Introducción

El diseño del trabajo practico esta dividido en cuatro partes principales:

1. Instrucciones del usuario
2. Cliente
3. Servidor
4. Juego (Sudoku)

A grandes rasgos, podemos agrupar los TDAs en dos grandes categorías, las encargadas de recibir las instrucciones del usuario, procesarlas y enviarlas por medio del cliente y los TDAs encargados de recibirlas por medio del servidor y procesar una respuesta.

En las siguientes secciones del informe se explicara con mayor detalle el momento en que cada instancia es creada.

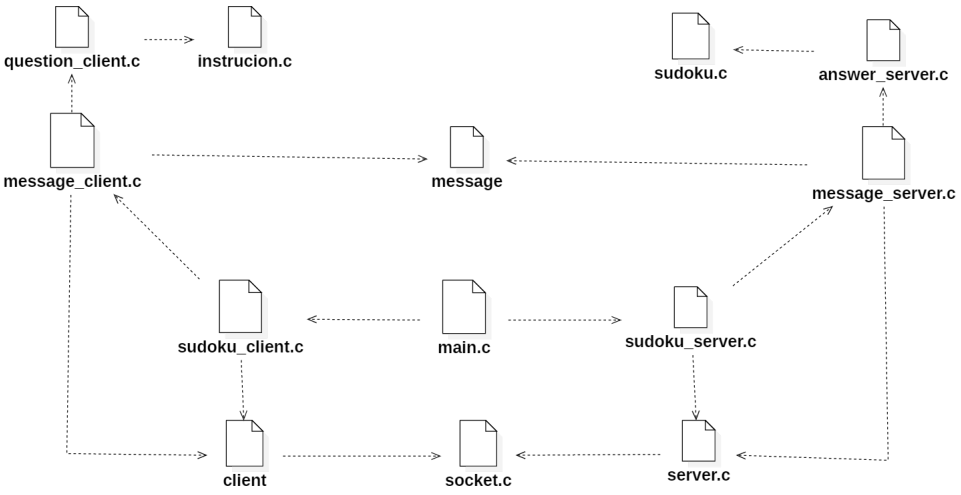

Figure 1: TDAs relacionadas al servidor

Ejecución Modo Servidor

Al iniciar el programa en modo servidor, el TDA **sudoku_server** inicializa además del servidor, una instancia (y unica) del juego y el TDA **server_message**, que es el medio por el cual se le envía la información al servidor para que luego sea enviada al cliente y también en se reciben los mensajes enviados por cliente al servidor.

La inicialización del Sudoku consiste básicamente en instanciar un parser con el archivo que contiene el los números del tablero e iterar con el para obtener cada uno. Cada numero se utiliza para crear una instancia de una variable del tipo **square_t**, contenidas en un vector del TDA **board**. Si bien un tablero es una matriz, este se represento como un vector de casilleros, por una cuestion de eficiencia. Cada elemento del tipo **square_t** posee un atributo booleano que que permite identificar si es un elemento fijo, avitando asi que el usuario tenga la capacidad de modificarlo o que este se borre al momento de querer reiniciar el tablero.

Durante la inicialización del servidor se instancia un el TDA **server** y se realiza una llamada a su metodo *server_listen*, que realiza un *binding* y accepta a la la conexion con el cliente cuando este esté disponible.

Finalmente, se inicializa el **server_message** y se le pasa un puntero al servidor, evitando asi de estar constantemente enviándolo como parámetro.

Una vez inicializado el juego y el server, el cliente puede ingresar y conectarse para comenzar a enviar instrucciones y recibir las respuestas.

En la figura abajo, se muestra como quedan las relaciones luego de la finalización del proceso de inicialización

