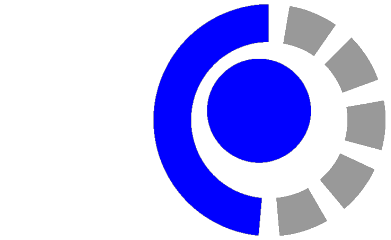
Laboratorio de Programación Distribuida

Introducción a Sockets

Bianca Antonella Torres FAI-2991
bianca.torres@est.fi.uncoma.edu.ar

Adriano Mauricio Lusso FAI-2908
adriano.lusso@est.fi.uncoma.edu.ar

Primer cuatrimestre 2025



Facultad de Informática
UNIVERSIDAD NACIONAL DEL COMAHUE

Índice

Índice de figuras

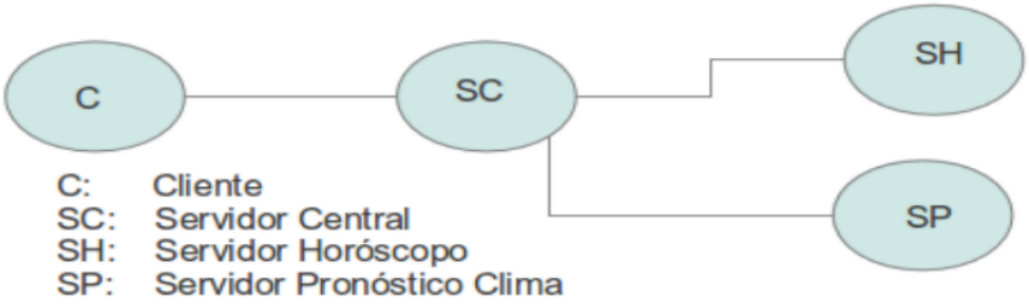


Figura 1: Estructura del programa a desarrollar.

1. Introducción

Este trabajo tiene como objetivo implementar un sistema distribuido utilizando sockets de tipo stream en el lenguaje de programación Java. El sistema está compuesto por un cliente y tres servidores. Además, se mostrarán el flujo de ejecución del programa en su conjunto y de los módulos por separado. Finalmente, se definen los trabajos futuros y las conclusiones obtenidas de este trabajo.

2. Desarrollo

Se pide desarrollar un programa en el lenguaje Java utilizando las primitivas de programación Sockets de tipo Stream. Su estructura se define en la Figura 1 y sus componentes se describen a continuación.

- Cliente: Realiza una consulta sobre el horóscopo o el pronóstico del clima, enviando dicha consulta a un servidor central.
- Servidor Central: Recibe las consultas del cliente, consulta los servidores correspondientes (Servidor de Horóscopos o Servidor de Clima) y responde al cliente con la información solicitada.
- Servidor de Horóscopos: Proporciona una predicción basada en un signo zodiacal.
- Servidor de Pronóstico del Clima: Devuelve un pronóstico del clima aleatorio para una fecha solicitada.

2.1. Estructura de la implementación

La implementación del sistema fue implementada en siete clases de Java. Las primera cuatro clases corresponden a los componentes de la estructura teórica: cliente, servidor central, servidor de horóscopos y servidor de pronóstico de clima. Además, cada servidor tiene su propia clase gestora de clientes, que se define internamente a la clase del respectivo servidor. Las clases gestoras son las que permiten la gestión concurrente de clientes, ya que por cada solicitud de un cliente al servidor, se inicializa un gestor que se encarga de atender la solicitud. En las Figuras 2 a 5 se muestran los flujos de ejecución de los componentes descritos.

