Enrique Kessler Martínez
Juan Jesús Herrero Navarro
GRUPO 1.1

DOCUMENTO DE DISEÑO

ÍNDICE

INTRODUCCIÓN

Este es el documento de diseño de nuestro proyecto de Protocolo de Chat. Además de las mínimas implementaciones, hemos decidido implementar las mejoras: la creación de sala en el caso de que no exista al intentar acceder a ella.

FORMATO DE LOS MENSAJES

Este es un resumen de los formatos utilizados para todos los mensajes de conexión entre los distintos módulos implementados (Servidor, cliente y directorio). Visto que estamos utilizando el formato FV, usamos:

- Quit:

Operation: quit\n
\n

Esta instrucción utiliza un formato simple solo incluyendo el Opcode, de forma que contiene un byte de información donde está almacenado el Opcode.

- Nick:

Operation: nick\n
Name: <nombre>\n
\n

Para el envío de este mensaje de cliente a servidor, hace uso del formato de mensaje de arriba, conteniendo un opcode (1 byte) y el nick en cuestión (4 bytes), como por ejemplo:

> nick Qkessler

Operation: nick\n
Name: Qkessler\n
\n

Para la respuesta por parte del servidor, hace uso de un formato simple de Opcode (1 byte), con la repuesta OK, o duplicated.

Operation: Nick_OK\n
\n

Operation: Nick_DUPLICATED\n
\n

Para este ejemplo, ha aceptado el nick Qkessler. Si nos hubiera devuelto el segundo mensaje, significaría que tendría que intentar registrarme con otro nombre, ya que este ya está en el sistema.

- RoomList:

Para el envío por parte del cliente, el formato es un mensaje simple de opcode, pidiendo la lista de salas.

Operation: getRoomList\n
\n

> roomlist

En cambio, el servidor devuelve un mensaje de formato diferente(TLV), ya que debe contener el opcode de la operación de vuelta, la longitud de los campos que va a almacenar (no sabemos la cantidad de nombres de Salas que tiene que almacenar, de forma que el campo longitud nos indica cuanto espacio guardar) y la información de la lista de las salas que está almacenada en un campo de “longitud” bytes.

Operation: sendRoomList \n
Room Name: <name> \n
Members length: <length> \n
Members: <members names> \n
Time Last Message: <time> \n
...
\n

En este caso, los puntos suspensivos expresan que como existe la posibilidad de haber más salas, se repetirán los 4 campos.

> roomList
> Room Name: RoomA Members(0): Time Last Message : not yet

- Enter:

Para el envío por parte del cliente hacia el servidor, el cliente envía un opcode (1 byte)y el nombre de la sala a la que está solicitando entrar (4 bytes), por tanto, el formato será:

Operation: enter \n
Name: <nameRoom> \n
\n

El servidor, por su parte, envía al cliente, un mensaje opcode de confirmación de si ha conseguido entrar en la sala, o no (1 byte).

Operation: enter_True \n
\n

Operation: enter_False \n
\n

> (NanoChat)enter sala1
> You are in a room, be kind to people.
>(NanoChat-sala1)

- Info:

Para el envío por parte del cliente utiliza el formato simple de un mensaje de Opcode, de tamaño 1 byte. En cambio, el servidor envía la información de la sala en la que está el cliente, de forma que cuenta con más campos de información.

Operation: getRoomInfo \n
\n

Operation: sendRoomInfo \n
Room Name : <roomname> \n
Members length: <length> \n
Members: <members names> \n
Time Last Message: <time> \n
\n

Contiene RoomName, que será un campo de 4 bytes, una lista de Members, que por ser un campo de longitud variable, vamos a tener que incluir un campo más de longitud de 1 byte. Por último, Last Message que contendrá 4 bytes, por ser de tipo long.

> info
> Room Name: RoomA Members(1): Juanje Time Last Message: not yet

- Send:

Envía un mensaje al chat de una sala en concreto, por parte del cliente. Utilizará un formato que contiene un opcode (1 byte), el tamaño de longitud del mensaje (1 byte) y por último un campo de mensaje que será de tamaño indicado por el campo longitud.

Operation: sendChat \n
Text : <mensaje> \n
\n

> send Hola, ¿qué tal?

En el caso de que recibamos un mensaje:

> (nanoChat-a) * Message received from server...
a: Hola

- Exit:

Comando ejecutado por el cliente, que hace que salga de la sala en la que se encuentra. Utiliza un opcode (1 byte).

Operation: exit \n
\n

> exit

- Consulta:

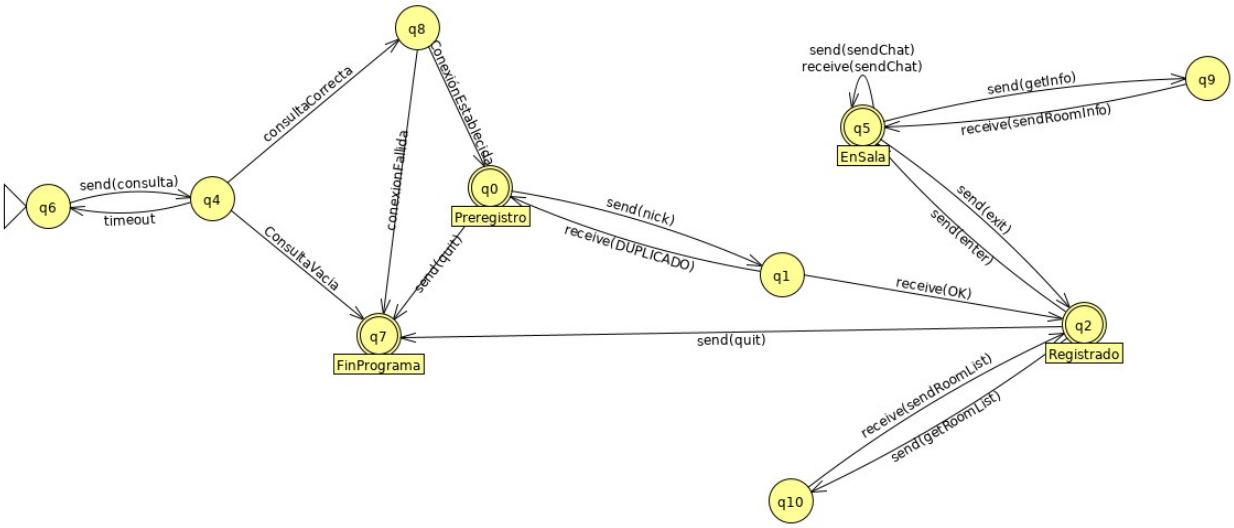
Función del cliente para obtener el servidor asociado, contiene un campo opcode (1 byte) y un campo de protocolo de 4 bytes. Es utilizada no en el shell, sino por el propio programa. Devuelve, o consulta vacía (opcode de 1 byte) o un opcode de respuesta de la consulta (1 byte).

- Registro:

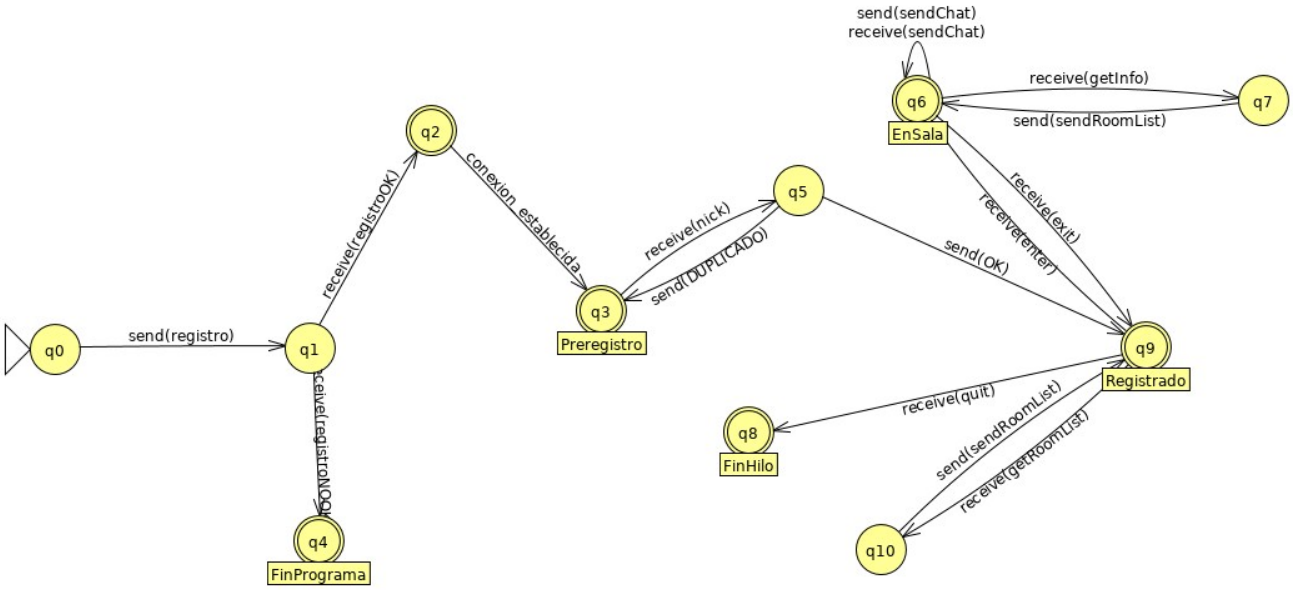
Solicitud para registrar un servidor de chat, asociado a un determinado protocolo. Contiene un opcode registro (1 byte), el protocolo (4 bytes) y el puerto (4 bytes). Devuelve registro_OK (opcode 1 byte) o registro_NO_OK (opcode 1 byte).

AUTÓMATAS

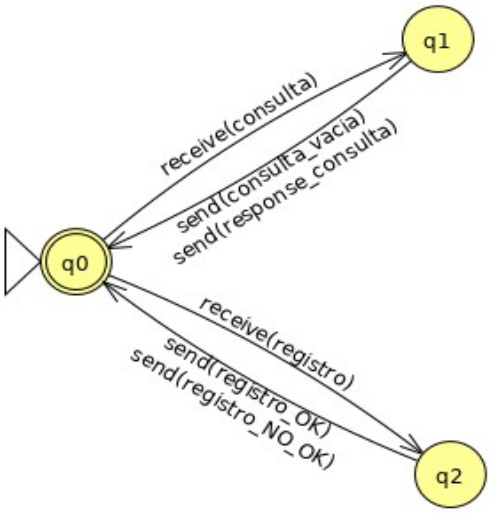
- Autómata de las conexiones del **cliente** de chat:



- Autómata del **servidor** de chat:



- Autómata del **Directorio**.



CONCLUSIÓN

Este proyecto ha resultado interesante para el trabajo con distintos protocolos de transporte, como puede ser UDP y TCP. Además, nos hace tener un primer contacto con un proyecto grande, al que le hemos dedicado mucho tiempo, incluso aunque trabajemos únicamente con clases ya definidas por profesores. Nos ha parecido una buena idea como proyecto, y animamos a que se siga realizando en próximos años.