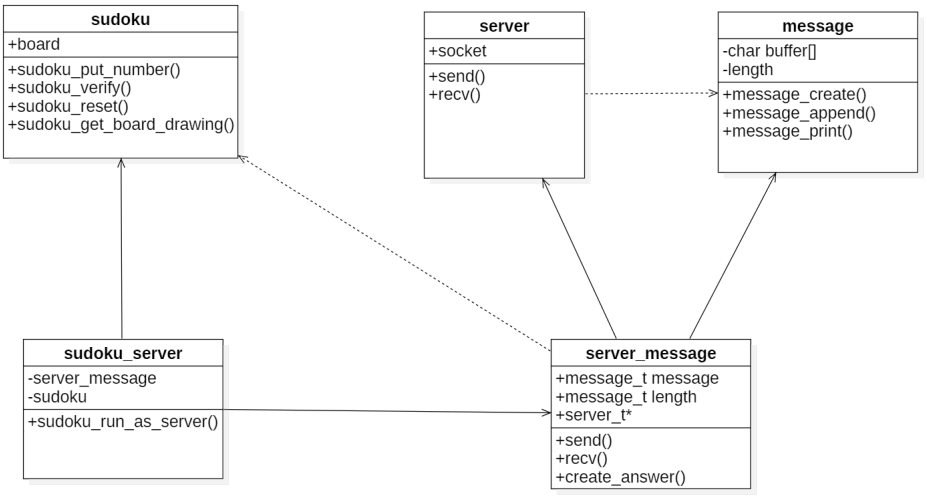

Figure 2: TDAs relacionadas al servidor

Ejecución Modo Cliente

Al momento en que un usuario entra en modo cliente, el juego ya esta inicializado y listo para empezar a recibir instrucciones. La funcion del TDA **sudoku_client** es inicializar el TDA **instruction**, el TDA **client_message** y obviamente el TDA **client**.

El modulo **instruction** es el encargado de leer la instrucción por entrada estándar dada por el usuario y darle la primera validacion, es decir, corroborar que los parámetros pasados se correspondan con una instrucción valida.

La funcion del TDA **client_message** es muy similar a la funcion del server_message pero del lado del cliente, y al igual que en caso anterior, este TDA también recibe un puntero a la instancia del cliente, evitando su paso constante como parámetro.

En la figura abajo, se muestra como quedan las relaciones luego de que se inicialicen y el cliente quede listo para comenzar a comunicarse con el servidor.

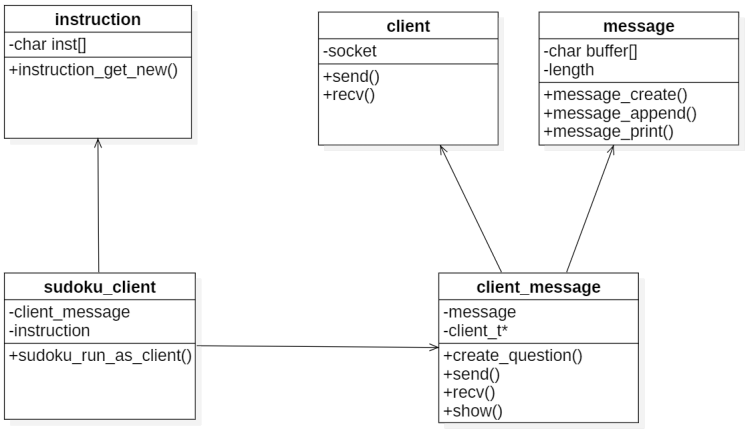

Figure 3: TDAs relacionadas al cliente

Abstracción del esquema Cliente-Servidor

En esta sección se explica como se abstrae la comunicación del cliente-servidor con el protocolo de mensajes utilizados en el programa.

La comunicación entre servidor y cliente se realiza mediante un TDA **message**, que permite guardar un mensaje como *string* y su longitud. Como el protocolo de envío y recepción es distinto para el servidor y para el cliente, se agrego una capa superior de abstracción.

Por un lado tenemos el TDA **instruction** y el TDA **client_message**. El primero, como ya mencionado previamente, captura la instruccion dada por el usuario, asegurando que esta sea valida y pueda ser procesada mas adelante sin problemas. Esa instruccion capturada es enviada al segundo TDA mencionado, que inmediatamente la delega para ser procesada por un modulo llamado **question_client**, que conoce el formato en que una instruccion es enviada al servidor, por lo tanto se encarga de crear el mensaje correctamente. Luego de creado el mensaje es enviado por el cliente.

