

Enrique Kessler Martínez  
Juan Jesús Herrero Navarro  
GRUPO 1.1

DOCUMENTO  
DE  
DISEÑO

# ÍNDICE

Índice .....	2
Introducción .....	3
Formato de los mensajes .....	4, 5, 6, 7
Autómatas .....	8, 9
Conclusiones .....	10

## INTRODUCCIÓN

Este es el documento de diseño de nuestro proyecto de Protocolo de Chat. Además de las mínimas implementaciones, hemos decidido implementar las mejoras: Obtener el historial de mensajes de los usuarios de una sala, tener la posibilidad de renombrar una sala, y por último, notificar entradas y salidas de las salas de los usuarios.

## 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\nNick: <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\nNick: Qkessler\n\n
```

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

```
Operation: OK\n\n
```

Para este ejemplo, ha aceptado el nick Qkessler.

### - RoomList:

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

```
Operation: roomList\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: roomList \n
RoomList: <roomList> \n
\n
```

```
> roomList
> [ sala1 , sala2 , sala3]
```

- 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
Room: <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: OK \n
\n
Operation: ERROR \n
\n
```

```
> enter sala1
> Ha conseguido entrar en la sala. // Si el opcode recibido es OK.
```

- 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: info \n
\n
Operation: info \n
RoomName : <roomname> \n
Members: <listademembers> \n
LastMessage: <time> \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
> sala1
    pepito, juanje, quique, mickael
    3
```

- 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.

```
Opcode: send \n
Mensaje : <mensaje> \n
\n
```

```
> send "Hola, ¿qué tal?"
```

- Exit:

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

```
Opcode: exit \n
\n
```

```
> exit
```

- Rename:

Comando que cambia el nombre de una sala en la que se encuentra el usuario. Se trata de un formato de varios campos: opcode (1 byte), un campo longitud (1 byte), y por último un campo con el nuevo nombre de la sala (tamaño indicado por el campo longitud).

```
Opcode: rename \n
NombreNuevo : <nuevoNombre> \n
\n
```

```
> rename "salanueva"
```

- Historial:

Comando que imprime los mensajes enviados anteriormente en una sala. Para el mensaje de cliente a servidor, se trata de un formato con el campo opcode (1 byte).

```
Opcode: historial \n
\n
```

El servidor, al recibir este, envía un mensaje de vuelta, que contiene un opcode (1 byte), un campo de longitud (1 byte) y por último campos de tamaño indicada por el cambio longitud.

