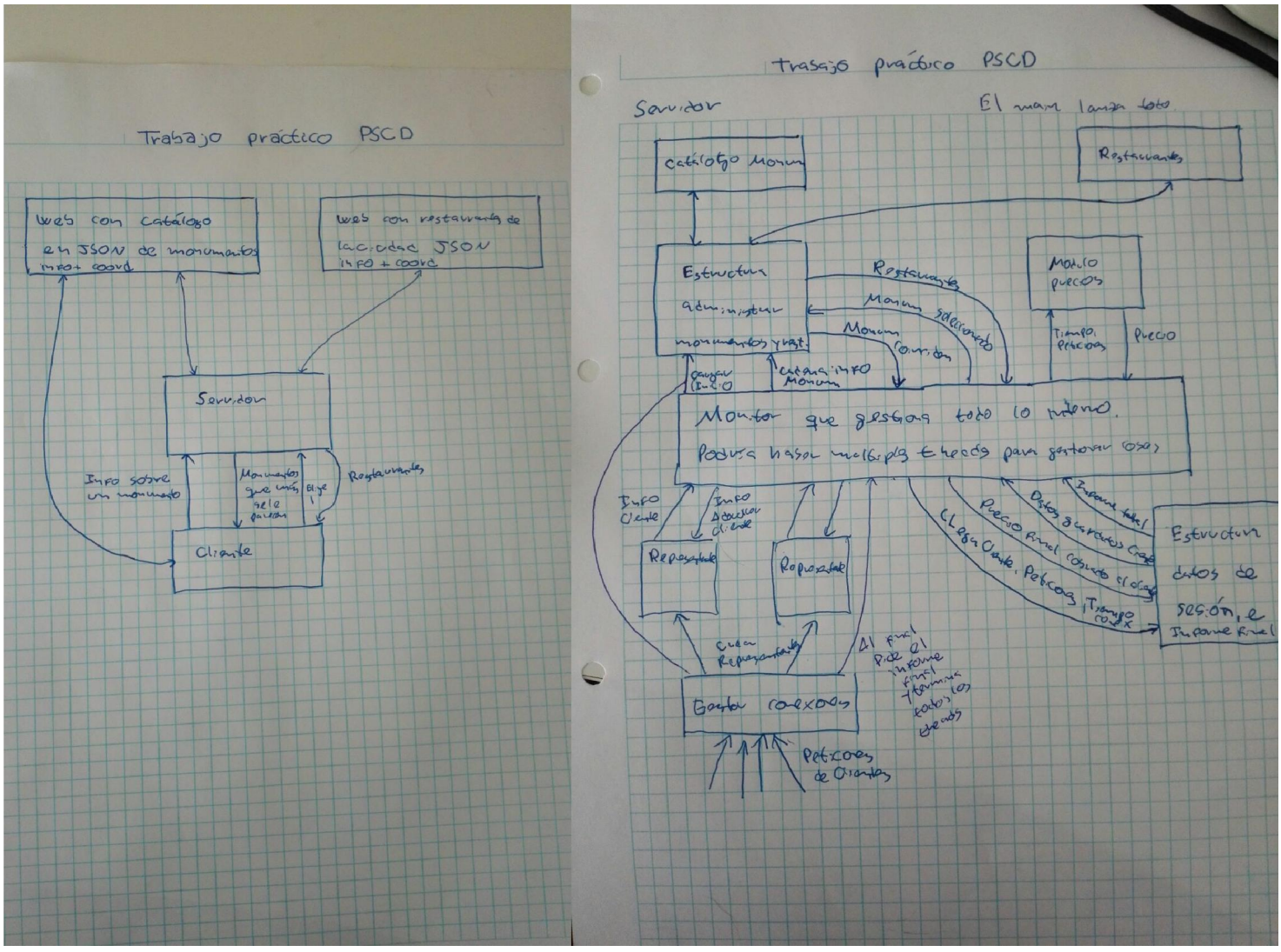
Decisiones de diseño adoptadas para el Trabajo Práctico

Jorge Aznar, Abel Chils y Daniel Fraile

A modo de comienzo, aquí mostramos una imagen descriptiva del funcionamiento del modelo cliente/servidor que hemos tomado para la realización de este trabajo, y del cual describiremos posteriormente todos los módulos involucrados.



La primera imagen describe la interacción entre el cliente/servidor, donde este primero envía una serie de términos relacionados con su monumento de interés, el servidor determina qué monumento es el más adecuado a su búsqueda , es decir, que contenga alguno de los términos recibidos, y le

devuelve una gama de opciones entre las cuales el cliente elige el que prefiera, a partir de lo cual el servidor le indica los restaurantes más cercanos a dicho monumento.

