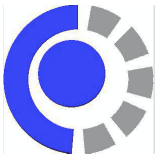

Laboratorio de Programación Distribuída
Departamento de Ingeniría de Computadoras

Pesce Fiorella FAI 213
Moreira Martin FAI 453

April 12, 2017

Contents

List of Figures

Part I
Laboratorio I

1 Introducción

El problema planteado por la cátedra y que se procede a resolver en el presente trabajo, corresponde a realizar un programa en el lenguaje Java, que respete el esquema de comunicación presentado en la Figura 1. La idea central del ejercicio, es que uno o varios Clientes solicitan a un Servidor, las predicciones del clima y del horóscopo según la fecha y el signo ingresados respectivamente. Para el mismo, se especifica que se utilicen las primitivas de programación de sockets de tipo Stream.

El punto clave se encuentra en que se puedan comprender los conceptos básicos de la comunicación en sistemas distribuidos, soportada por el modelo orientado a mensajes ofrecido por la capa de transporte, mediante la utilización de sockets. Conceptualmente, un socket es un punto final de comunicación sobre el cual una aplicación puede escribir datos que deben ser enviados sobre la red subyaente, y desde la cual la información entrante puede ser leída. Un socket forma una abstracción sobre el actual punto final de comunicación real usado por el sistema opertivo local para un específico protocolo de transporte.

Por lo tanto, se identifican los siguientes objetivos:
- Desarrollar el diagrama de comunicación entre Cliente y Servidores
- Permitir que los Servidores puedan atende varias consultas a la vez
- Mismas consultas deberían devolver los mismos resultados

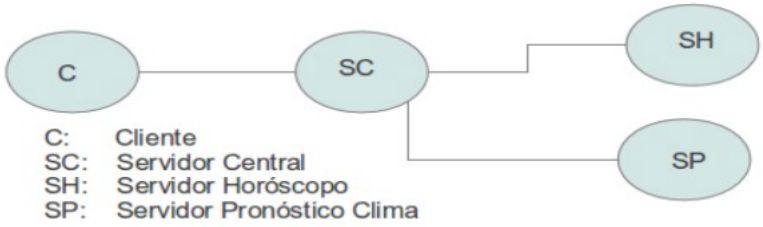

Figure 1: Esquema dado

2 Desarrollo

El ejercicio que se nos plantea nos obliga a definir varios puntos desde el principio; ¿Cuál va a ser el protocolo de comunicación entre Cliente y Servidor?, ¿Dónde se van a realizar las validaciones de los datos de entrada?, ¿En qué lugar se va a situar la caché? ¿Estará o no replicada?. Para aclararlos, presentamos el esquema general de comunicación de nuestra aplicación en la Figura

2.