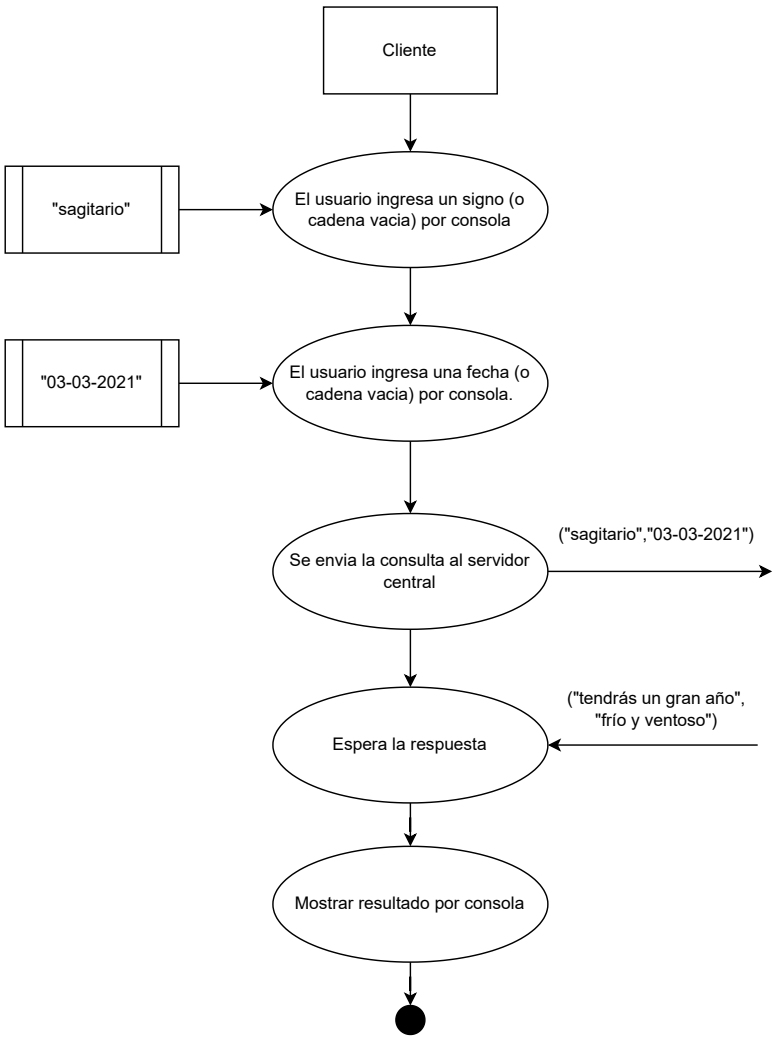


Figura 2: Flujo de ejecución del cliente.

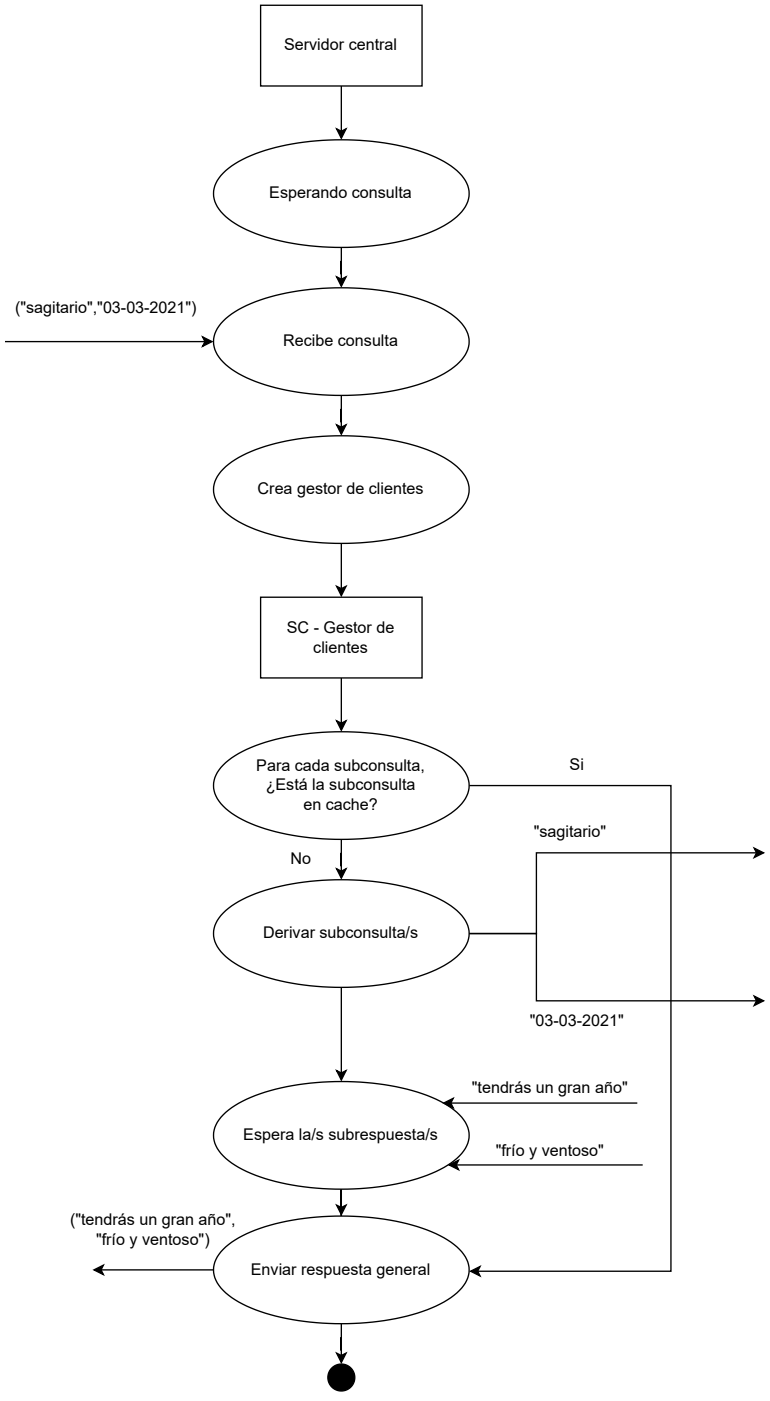


Figura 3: Flujo de ejecución del servidor central.