```
Opcode: historial \n
ListaMensajes : <listaMensajes> \n
\n
```

```
> historial
> “Hola, ¿qué tal estás Pepito? ” - José.
    “Hola José, estoy bien” - Pepito.
    “Me alegro por ti, Pepito” - José.
```

- Consulta:

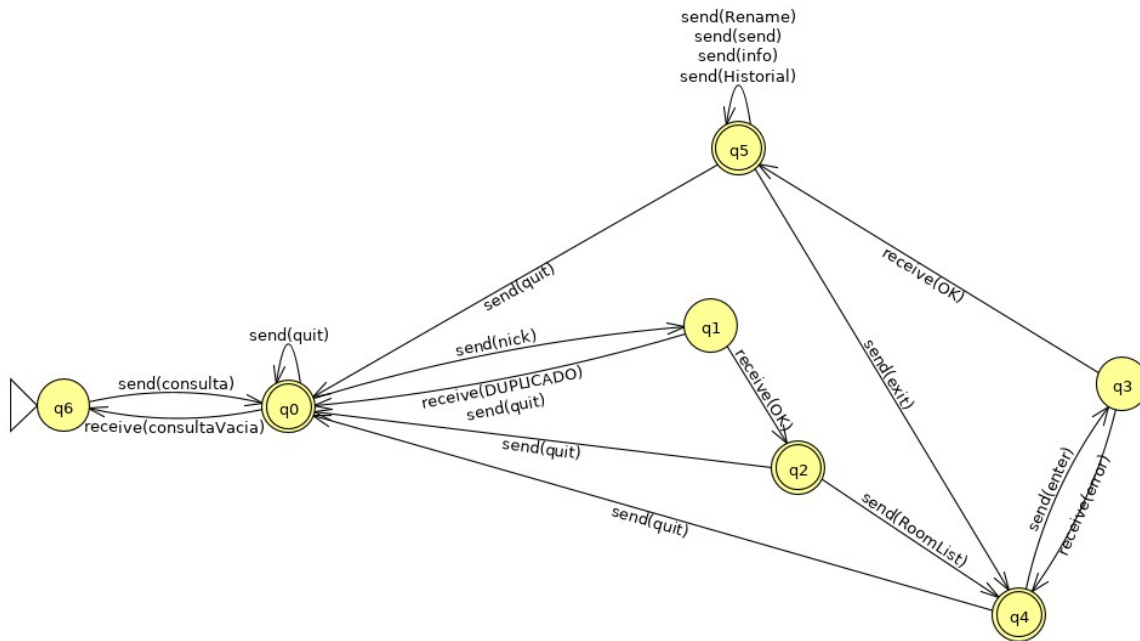
Función del cliente para obtener el servidor asociado, contiene un campo opcode (1 byte) y un campo de protocolo de 4 bytes. Es utilizada no en el shell, sino por el propio programa. Devuelve, o consulta vacía (opcode de 1 byte) o un opcode de respuesta de la consulta (1 byte).

- 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).

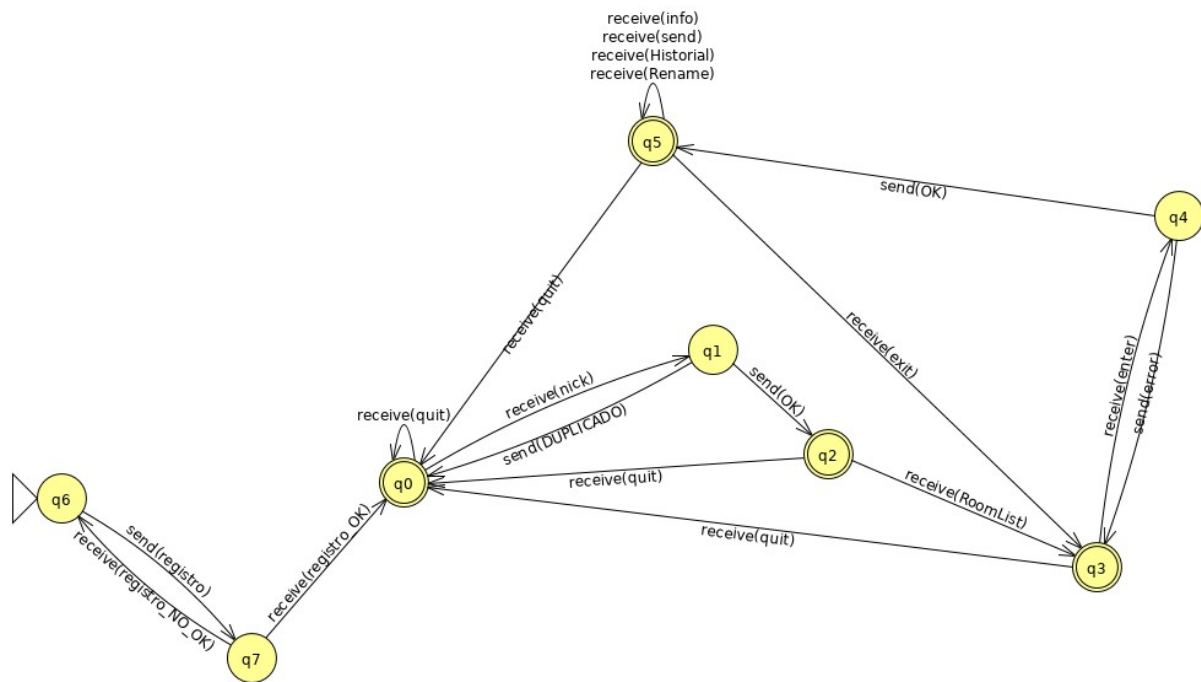
# AUTÓMATAS

- Autómata de las conexiones del cliente de chat:

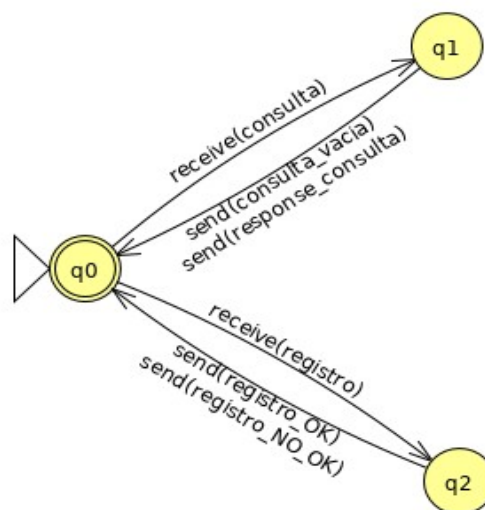




- Autómata del servidor de chat:



- Autómata del Servidor de 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.