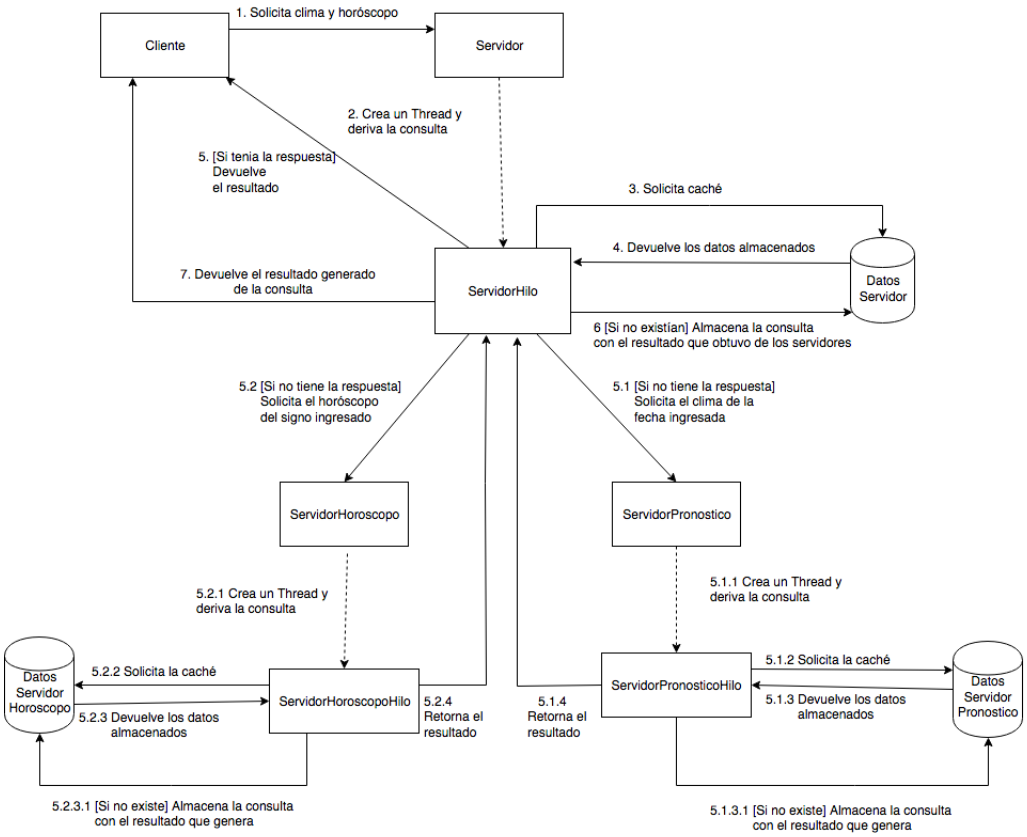

Figure 2: Esquema de comunicación

En una primera instancia, el Cliente solicita al usuario los datos de entrada. Esto debería ser, la fecha y el signo deseados separados por un guión. Por ejemplo: '25/10/2017-Tauro'. Una vez validada la entrada, el Cliente envía esta consulta al Servidor, y luego solo espera la respuesta del mismo. Al principio, solo se había validado que se ingresara un formato de fecha correcto, y un signo que existiese. En una segunda instancia, se decidió que los datos se pueden ingresar de cualquier manera siempre y cuando exista una fecha o un signo, de esta manera, se brinda la posibilidad de solicitar solo un servicio. Para ver estas posibilidades, se debe escribir el comando 'manpage'.

Mientras el Cliente espera, el Servidor tiene que conseguir la respuesta; para esto, lo que hace es crear un thread - ServidorHilo - y derivarle la consulta. De esta manera, posibilita responder varias consultas simultáneamente, porque una vez derivada la consulta al thread creado, solo se queda esperando una conexión de otro Cliente (para el cual procederá de la misma manera).

Lo primero que hace el ServidorHilo es verificar en su caché, un hash que

nosotros llamamos DatosServidor, que no tenga la información que le ha sido solicitada. Este hash le es pasado como parámetro al momento de su creación, por lo que en realidad el propietario es el Servidor central. Con esto, estamos asegurándonos de que todos los threads creados por él compartan la misma memoria, obviamente con los pertinentes mecanismos de sincronización. Si la consulta ya había sido realizada con anterioridad, y la respuesta requerida se encuentra en la caché, el thread simplemente devuelve estos datos al Cliente. En caso contrario, separa la consulta en Fecha y Signo y los envía al Servidor Pronóstico y al Servidor Horóscopo respectivamente, para que ellos traten con las mismas.

La forma de operar de estos dos servidores es la misma, más allá de que manejan información diferente. Primero, verifican en su caché - también llamada Datos Servidor - si tienen la respuesta a la consulta entrante; si la poseen, la devuelven al Servidor Hilo; si no, crean un thread al cual le derivan la consulta (en el caso de Servidor Pronóstico, creará un ServidorPronósticoHilo; lo mismo para el horóscopo), el cual es el encargado de generar la respuesta, almacenarla en la caché y luego retornarla.

Finalmente, el Servidor Hilo actualiza su respectivo Datos Servidor, y luego devuelve la respuesta conjunta al Cliente.

Con esto, se describió a grandes rasgos como es la secuencia de comunicación entre los participantes para responder a una consulta del Cliente. Ahora se va a pasar a describir cómo es que realizan dicha comunicación.

Tanto Cliente como Servidor establecen una conexión mediante sockets. El servidor crea un socket, en un puerto determinado, y se queda escuchando, esperando por una conexión. Por su parte, el Cliente crea un socket, especificando Servidor y puerto; obviamente, este puerto debe coincidir con el puerto del Servidor al cual desea conectarse. Como se ve en la Figura 3, estamos hablando del puerto 10577.

