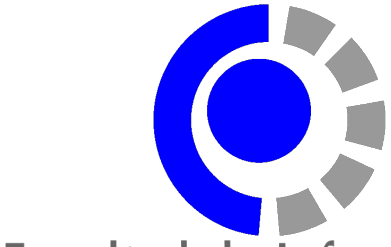
Laboratorio de Programación Distribuida

Introducción a RMI

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

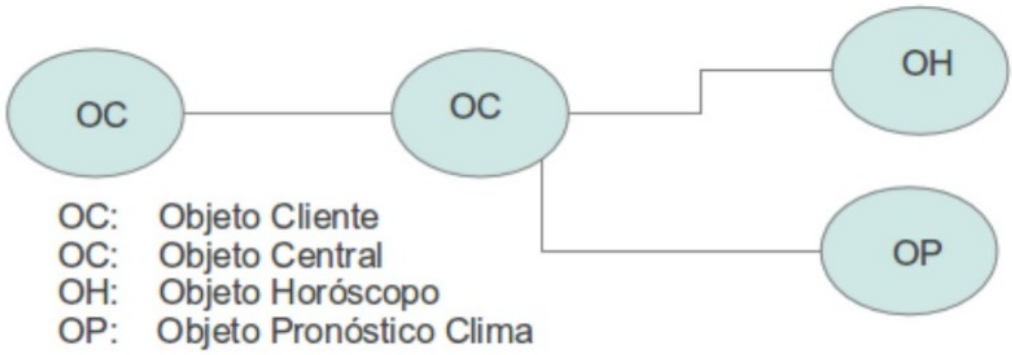


Figura 1: Estructura del programa a desarrollar.

1. Introducción

El objetivo de este trabajo es implementar un sistema distribuido compuesto por un cliente y tres servidores, utilizando Java como lenguaje de programación. La comunicación remota entre los distintos componentes se realizó mediante la tecnología RMI (Remote Method Invocation), permitiendo invocaciones de métodos a través de la red de forma transparente. En este informe se presenta el flujo general de ejecución, la estructura de los módulos que conforman el sistema, los detalles más relevantes de su implementación, las instrucciones necesarias para su ejecución y las conclusiones obtenidas a partir del desarrollo.

2. Desarrollo

Se pide desarrollar un programa en lenguaje Java utilizando invocación de métodos remota - RMI. Su estructura se define en la Figura 1 y sus componentes se describen a continuación.

- **Cliente:** envía consultas remotas al servidor central, solicitando predicciones de horóscopo o pronósticos del clima para una fecha dada.
- **Servidor Central:** recibe las solicitudes remotas del cliente, las deriva a los servidores secundarios correspondientes mediante llamadas remotas, y responde al cliente con la información combinada.
- **Servidor de Horóscopos:** provee una predicción basada en el signo zodiacal, expuesta como un servicio remoto mediante RMI.
- **Servidor de Clima:** devuelve un pronóstico aleatorio para la fecha solicitada, también implementado como un servicio remoto accesible vía RMI.

2.1. Estructura de la implementación

El sistema está compuesto por siete clases de Java principales: las cuatro clases correspondientes a los componentes teóricos —cliente, servidor central, servidor de horóscopos y servidor de clima— y las tres interfaces remotas que definen los contratos de comunicación entre ellos.

Cada servidor implementa su interfaz remota y se registra en el RMI Registry para ser localizado tanto por el servidor central como por el cliente. La configuración de los servidores secundarios (direcciones, puertos y nombres de servicios) se carga dinámicamente desde un archivo `config.properties`.

Este archivo contiene la dirección IP (o hostname), el puerto y el nombre del servicio remoto. En nuestro caso específico, los hosts están configurados como `"localhost"` para facilitar la ejecución y pruebas en una misma máquina. En un entorno real, estos valores podrían apuntar a direcciones IP o nombres de dominio de máquinas remotas. Si el archivo de configuración no se encuentra, el sistema utiliza valores por defecto para garantizar la continuidad del servicio sin necesidad de recompilar.

2.2. Detalles de la implementación

Para la gestión concurrente de las solicitudes, el sistema aprovecha la capacidad intrínseca de RMI para manejar múltiples llamadas remotas simultáneamente. RMI crea internamente un hilo por cada invocación remota, por lo que no fue necesario implementar manejo explícito de hilos ni clases internas gestoras para atender las solicitudes concurrentes.

El servidor central implementa una caché local utilizando `HashMap` para almacenar respuestas previas de los servidores de horóscopos y clima. Cada vez que llega una nueva consulta, se verifica si la respuesta está en la caché. Si no está, se realiza la llamada remota al servidor correspondiente, y la respuesta se almacena para futuras consultas. Para asegurar la consistencia y evitar problemas en accesos concurrentes, el método que accede a la caché está sincronizado.

El servidor de horóscopos posee una predicción fija para cada signo zodiacal, mientras que el servidor de clima genera pronósticos aleatorios o selecciona uno de un conjunto predefinido, simplificando así la lógica para el pronóstico del clima.

Para facilitar la depuración y simular tiempos de procesamiento, en algunos servidores se añadieron retardos aleatorios antes de responder, aunque esto no afecta el funcionamiento general del sistema.