Observamos que a su vez el servidor posee dentro de sí el catálogo de monumentos/restaurantes ya cargado.

La segunda imagen describe el funcionamiento interno del propio servidor, es decir, como sus diferentes módulos interactúan entre sí para llevar a cabo un servicio correcto.

Al llegar una nueva conexión se crea un nuevo representante para dicho cliente, que será el que se encargará de procesar la información de dicho cliente a los demás módulos, generando así el precio, la lista de monumentos preferidos, sus correspondientes restaurantes, y dándolo de alta en el registro de sesión, donde se guardará toda la información de nuestros clientes para que al finalizar la sesión se muestre a modo de resumen.

Aclaraciones: En los módulos que tienen que leer JSON usamos una librería externa de código abierto para C++, pues así evitamos el proceso de desarrollar una siendo que la obtenida es perfectamente funcional.

Asimismo, por decisiones de implementación hemos decidido que los términos de búsqueda pasados por el cliente sean independientes entre sí, es decir, que podamos acceder desde ellos a diferentes monumentos, lo cual aumenta las posibilidades de obtener un monumento que se acerque más a la descripción del cliente en base a los términos, devolviendo como máximo 5 monumentos distintos entre los cuales el cliente podrá elegir el más adecuado a su gusto. Este modelo de búsqueda que hemos decidido usar podría modificarse de manera trivial si se desease que sólo nos devolviese un monumento que tuviese incluidos todos los términos modificando el árbol de sufijos implementado.

Funcionamiento detallado del cliente

El funcionamiento del módulo de clientes es el siguiente, hemos desarrollado un cliente de funcionamiento manual, es decir, en el que el propio programador introduce los datos manualmente y selecciona que restaurante quiere que le atienda y un cliente que genera todos estos parámetros automáticamente conforme discurre la ejecución del programa cliente/servidor.

El cliente, automático o manual, realiza sucesivas peticiones de servicio, enviando en un primer momento un identificador, a continuación los parámetros de búsqueda y después tiene la opción de solicitar el restaurante

más cercano a uno de los monumentos recibidos o finalizar el servicio. Al recibir los monumentos abre las URL y al recibir las coordenadas del restaurante abre una pestaña con un mapa mostrando el lugar.

Los mensajes recibidos del servidor son tratados de forma distinta según corresponda. En el caso de error al enviar o recibir los datos termina la ejecución. En el caso de solicitar la finalización muestra por pantalla el precio del servicio.

No hemos necesitado manejar ningún aspecto de sincronización en el desarrollo de la implementación del cliente.

Funcionamiento detallado de los módulos del servidor

Método de gestión de las conexiones obtenidas: Hemos implementado un gestor de conexiones que dispone

Respecto a la manera de gestionar la **base de datos que contiene nuestros monumentos** hemos decidido implementar un árbol de sufijos ya que es una estructura de datos que aunque en memoria tenga un gran costo y en tiempo de creación también lo tenga, las búsquedas son de orden lineal en la longitud del substring insertado para buscar, lo cual hace que estas sean muy rápidas, que es lo que se espera de un servidor, además de esto las búsquedas se pueden realizar desde múltiples threads simultáneamente sin utilizar medios de sincronización. En este árbol de sufijos hemos tomado algunas decisiones para mejorar el rendimiento como son: almacenar en cada nodo los valores de las claves que empiecen por ese inicio de cadena. Entonces en cada nodo se guardan tantas coincidencias posibles como monumentos que posean el substring introducido, debido a que aunque hayan mas almacenadas no se devolverán más de 5, solamente se ha de devolver el link y las coordenadas de cada clave, que son la información con la que se trabaja en los módulos Monumentos y Restaurantes.

La política de proximidad en las búsquedas consiste en que el resultado devuelto ha de poseer todos los parámetros insertados para buscar, como pasaría en un buscador

Para conseguirlo primero busco en el árbol de sufijos los términos independientemente y más tarde de estos términos busco los resultados que coinciden en todos hasta llegar a un máximo de 5.