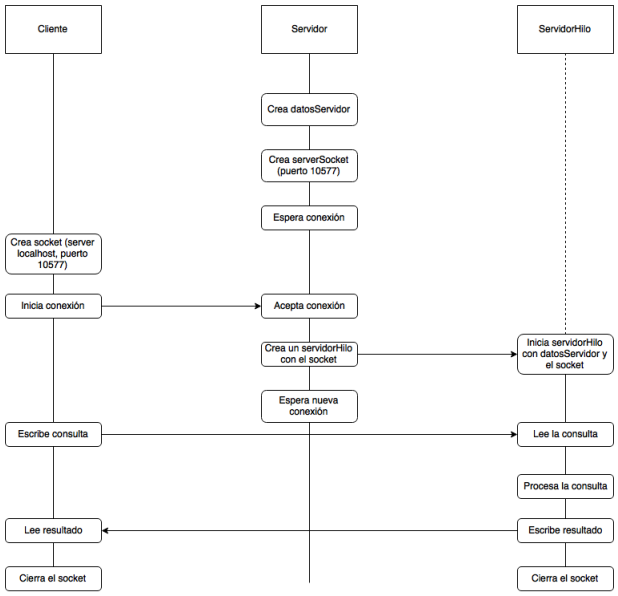

Figure 3: Comunicación Cliente-Servidor

Una vez que el Servidor acepta la conexión, ambos - Cliente y Servidor - pueden escribir y leer sobre la misma. De esta manera es que se envían consulta y respuesta respectivamente. Luego de que el Cliente obtiene su respuesta sin embargo, cierra el socket cortando el enlace. Por contraparte, el Servidor continúa escuchando por otras posibles conexiones.

Ahora bien, cuando el Servidor central llega al punto de tener que crear un thread al cual derivarle la consulta como se explico anteriormente, no solo le envía como parámetro su caché, sino también el socket con el cual se ha establecido la conexión con el Cliente; de esta manera se "desliga" de ese request y puede esperar nuevas conexiones.

En caso de que el nuevo thread deba comunicarse con Servidor Pronóstico y Servidor Horóscopo, debe seguir el mismo proceso, detallado en las Figuras 4 y 5. Creará un socket con el Servidor y el número de puerto que corresponda. Nosotros hemos utilizado el puerto 10578 para el pronóstico y 10579 para el horóscopo.

