Universidad ORT Uruguay

Facultad de Ingeniería

Bernard Wand Polak

Programación de Redes

Obligatorio 1 - “Tiracaja”

Entregado como requisito para la obtención del crédito Programación de Redes

Ignacio Piccininno 177753

Verónica Seoane 172103

Grupo N6A

Docentes: Luis Barrague y Roberto Assandri

Año 2017

Índice

Objetivo

El objetivo de este documento es mostrar brevemente el funcionamiento de Tiracaja así como también se comentará su arquitectura y diseño a grandes rasgos así mismo como las decisiones tomadas y los supuestos a tener en cuenta.

Alcance de Tiracaja

El proyecto Tiracaja, consta de dos aplicaciones de consola. Por un lado la aplicación cliente y por otro lado la aplicación del servidor.

El cliente se conectará al servidor para poder hacer los pedidos que se indican en la letra del obligatorio (listar archivos, listar usuarios, descargar archivos, cargar archivos y notificar usuarios que un archivo está subido).

El servidor aceptará un máximo de x clientes conectados simultáneamente.

Decisiones y Supuestos

- El nombre del usuario era una forma de identificar al cliente, contra el servidor.
- También se supuso un protocolo de trama (*frame*) específico para la comunicación con el servidor.
- La cantidad máxima de clientes conectados concurrentemente es de x.
- Los archivos a subir o bajar deberán tener un tamaño menor a 10 MB.
- Tanto el puerto del servidor, las direcciones de IP necesarias y las rutas de los archivos, será almacenada en un archivo de configuración (*App.Config*).
- Para que la aplicación funcione correctamente el servidor debe ser iniciado antes que los clientes, sin embargo, si un cliente se quiere conectar y el servidor está desconectado, se le enviará un mensaje de error y se le cerrará la ventana.
- Si el nombre del archivo a descargar o cargar ya existe, se le agregará un número al final del nombre del archivo, para evitar que se sobre escriban archivos.
- Se utilizó *Sockets* para el desarrollo ya que se comenzó a avanzar desde el inicio y preferimos seguir con ello en vez de involucrarnos con *TCPListener* y *TCPClient*.

Arquitectura y Diseño

Utilización de protocolo

El equipo decidió aplicar el consejo que brinda la letra del obligatorio, y decidimos utilizar el protocolo designado con algunas variantes.

Este es el protocolo que ofrece la letra:

Nombre Del Campo	**HEADER**	**CMD**	**LARGO**	**DATOS**
Valores	RES/REQ	0-99	Entero	Variable
Largo	3	2	4	Variable

En el protocolo elegido por nosotros enviaremos:

Nombre del Campo	CMD	DATALENGTH	DATA
Valores	0-99	Entero	Variable
Largo	2	4	Variable