2.3. Documentación del Sistema

A continuación se presentan distintos diagramas que permiten describir y comprender el funcionamiento interno del sistema. Se incluyen diagramas de secuencia, que muestran la interacción entre los distintos componentes a lo largo del tiempo; un diagrama de flujo, que representa el proceso lógico y la toma de decisiones involucradas; y una tabla con la interfaz pública de las clases principales, donde se detallan los métodos expuestos por cada una. Estos elementos ofrecen una visión integral tanto del comportamiento dinámico como de la estructura del sistema.

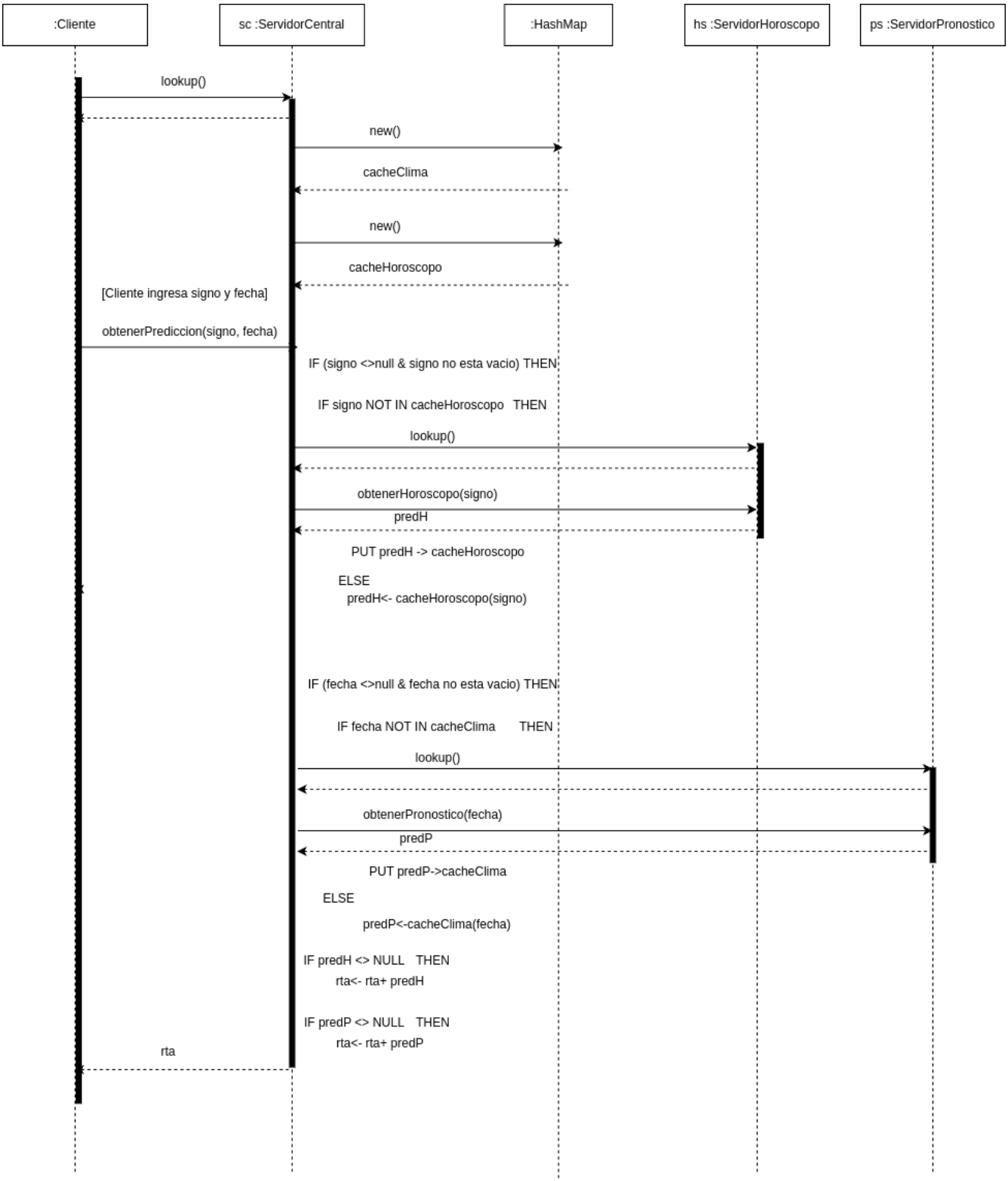


Figura 2: Diagrama de secuencia del proceso de consulta para obtener una predicción