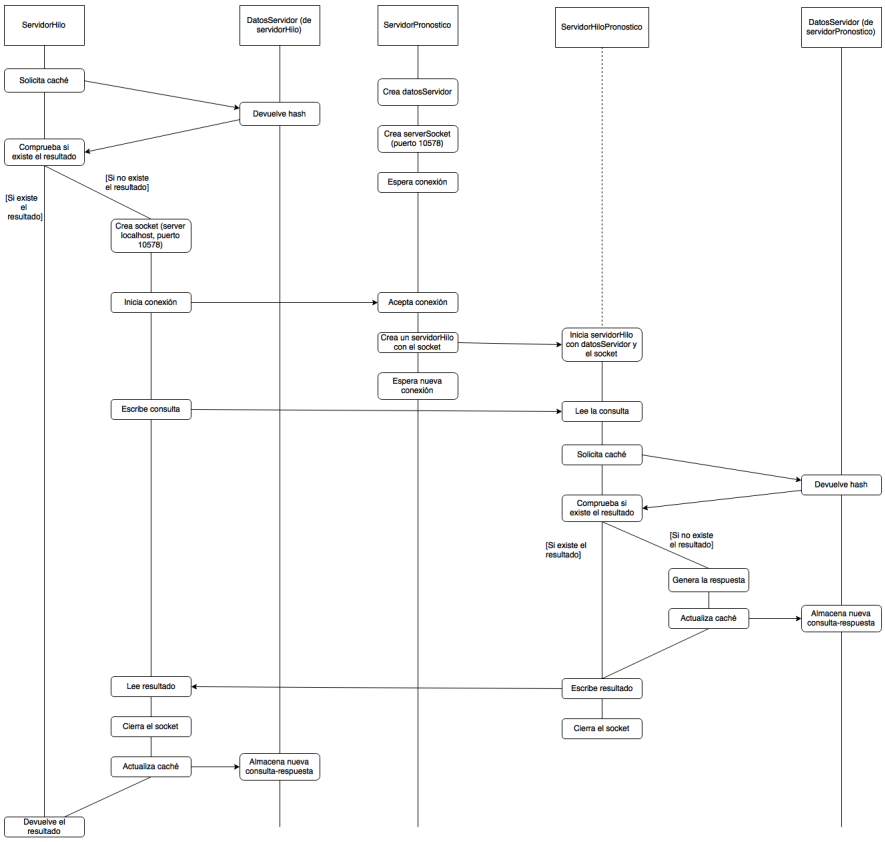

Figure 4: Comunicación Servidor-ServidorPronóstico

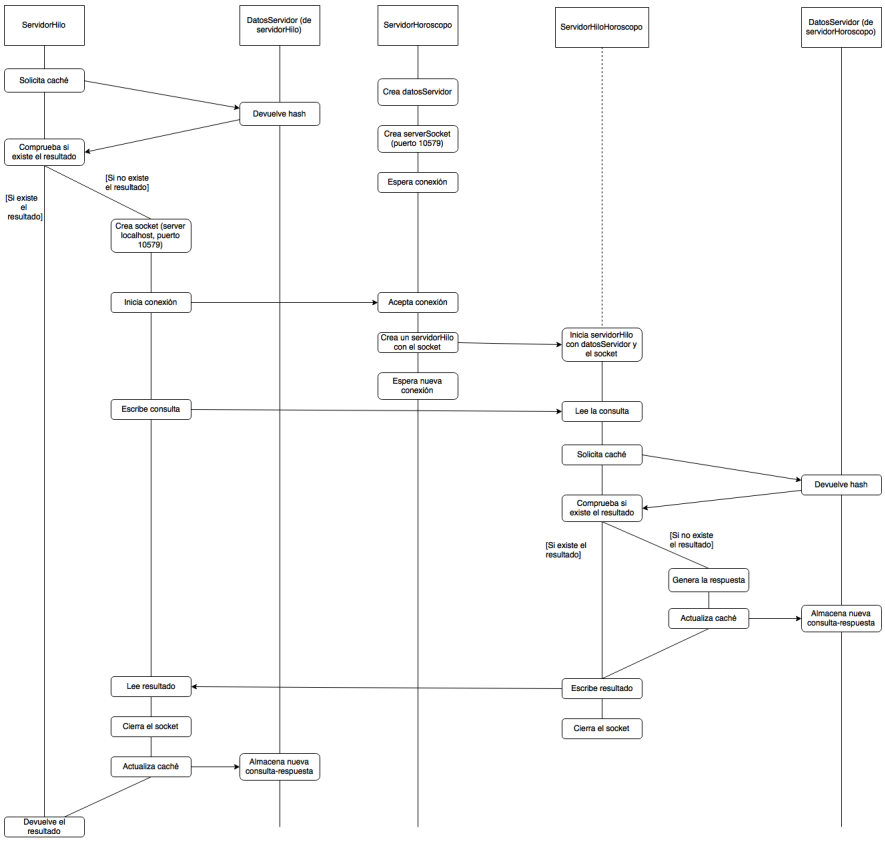

Figure 5: Comunicación Servidor-ServidorHoroscopo

Cabe aclarar que, un thread Servidor, ya sea hijo del Servidor Central, Pronóstico o de Horóscopo, solo atiende una consulta. Luego de que devuelven las respuesta que les corresponda, cierran su socket y mueren.

Otro aspecto importante de la implementación es que las validaciones se realizan mediante expresiones regulares a fin de reducir al mínimo los errores. Básicamente, en el ingreso de datos de cada servidor, se busca la entrada pertinente (fecha y/o signo) y se descarta el resto. Si no se encuentra ningún dato válido, asumimos que la entrada es errónea y devuelve un mensaje de error.

3 Conclusión

Luego de haber implementado la solución del problema planteado, se concluye que la programación distribuida soportada mediante sockets, es un tanto compleja de manejar ya que hay muchas situaciones en donde la comunicación entre dos procesos distribuidos puede romperse. Sin embargo, la implementación es

mucho más sencilla debido a que en pocas lineas de codigo pudimos experimentar la interacción entre cuatro estaciones (en este caso locales) distribuidas.

En nuestro caso, al validar todas las comunicaciones existentes, pudimos salvar muchos de los errores que existian y así generar una estabilidad considerable. Por lo tanto se puede apreciar que en un tiempo reducido pudimos cubrir los aspectos principales de la comunicación via sockets en el lenguaje Java, y experimentar problemas que existen en una aplicación real, como el uso de caché, la validación de la entrada de datos, así como también la ayuda para el uso del sistema distribuido.

4 Bibliografía

- Distributed Systems, Principles and Paradigms (2da edition) - Tanenbaum, Van Steen.

- Generador de expresiones regulares: https://regex101.com/