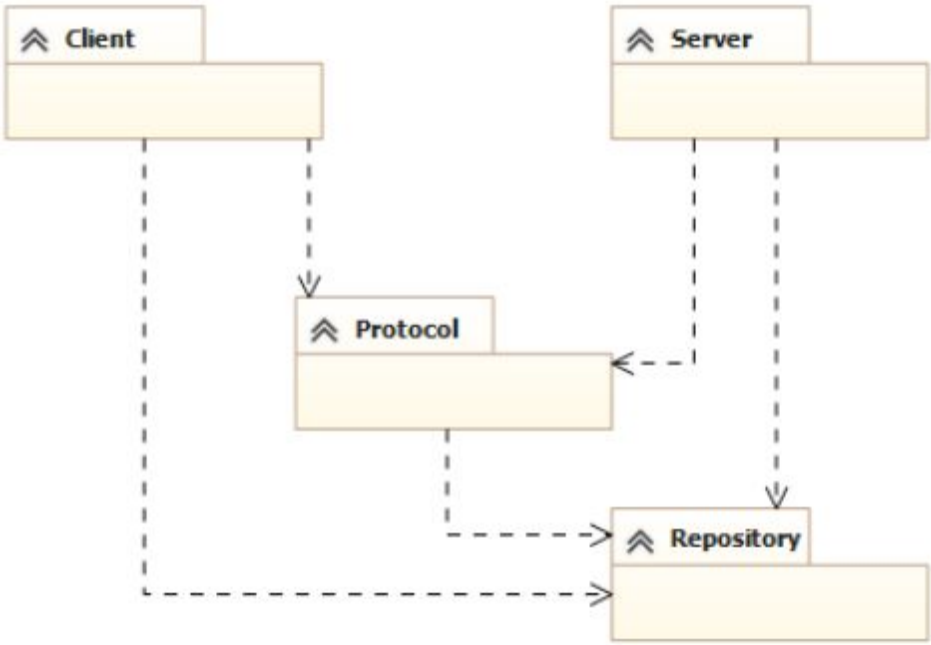
La trama está compuesta por el comando (CMD), los datos (Data) que es un *array* de *bytes*, y un entero que indica el largo de los datos (DataLength).
Para manipular las tramas, tenemos una clase llamada *FrameUtil*, ésta posee los métodos que permiten enviar y recibir datos.
Los comandos existentes son:
Connect, que permite conectar el cliente con el servidor.
ListFile, que permite listar los archivos del servidor.
ListUser, muestra los usuarios conectados.
UploadFile, sirve para que el cliente cargue un archivo en el servidor.
DownLoadFile, sirve para que el cliente descargue un archivo en el servidor.
Notify, sirve para notificar a un cliente dado con opción de enviar a todos, sobre la subida de un archivo.