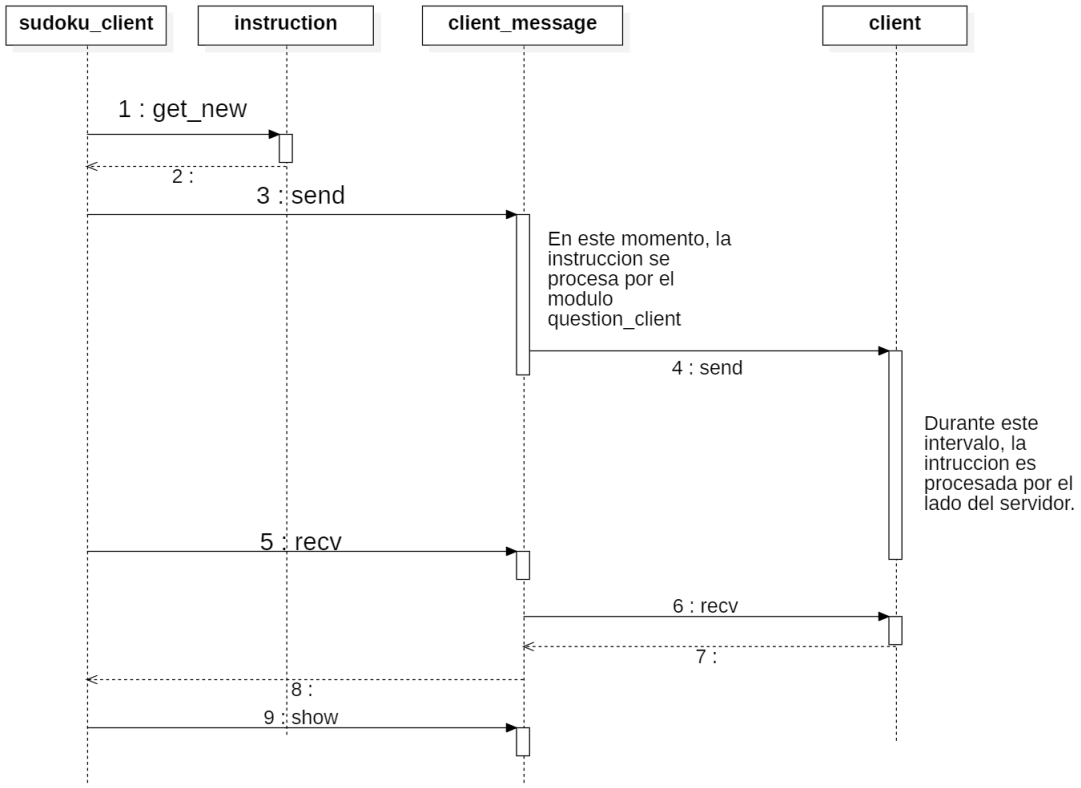

Figure 4: Secuencia del envío/recepción de mensajes por el lado del cliente

Por otro lado tenemos el TDA **sudoku** y el TDA **server_message**. El mensaje enviado previamente por el cliente es recibido por el segundo TDA, y al igual que sucedía con el cliente, este es procesado por un modulo llamado **answer_server.h**, que crea una respuesta apropiada a la instruccion del cliente.

