nombre:	**Iniciar una conversación (remota)**
Clases Involucradas	
InterfazClienteMessengerAmigos	En este caso esta clase solamente se encarga de crear un objeto de tipo VentanaConversacion.
ClienteAmigos	Este objeto recibe el comando para iniciar una convesación: crea el objeto Conversacion y le indica que debe conectarse a un cliente remoto. Luego de

	construir el objeto Conversacion le indica a la interfaz que debe construir una ventana asociada a él.
Conversacion	Cuando se construye este objeto se conecta al cliente remoto.
VentanaConversación	Esta ventana se encargará de enviar y mostrar los mensajes de la conversación.
ThreadRecibirMensajesConversacion	Este objeto se encarga de recibir los mensajes enviados por el otro participante en la conversación y delegar su procesamiento a la clase Conversacion.

Secuencia de Acciones

1. El ClienteAmigos recibe un mensaje que le indica que se va a crear una conversación: crea el objeto Conversacion y le indica que debe conectarse al cliente remoto. Luego le indica a la interfaz que debe crear una ventana y le da el objeto Conversacion al cual debe estar asociado.

2. El objeto Conversacion creado establece una conexión al otro cliente: crea una instancia de ThreadRecibirMensajesConversacion para que se encargue de esperar los mensajes que el otro usuario envíe.

3. Con el objeto Conversacion que le pasaron, la Interfaz crea una nueva VentanaConversacion y la asocia a la nueva conversación.