Diagrama de clases y paquetes

Diagrama de Paquetes

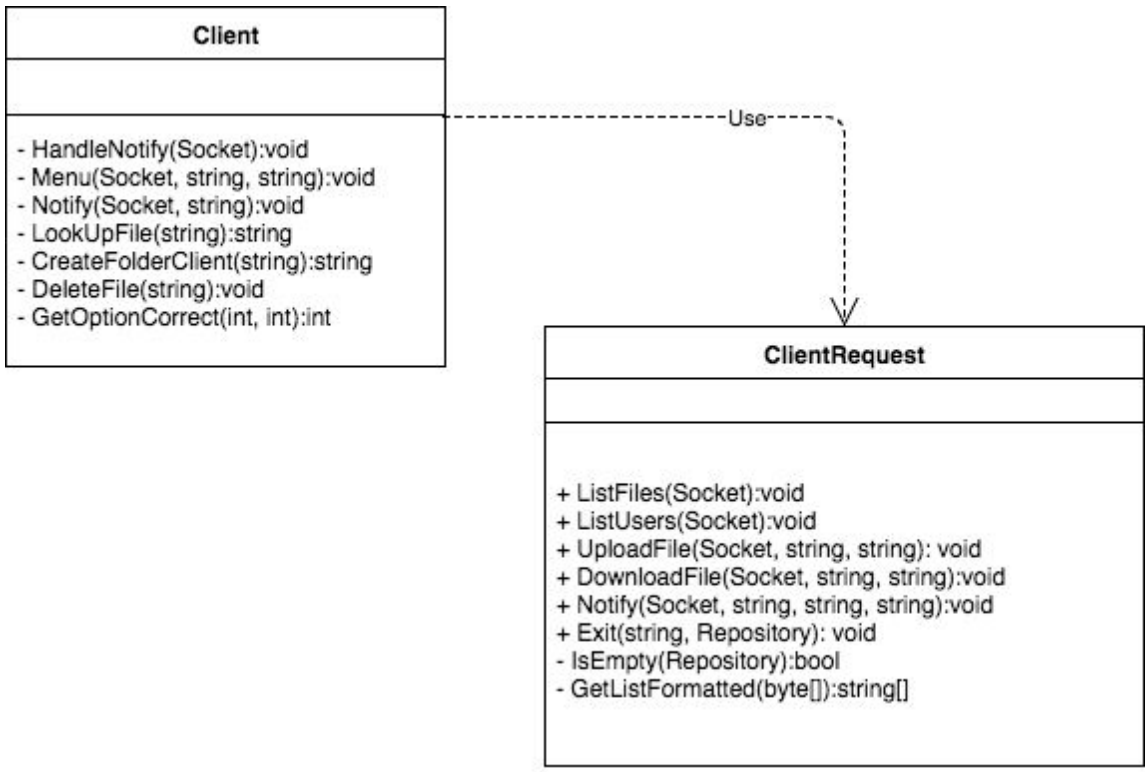
Proyecto Tiracaja

En el paquete cliente se encuentran las clases Client y ClientRequest, En el client se encuentra el menú que interactúa con el usuario que utiliza la aplicación, éste llama a los métodos que se encuentran en ClientRequest, encargado de la lógica pesada por parte del Client. Así mismo en el paquete Server se encuentran Server y ServerRequest que cumplen la misma función respectivamente a Client y ClientRequest.
Protocol es el paquete donde se encuentra definida la trama (Frame) que allí se encuentran definidos los comandos que se pueden usar, representantes de las funcionalidades del sistema. Luego dentro de ese paquete se encuentra el FrameUtils que se encuentran los métodos Send, Receive,

