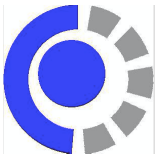

Laboratorio de Programación Distribuída
Departamento de Ingeniría de Computadoras

Pesce Fiorella FAI 213
Moreira Martin FAI 453

May 16, 2017

Contents

List of Figures

Part I
Laboratorio II

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 utilice una aplicación distribuída RMI (Remote Method Invocation).

El punto clave se encuentra en que se puedan comprender los conceptos básicos de la comunicación en sistemas distribuidos, soportada por la invocación remota de métodos pertenecienes a objetos que han sido previamente comunicados o expuestos a través de la red. Dicha invocación es soportada mediante una arquitectura de cuatro capas. La primera capa es la de aplicación y se corresponde con la implementación real de las aplicaciones cliente y servidor. En ella tienen lugar las llamadas a alto nivel para acceder y exportar objetos remotos. La capa número 2 es la Proxy. Esta capa es la que interactúa directamente con la capa de aplicación. Todas las llamadas a objetos remotos y acciones junto con sus parámetros y retorno de objetos tienen lugar en ella. La capa número 3 es la de referencia remota, y es responsable del manejo de la parte semántica de las invocaciones remotas. También es responsable de la gestión de la replicación de objetos y realización de tareas específicas de la implementación con los objetos remotos. Finalmente, la cuarta capa es la de transporte, responsable de manejar las conexiónes correspondientes y el transpaso de datos de una máquina a otra; el protocolo subyacete para RMI es JRMP (Java Remote Method Protocol).

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

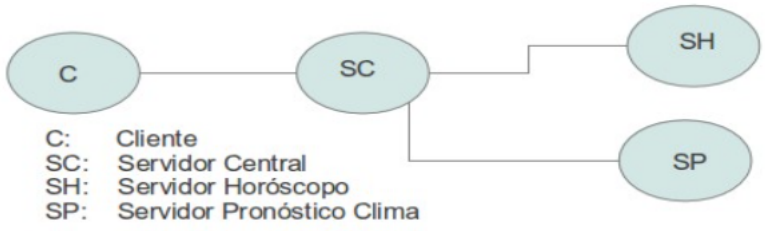

Figure 1: Esquema dado

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.

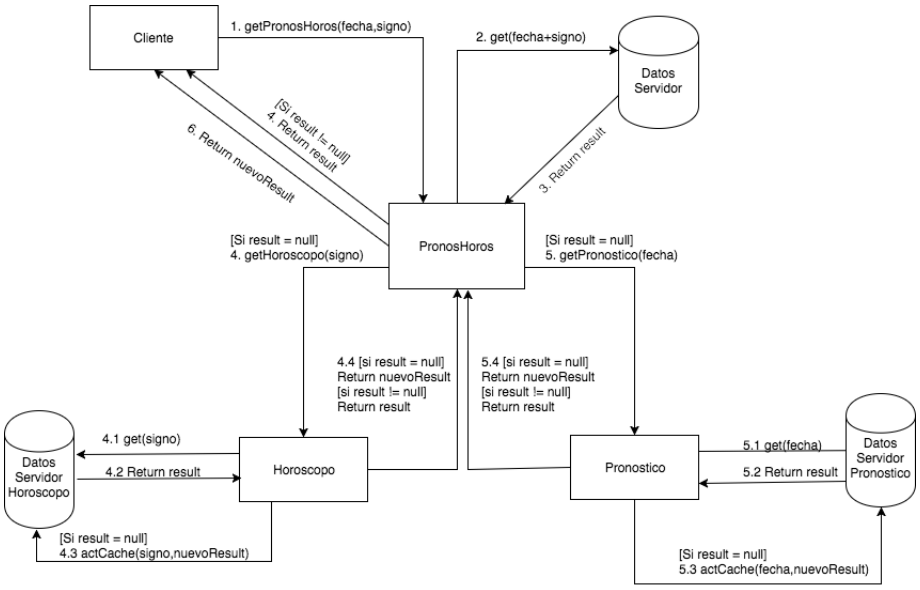

Figure 2: Esquema de comunicación

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 invoca el método getPronosHoros que pertenece a la clase PronosHoros y simplemente espera la respuesta. Para poder realizar esto, ambas clases tienen conocimiento de la interfaz, la diferencia radica en que PronosHoros es la clase que implementa los metodos que en ella figuran. Además, y para que puedan ser invocados, debe haberlos registrado en Registry, que es el que se encarga de hacer "visibles" esta clase en forma de objeto remoto para que los Clientes puedan invocar los métodos. 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 'help'.