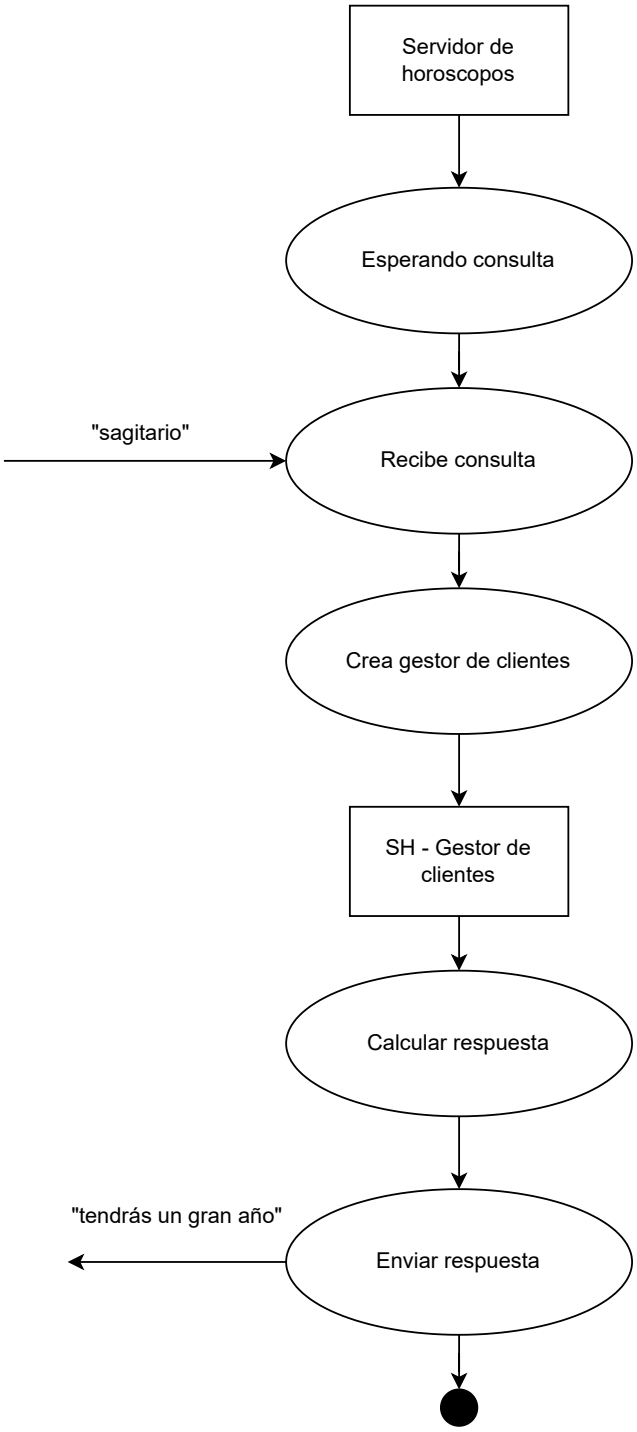


Figura 4: Flujo de ejecución del servidor de horóscopos.