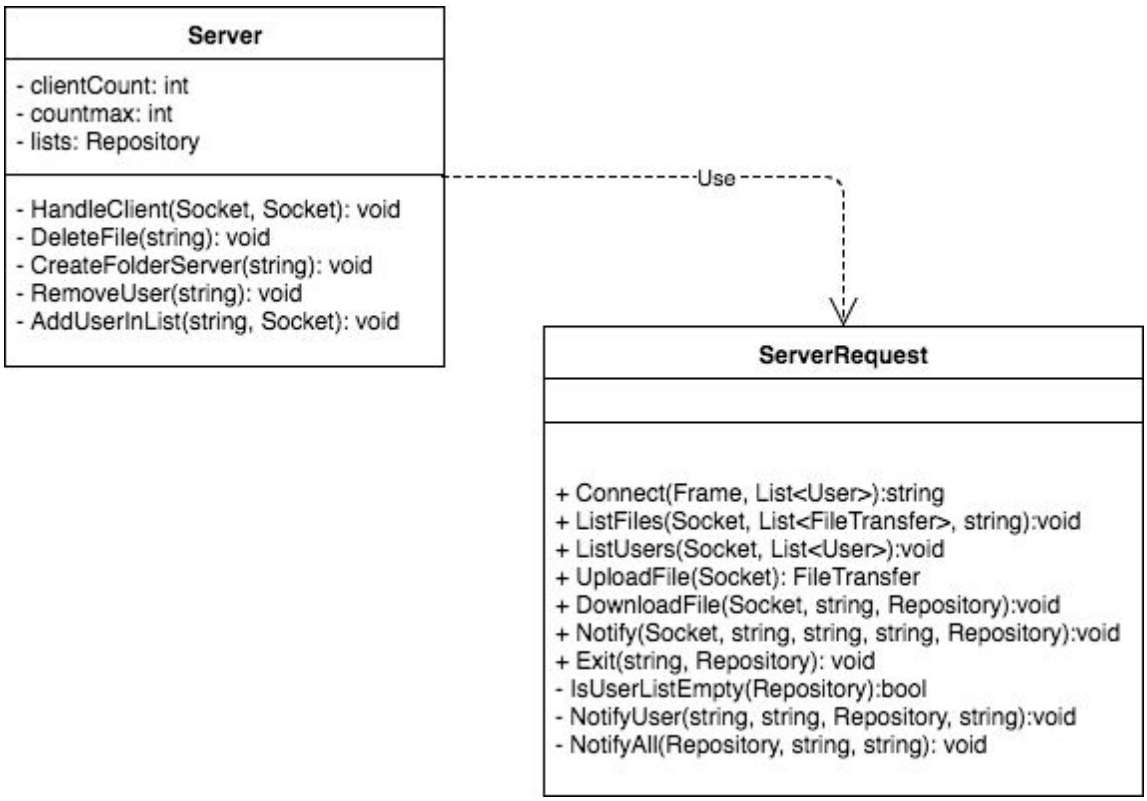


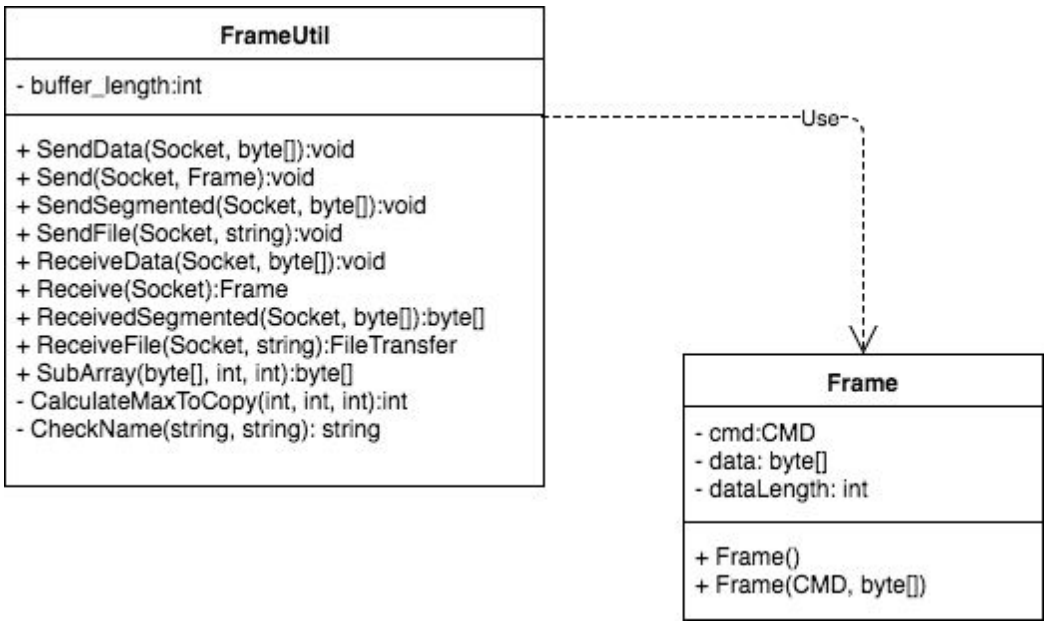
Diagramas de Clases

Client



Server


Server
- clientCount: int
- countmax: int
- lists: Repository
- HandleClient(Socket, Socket): void
- DeleteFile(string): void
- CreateFolderServer(string): void
- RemoveUser(string): void
- AddUserInList(string, Socket): void
Use
ServerRequest
+ Connect(Frame, List<User>):string
+ ListFiles(Socket, List<FileTransfer>, string):void
+ ListUsers(Socket, List<User>):void
+ UploadFile(Socket): FileTransfer
+ DownloadFile(Socket, string, Repository):void
+ Notify(Socket, string, string, string, Repository):void
+ Exit(string, Repository): void
- IsUserListEmpty(Repository):bool
- NotifyUser(string, string, Repository, string):void
- NotifyAll(Repository, string, string): void


FrameUtil
- buffer_length:int
+ SendData(Socket, byte[]):void
+ Send(Socket, Frame):void
+ SendSegmented(Socket, byte[]):void
+ SendFile(Socket, string):void
+ ReceiveData(Socket, byte[]):void
+ Receive(Socket):Frame
+ ReceivedSegmented(Socket, byte[]):byte[]
+ ReceiveFile(Socket, string):FileTransfer
+ SubArray(byte[], int, int):byte[]
- CalculateMaxToCopy(int, int, int):int
- CheckName(string, string): string
Use
Frame
- cmd:CMD
- data: byte[]
- dataLength: int
+ Frame()
+ Frame(CMD, byte[])