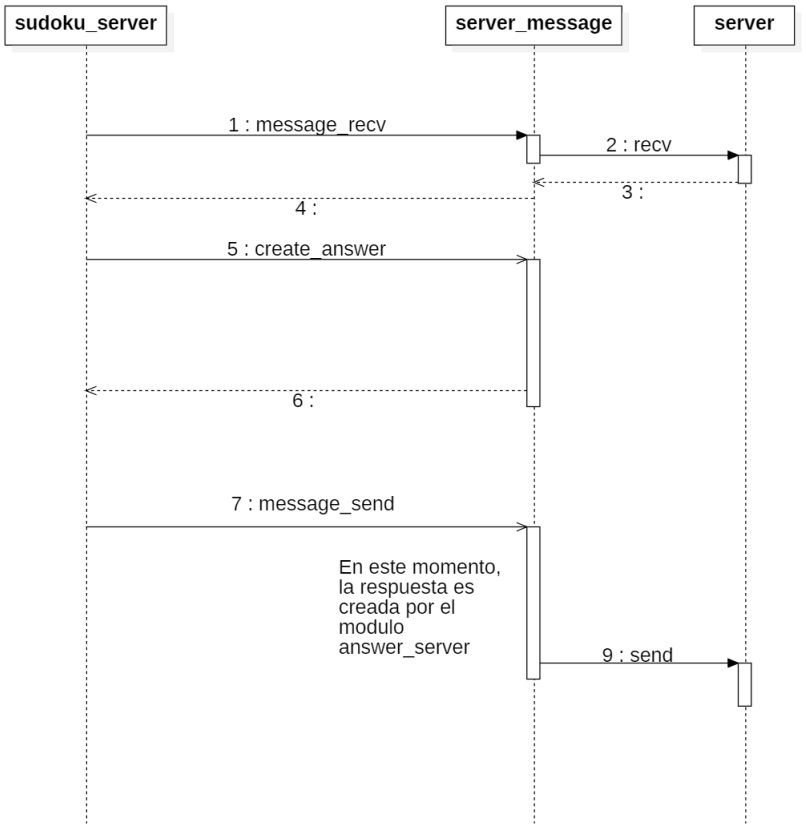

Figure 5: Secuencia del envío/recepción de mensajes por el lado del servidor

Este diseño permite realizar abstracciones en tres niveles. La abstracción mas alta esta presente en el **sudoku_client** y en el **sudoku_server**, que desde un alto nivel, reciben e envían instrucciones y respuestas respectivamente, sin preocuparse por su formato. Un poco mas abajo, estan los TDA **client_message** y **server_message** que son los que mantienen contacto con el cliente y el servidor respectivamente, envían y reciben los mensajes en el orden deseado (en el caso del server por ejemplo, envia primero la longitud y luego el resto del mensaje, y el caso del cliente, lee primero la longitud y luego corrobora recibir el mensaje con el tamaño esperado). Finalmente, el nivel mas bajo de la abstraccion se encuentra en los módulos **answer_server** y **question_client** que preparan los mensajes con el formato requerido por el protocolo.

Comunicacion Cliente-Servidor

El esquema cliente servidor esta conformado por cuatro módulos principales:

- message
- socket
- client
- server

El TDA socket ofrece un conjunto de funciones para establecer una conexion entre un cliente y un servidor.

Los TDA client y server son un nivel mas en la abstraccion del socket. Ambos reciben un mensaje como parámetro y controlan que los mensajes sean enviados o recibidos completos.

El TDA message es la forma en que el cliente y el servidor reciben la información a enviar, además de utilizarlo para guardar la información recibida. Los mensajes contienen una cadena y una longitud, que puede ser accedida por el servidor o el cliente para corroborar que el envio y la recepción hayan sido completas.