Mientras el Cliente espera, la clase PronosHoros en la implementación de su método get tiene que conseguir la respuesta; para esto, lo que hace es

como primer paso, verificar que 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, de la cual está encargado el Servidor. Si la consulta ya había sido realizada con anterioridad, y la respuesta requerida se encuentra en la caché, el método simplemente retorna estos datos. En caso contrario, separa la consulta en Fecha y Signo y procede a invocar dos métodos, getPronóstico y getHoróscopo, que pertenecen a las clases Pronóstico y Horóscopo respectivamente; obviamente, la Fecha es enviado al primero y el Signo al segundo como parámetros.

La forma de operar de estas dos clases 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 retornan a la clase PronosHoros; si no, estan encargados de generar la respuesta, almacenarla en la caché y luego retornarla. Por supuesto, como la invocación de los métodos de estas clases también es remota, previamente ambas interfaces se deben haber registrado en el Registry; cabe aclarar que esta tarea esta a cargo de cada Servidor pertinente.

Finalmente, cuando PronosHoros recibe la respuesta al método get, actualiza su respectivo Datos Servidor, y luego devuelve la respuesta conjunta al Cliente.

Con esto se ha descrito a grandes rasgos la forma de resolver le problema planteado. Ahora se procederá a explicar con un poco más de detalle el funcionamiento de cada clase. Para esto se presentan los siguientes diagramas.

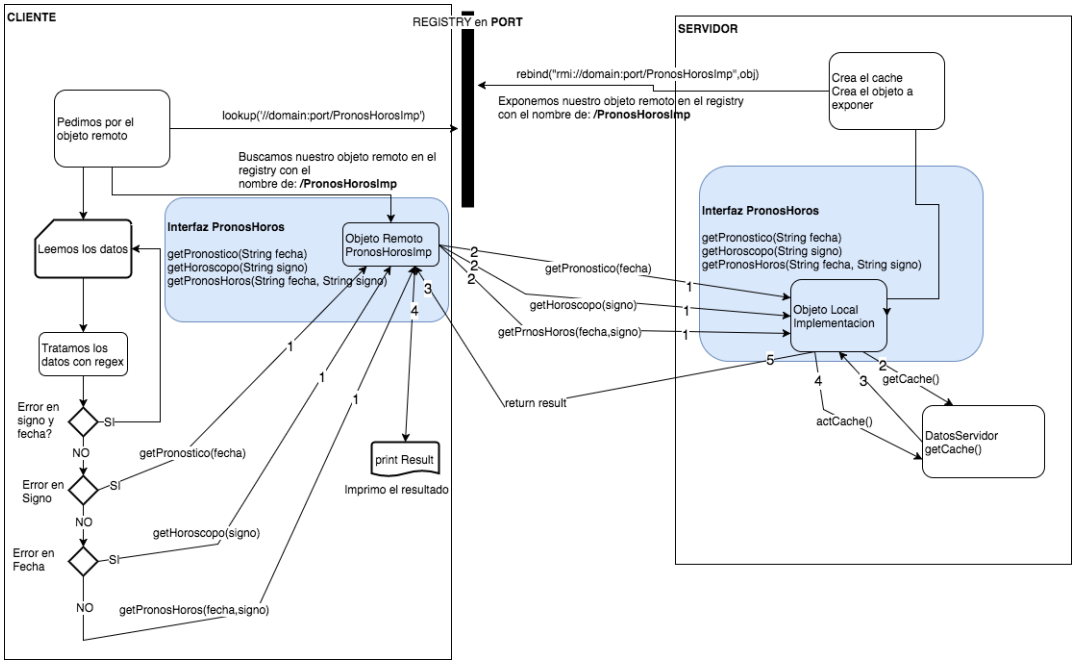

Figure 3: Comunicación Cliente-Servidor

Tanto el Cliente como el ServidorCliente tienen conocimiento de la interfaz PronosHoros, la cual define los metodos necesarios para obtener la información.

El que implementa esa interfaz, en este caso el Servidor, está obligado a exponer el objeto remoto en el registry mediante un domain especificando un host, puerto y un nombre del objeto. En nuestro caso localhost:port/PornosHorosImp.

Cuando el Cliente realiza una consulta, se verifica mediante expresiones regulares que los datos ingresados sean correctos, para luego invocar al método remoto. En el caso de que el par de datos este mal ingresado, se notifica y se piden los datos nuevamente. Sin embargo si al menos un dato se encuentra ingresado de forma correcta, el Cliente procede a hacer la consulta que corresponde a ese dato solamente.

Tomando por sentado que los datos se ingresaron correctamente, se invoca del objeto remoto el método getPronosHoros, el cual necesita fecha y signo como parámetros, luego este se ejecuta remotamente en el servidor, en el cual este realiza la verificación de la caché como se explicó anteriormente.