El **funcionamiento estructura de datos de la sesión** es el siguiente, esta estructura de datos tiene como fin almacenar, a modo de resumen, todas las peticiones realizadas por cada cliente que solicite conexión a nuestro servidor. Sin importar cuantas peticiones de información solicite estas se guardarán como el valor de un par (clave,valor) en el que la clave es el entero ID de nuestro cliente y el valor es la lista de peticiones realizada por el mismo. Para implementar este diccionario hemos hecho uso del TAD "Árbol tipo AVL"; estudiado en la asignatura de Estructuras de Datos y Algoritmos. Debido al factor de equilibrio que posee todas las operaciones que realicemos sobre él se mantendrán siempre en orden de complejidad O(log n), facilitando al servidor el servicio solicitado lo más rápidamente posible. La sesión que creemos, y en la que vayamos insertando la información obtenida con cada petición de nuestros clientes, nos permitirá listar todas sus peticiones individuales o el colectivo de las de todos los clientes registrados en dicha sesión, que se cerrará con el servidor, así como tan sólo el número de peticiones en un momento dado de un cliente dado o del colectivo de clientes. Sólo nos interesa usar las operaciones de inserción, extracción y búsqueda, que a su vez hacen uso del propio iterador que el árbol tiene ya incorporado. Para facilitar la sincronización se ha añadido un mutex que se ocupa de evitar el acceso a datos en memoria dinámica, o datos que puedan variar con facilidad.

DETALLE: hemos usado Apache Solr para leer JSON directamente desde HTML, Solr se ejecuta como un servidor de búsqueda de texto completo independiente. Utiliza la biblioteca de búsqueda de Lucene Java en su núcleo para la indexación y búsqueda de texto completo y tiene APIs HTTP / XML y JSON similares a REST que la hacen utilizable de los lenguajes de programación más populares.

Respecto al **sistema de tarifas**, este se ha basado en el que realizan los taxis, es decir, se pagará una tasa fija por el hecho de realizar la conexión y luego una cantidad variable que será proporcional al número de peticiones realizadas por dicho cliente.

Funcionamiento del gestor de conexiones: El servidor crea el socket y mientras no se le envíe el mensaje de finalización procede a crear un thread por cada

cliente. Este thread se crea con la función representante que se invoca con los parámetros id del cliente y el socket con el que trabajamos.
Esta función procede, en el caso de que no se haya solicitado la terminación del servidor, a hacer sucesivas recepciones y envíos de mensaje, hasta que se solicite la finalización, tratando el mensaje de forma distinta según corresponda.
Para la traza de ejecución y la modificación de variables globales se han usado respectivos mutex, para evitar que se solapen los mensajes por pantalla y evitar errores de concurrencia respectivamente.
En caso de error en la recepción o envío del mensaje se procede a cortar la comunicación con el cliente, enviando de ser posible el precio del servicio.
En caso de solicitud de finalización el servidor muestra la información de la sesión, envía el precio del servicio al cliente y procede a terminar la comunicación.
Acerca de la finalización, esta se realiza cuando se escribe fin por el terminal. Para lograr que acaben los procesos representantes actuales, no entren mas y además finalice el programa se hace lo siguiente: al principio del programa se lanza un thread auxiliar que se encargará de todo esto. En este thread , primero se llama a la entrada estándar, la cual bloquea el thread, y una vez que se ha escrito algo y se desbloquea el thread, se hace wait hasta que todos los procesos representantes hayan acabado; en ese momento se cierran los sockets por lo que si el proceso principal está en accept, es decir que si desde que se escribió fin no ha llegado ningún cliente mas, el accept fallará y devolverá -1 por lo que el proceso principal saldrá del bucle y terminará controladamente.

DIFICULTADES

Hemos encontrado dificultades en el uso de las tecnologías que necesitábamos para realizar la práctica.

La primera dificultad encontrada fue al utilizar ficheros json debido a que debimos de aprender la codificación de estos ficheros así como encontrar una biblioteca que nos permitiera leerlos de manera fiable y luego aprender a utilizarla.

Por otro lado también hemos encontrado dificultades al utilizar la api de Zaragoza para realizar las consultas debido a que es mucha documentación y son tecnologías desconocidas para nosotros por lo que hemos tenido que investigar sobre sorl .

La tercera dificultad que encontramos residía en que la web que generábamos con el buscador sorl no la podíamos descargar correctamente con el modulo imagen downloader , por lo que tuvimos que investigar sobre el uso de curl.

Asimismo también hemos empleado conocimientos obtenidos en otras asignaturas a modo de refuerzo en la realización del trabajo. Por ejemplo: estructuras de datos como árboles equilibrados o árboles de sufijos y listas.

También modificamos el ImageDownloader para que pueda descarar archivos tipo JSON, el nuevo nombre es JsonDownloader.