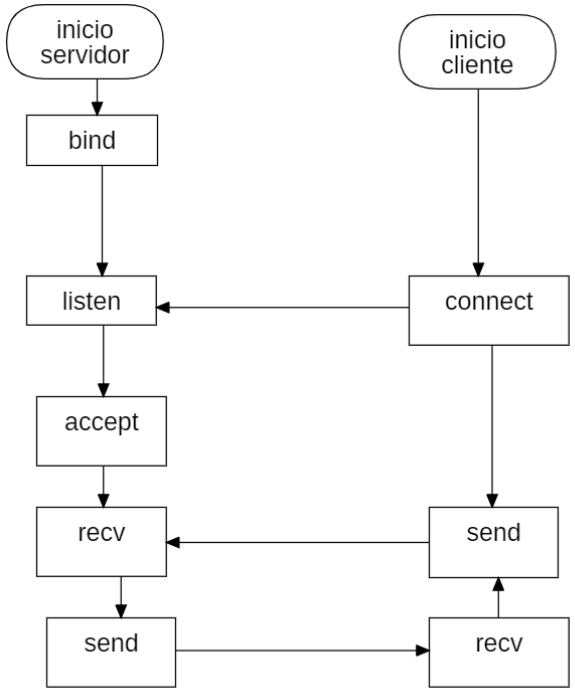

Figure 6: Conexión server-client

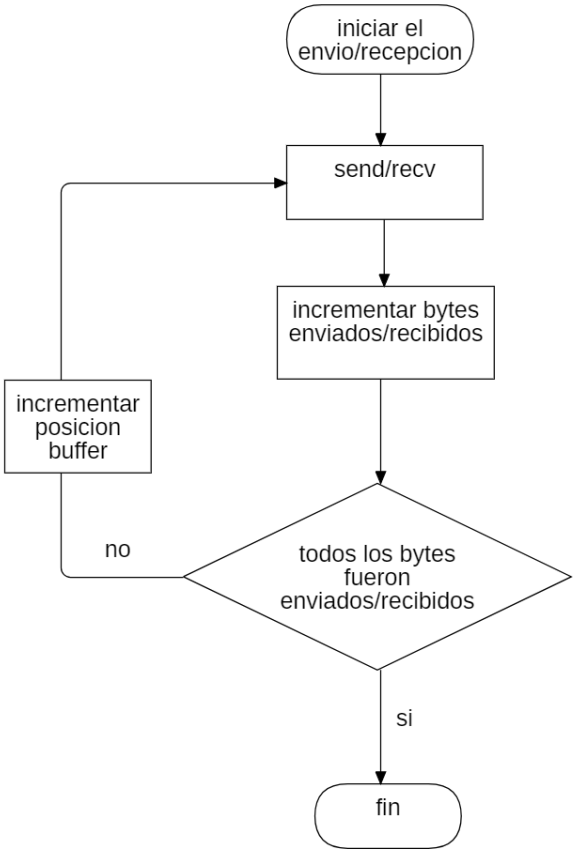

Figure 7: Control de envio/recepcion de datos

El juego

En esta sección se explica en detalle como es el diseño del Sudoku. TDA **sudoku** consiste un tablero (el TDA **board**) y este a su vez, formado por un vector de 81 casilleros (TDA **square**).

El TDA **sudoku** sirve para restar algunas responsabilidades al tablero, como obtener los números del archivo o transformar la información de filas y columnas, que viene como *char* desde el servidor.

El tablero tiene las mayores responsabilidades en lo que respecta al juego. Este no solo es el que carga el dibujo y lo actualiza, si no que también se encarga de verificar el estado del juego (correcto o inconrrecto). Para las validaciones, utiliza la ayuda de un modulo llamado **rules**, que se encarga de dado un conjunto de nueve valores (que pueden ser una fila, columna o cuadrado) verificar si cumplen con las reglas o no. Este modulo también es utilizado por el TDA **instruction** para verificar si los valores pasados de en la instruccion *put* son validos.

En el ultimo eslabón de la abstraccion se encuentran las celdas (TDA **square**). Estas contienen un numero asignado, que puede ser fijo o no. Para determinar su condición de variabilidad, contienen un atributo booleano.

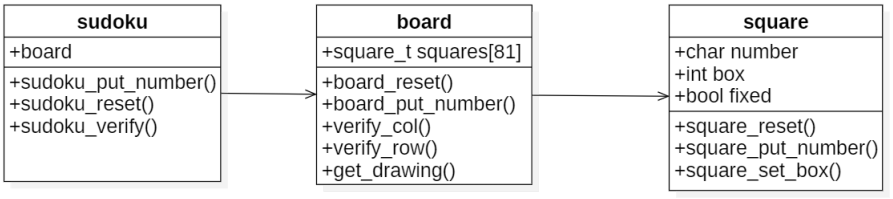

Figure 8: TDAs que conforman el juego

Módulos Auxiliares

A fin de evitar una excesiva cantidad de funciones en cada modulo, se crearon dos módulos auxiliares con algunas funciones útiles para ser usadas por las otras librearías del programa.

La primera es un modulo llamado **interfaces**, que maneja todas las salidas por pantalla, sean de las respuestas del servidor asi como los mensajes de errores. La idea de encapsular todo en este modulo permite que se puedan realizar cambios en la forma que deseamos mostrar los mensajes muy fácilmente, por ejemplo, cambiar el idioma de los errores. La interfaz también se encarga de validar los parámetros de inicio del programa y también de cargar en un buffer el dibujo del tablero. Como mencionado luego arriba, la decisión de esto ultimo es debido a que si se deseara cambiar el formato del tablero, no habría que adentrarse mucho en el código.

El otro modulo utilizado, llamado **utils** que contiene un conjunto de herramientas de uso general, como por ejemplo, funciones que permiten transformar un caractér en un índice o cargar la notación hexadecimal de un numero a un vector.

Cambios

1. Eliminación de las funciones *uint_to_array* y *array_to_uint* del modulo m**utils.h**
2. Cambio en la asignación de la instancia de server y sudoku al $sudoku_serverevitandolacopia$.
3. Construcción y destrucción del parser en el metodo constructor del sudoku (RAII).
4. Eliminación de los errores de memoria generados al metodo *freeaddrinfo*.
5. Cambio en la ubicación del ciclo que controla el envio del total del mensaje, pasando del server/client al socket.
6. Verificación de la correcta creación del socket.
7. Simplificación del parser. Una nueva implementación mas corta y menos propensas a errores.
8. Eliminación del TDA **message**. Los mensajes se envían directamente desde el TDA **server_message** y **client_message**.
9. Eliminación de la validacion innecesaria del formato del puerto.
10. Se definieron constantes sobre la posición de la fila, columna y numero en una instruccion.