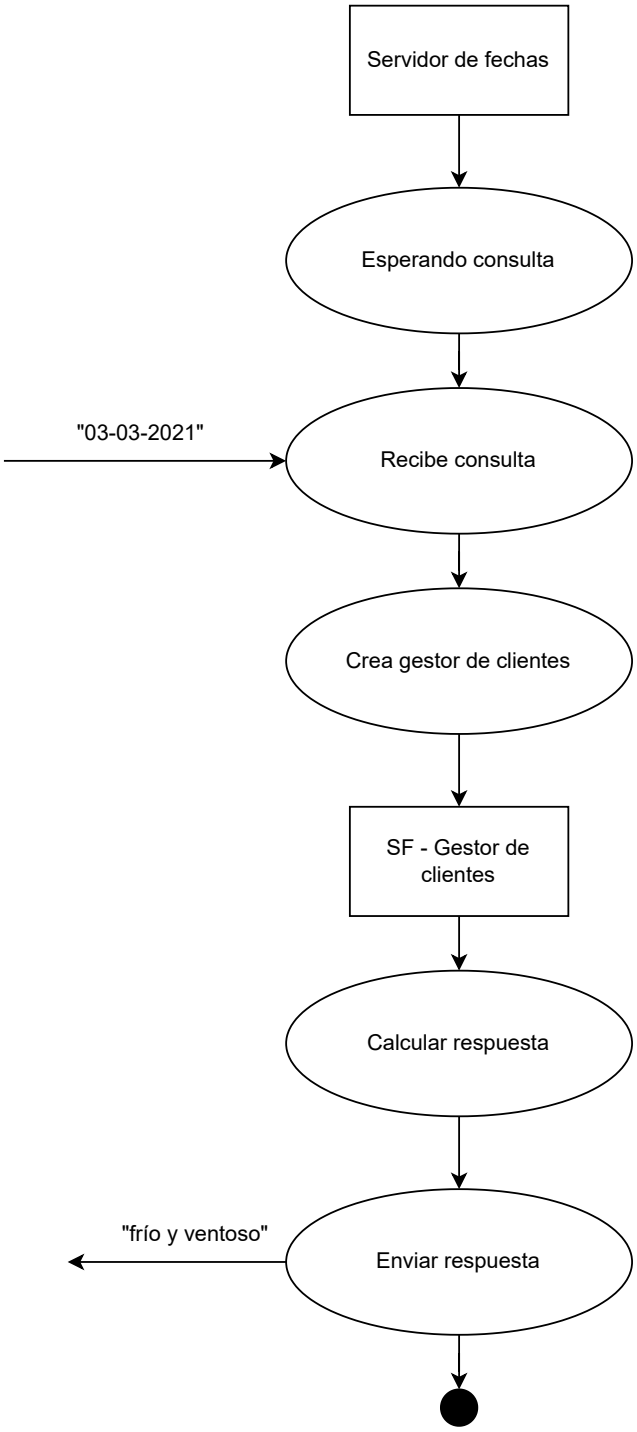


Figura 5: Flujo de ejecución del servidor de fechas.

2.2. Detalles de la implementación

El sistema multi-hilo fue implementado haciendo uso de clases tipo Thread internas a las clases servidor. Cuando el servidor detectaba una nueva petición, creaba una nueva instancia de la clase hilo, encargada de gestionar esa petición. En los servidores, para simular tiempo de cómputo y facilitar la depuración, se agregaron tiempos de espera aleatorios.

La cache fue implementada en el servidor central como una estructura de datos HashMap. Cada vez que una nueva consulta llegaba, se buscaba en la caché si ya había sido respondida. En caso negativo, se computaba su respuesta al comunicarse con el respectivo servidor secundario. Para evitar inconsistencias en los datos, la caché estaba sincronizada, de forma que un hilo a la vez pudiera acceder a la misma.

En el servidor de fechas, se predefinieron 9 pronósticos estáticos, con el fin de simplificar el modelo. Al momento de responder una consulta, se selecciona uno en función de la fecha ingresada. El servidor de horóscopos tiene una predicción predefinida para cada signo.

3. Instrucciones de uso

1. Descargar la carpeta con la implementación, anexada en la entrega junto a este informe.
2. Ejecutar los archivos java `sp.java`, `sh.java` y `sc.java` en el orden mencionado. Una forma, usando el editor de texto Visual Studio Code, es presionar el botón `Run` sobre el encabezado del método principal de cada clase.
3. En una terminal, ejecutar el comando `java cliente localhost 6789`.
4. Ingresar los datos que pide el cliente. Se utiliza `ENTER` como botón de confirmación.
5. Esperar unos segundos para obtener la respuesta.

4. Trabajos futuros y conclusiones

En este trabajo se implementó un sistema distribuido en Java utilizando sockets de tipo stream, compuesto por un cliente y tres servidores. A lo largo del desarrollo, se abordaron conceptos clave en la programación distribuida, como la comunicación entre procesos, la concurrencia y el manejo de memoria cache.

La implementación de un servidor central permitió coordinar las peticiones del cliente y distribuirlas a los servidores secundarios de manera eficiente. Además, la inclusión de una caché sincronizada en el servidor central mejoró el rendimiento del sistema al reducir la necesidad de consultas repetitivas a los servidores secundarios.

Otro aspecto destacado fue la gestión de múltiples conexiones simultáneas mediante hilos en cada servidor. Esta estrategia garantizó que el sistema pudiera manejar varias solicitudes concurrentemente sin afectar la respuesta general del servicio.

A partir de los resultados obtenidos, se identifican diversas mejoras y extensiones posibles para el sistema desarrollado. Se podría crear un segundo nivel de cache a nivel de cliente, de forma que un cliente particular pueda evitar incluso consultar al servidor central. Además, se podría

investigar por otras estructuras de datos para implementar la cache, de forma que la escalabilidad de la misma sea mejor.

Estas mejoras permitirían optimizar el desempeño y la usabilidad del sistema, ampliando sus capacidades y posibles aplicaciones en entornos reales.