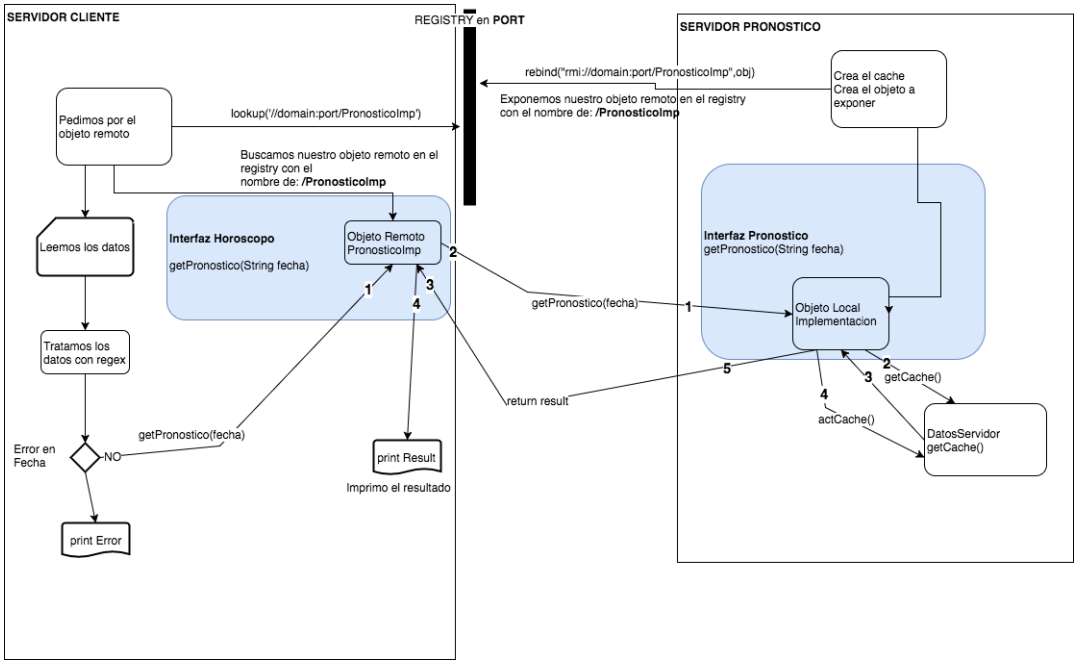

Figure 4: Comunicación Servidor-ServidorPronóstico

Las conexiones de ServidorCliente con ServidorHoroscopo y ServidorPronostico son muy similares a la anterior explicada. Los dos servidores - horoscopo y pronostico - deben exponer el objeto remoto a fin de poder obtenerlo mediante el registry para su posterior utilización.

ServidorCliente valida las entradas, para luego verificar su caché. Si necesita nueva información, hace uso de los objetos remotos realizando getPronostico en-

viando la fecha o getHoroscopo enviando el signo. En la implementación de esos métodos (muy similares) se debe buscar en caché y si no se encuentra generar una salida. Esta salida se almacena en el caché y se retorna al servidorCliente, el cual a su vez, guarda esa respuesta en caché y la retorna al cliente.