Repository

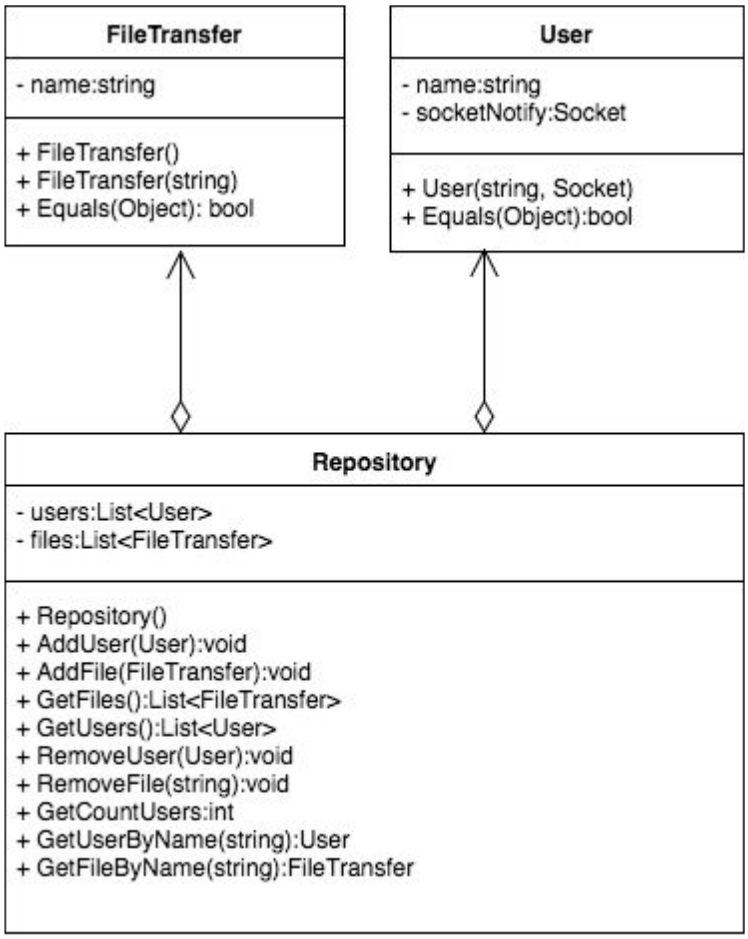
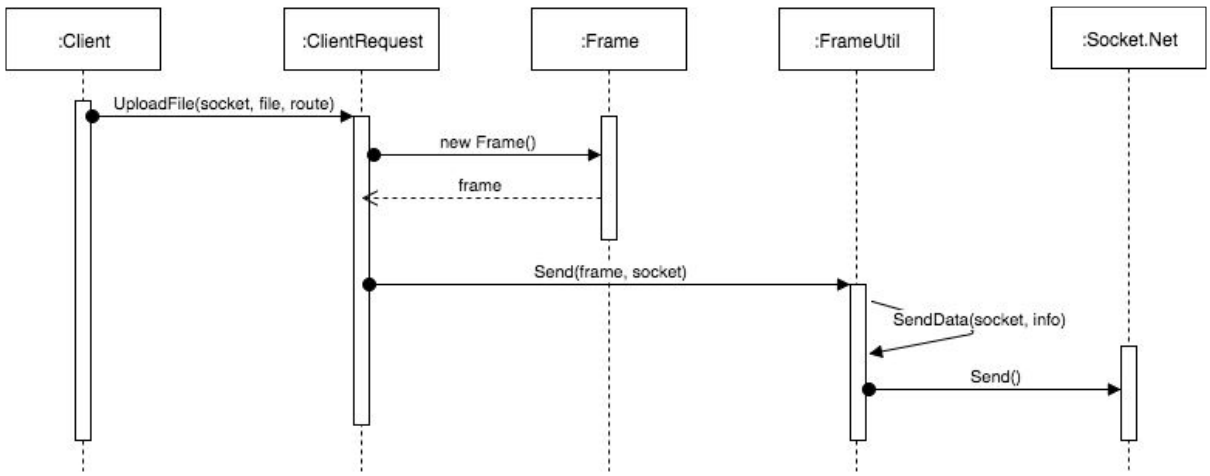