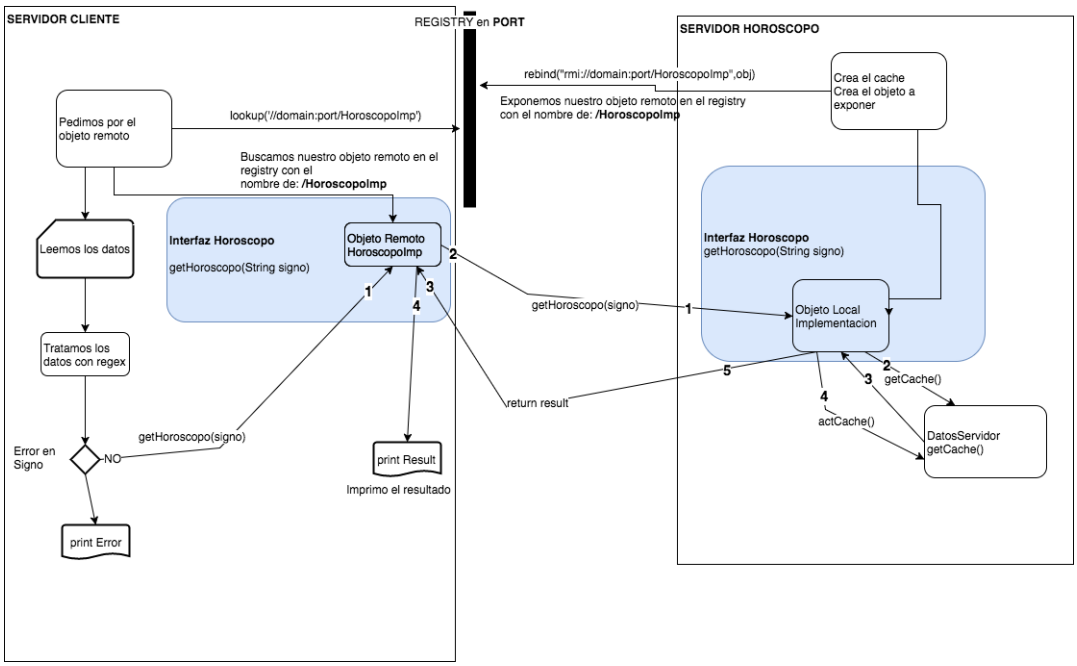

Figure 5: Comunicación Servidor-ServidorHoroscopo

Como se menciono anteriormente 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 Instrucciones de uso

A continuación se detallan los pasos a seguir para poner en funcionamiento tantos los distintos servidores como el cliente, y poder realizar las pruebas que correspondan.

1. Deben abrirse cuatro terminales, una debe estar situada en la carpeta Cliente y el resto en cada uno de los servidores (Servidor, ServidorPronóstico y ServidorHoróscopo)

2. En cada terminal, se compila el archivo que corresponda (MainCliente o

Servidor). La forma de realizar esto es escribir el comando javac NombreArchivo.

3. En las terminales de ServidorCliente, ServidorHoroscopo y ServidorPronostico se inicia el registro RMI de la siguiente manera "rmiregistry PORT &" donde PORT es el puerto elegido

4. En las tres terminales correspondientes, se ejecutan los Servidores de Pronostico y Horocopo de esta manera: 'java Servidor PORT' en donde PORT es el puerto elegido (debe coincidir con el que elegimos en el rmiregistry)

5. En el caso del ServidorCliente, debemos seguir la etapa de configuracion en donde ingresamos host y port de los servidores de Horoscopo y Pronostico.

6. Se ejecuta el Cliente siguiendo la etapa de configuracion en donde tenemos que brindarle el host donde vamos a realizar las consultas (ServidorCliente), asi como el puerto, para luego estar en condiciones de iniciar el uso del sistema

7. Se ingresa la fecha y el signo deseados para la consulta con el siguiente formato: 'DD/MM/AAAA-Signo' en la terminal del Cliente

8. Se obtiene la respuesta pertinente

4 Limitaciones, conclusión y trabajos futuros

Luego de haber implementado la solución del problema planteado, se concluye en que la utilización de RMI ha sido un gran avance en comparación a la solución anteriormente planteada con sockets. De esta manera, el programador se desliga de realizar las conexiones a bajo nivel; sólo se tiene que encargar de que las clases sean registradas en el Registry, y con esto, ya brinda la posibilidad de que sus métodos puedan ser accedidos remotamente. Así, un Cliente que se ejecute puede invocar dichos métodos sin saber particularmente si estos son locales o remotos. Por lo tanto creemos que la implementación ha sido mucho mas sencilla en comparación. Otra ventaja importante es que como programadores nos hemos visto desligados de creación de threads para le manejo de consultas en paralelo.

Más allá de que el trabajo de desarrollo fue más simplificado, hay problemas que siguen existiendo no importa cual sea la implementación, tales como el uso del caché y su posición, la validación de la entrada de datos, la ayuda para el uso del sistema. Además, cabe añadir que el Cliente tiene que conocer la Interfaz que va a invocar, la nomenclatura de los métodos, los argumentos solicitados, tipos de datos de los mismos; esto puede verse como una limitación. Otra limitación ligada al uso de RMI es que es propio del lenguaje de programación Java, por lo que no permite la interacción con aplicaciones escritas en otro lenguaje.

En nuestro caso además, logramos plantear algunas validaciones de entradas más que hacen posible que el sistema siga funcionando no importa cual sea la cadena ingresada, es decir, en caso de que el usuario no respete el formato antes nombrado.

Como trabajos futuros, es posible mencionar que puede optimizarse el uso de la caché. Si bien actualmente se manejan tres cachés, cada una en un Servidor, podría modificarse la estructura general que hemos planteado para que la caché se encuentre sólo en el Servidor PronosHoros; de ser así, esta debería permitir la consulta de resultados previamente almacenados tanto de Fecha como de Signo por separado - y no ligados como actualmente se realiza -. Debería estudiarse si la mejor manera de hacer esto es con una sola caché o con dos para distinguir los diferentes tipos de elementos en ella almacenada.

5 Bibliografía

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

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