FileTransfer
- name:string
+ FileTransfer()
+ FileTransfer(string)
+ Equals(Object): bool
User
- name:string
- socketNotify:Socket
+ User(string, Socket)
+ Equals(Object):bool
Repository
- users:List<User>
- files:List<FileTransfer>
+ Repository()
+ AddUser(User):void
+ AddFile(FileTransfer):void
+ GetFiles():List<FileTransfer>
+ GetUsers():List<User>
+ RemoveUser(User):void
+ RemoveFile(string):void
+ GetCountUsers:int
+ GetUserByName(string):User
+ GetFileByName(string):FileTransfer

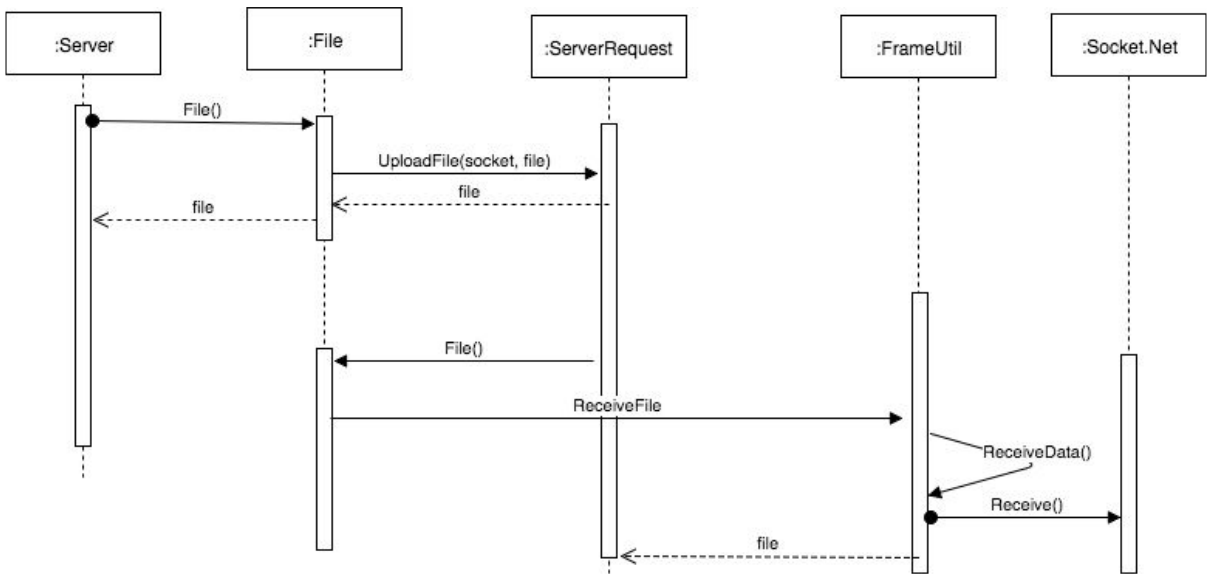
Diagrama de secuencia

Para el caso de diagramas de secuencia mostraremos básicamente de qué forma se envían datos, para ello elegimos representar la carga de archivos y las notificaciones.

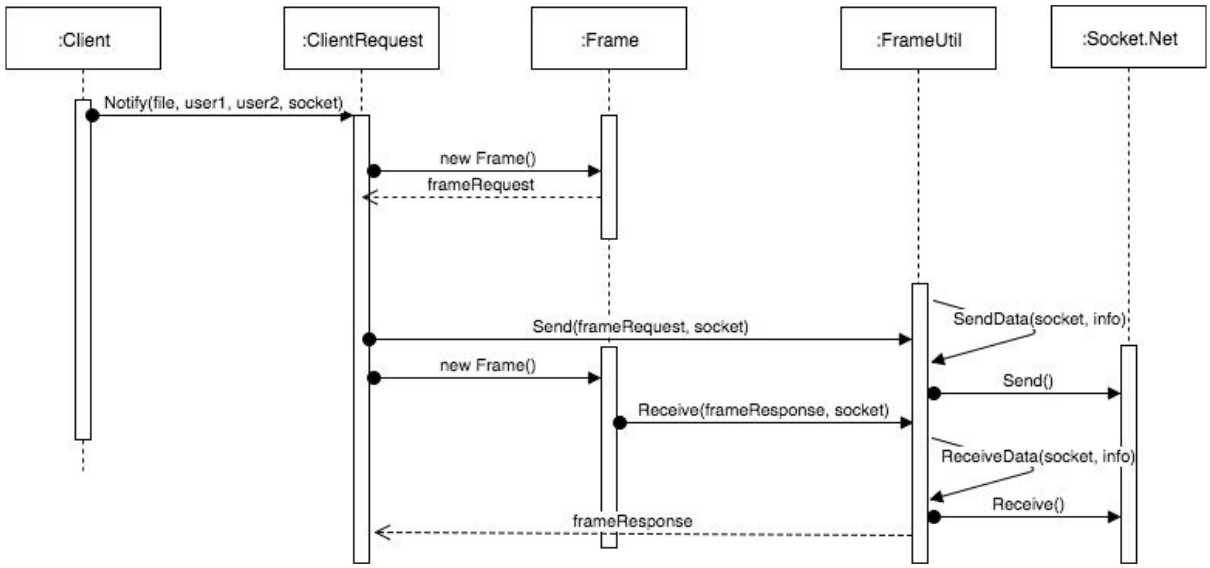
Upload File - Client



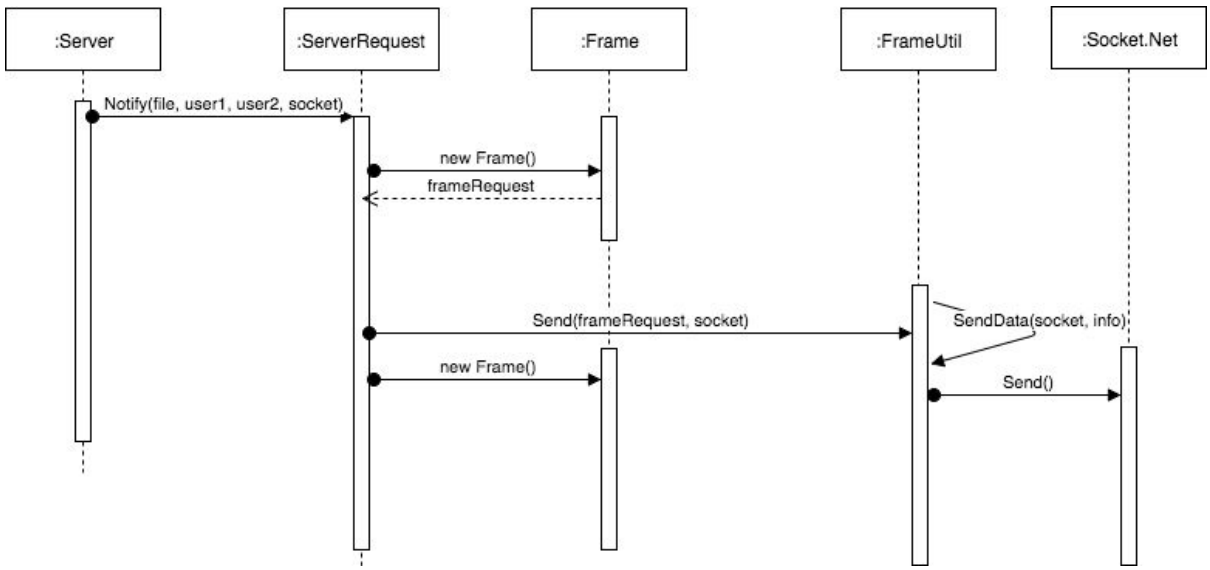
Upload File - Server



Notify - Client



Notify - Server



Otro caso podría ser:

Listar Archivos