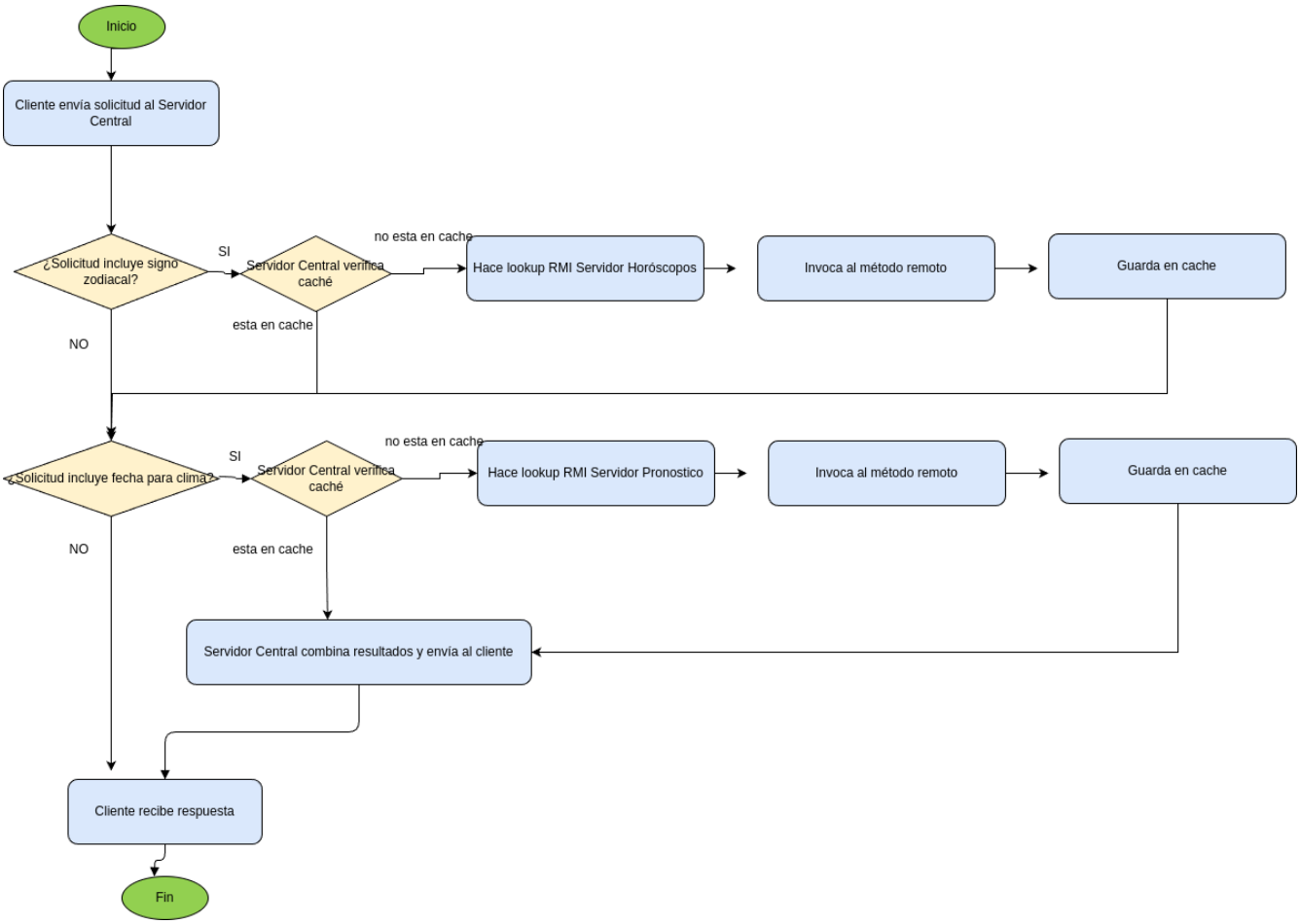


Figura 3: Diagrama de flujo del proceso de consulta en el Servidor Central RMI

ServidorCentral
+ obtenerPrediccion(signo: String, fecha: String): String

ServidorHoroscopo
+ obtenerHoroscopo(signo: String): String

ServidorPronostico
+ obtenerPronostico(fecha: String): String

Cuadro 1: Interfaces de las clases del sistema

2.4. Limitaciones y futuros trabajos

En sistemas distribuidos como el desarrollado en este trabajo, la seguridad es un aspecto fundamental debido a la exposición de los componentes a la red. En implementaciones clásicas de

Java RMI, se utilizaba un `SecurityManager` junto a políticas de seguridad para restringir qué acciones podía realizar el código descargado o ejecutado remotamente. Sin embargo, a partir de Java 17, la clase `SecurityManager` fue deprecada y su uso será removido en futuras versiones de Java, dado que se considera un enfoque obsoleto y poco efectivo frente a amenazas modernas.

Esta situación plantea la necesidad de adoptar nuevas estrategias de protección en aplicaciones distribuidas. Algunas posibles líneas de acción que podrían aplicarse como mejora al sistema desarrollado son:

- Comunicación segura mediante SSL/TLS: Configurar los canales de comunicación RMI para que utilicen estos protocolos de cifrado garantiza la confidencialidad e integridad de los datos transmitidos entre clientes y servidores. Esta práctica protege contra ataques de intermediarios y permite autenticar los extremos mediante certificados digitales, evitando accesos no autorizados y asegurando que la información intercambiada no sea interceptada ni alterada.
- Uso de criptografía: Para proteger la información que se transmite entre cliente y servidores en un sistema distribuido, es fundamental aplicar técnicas criptográficas. Estas permiten cifrar los datos para que solo los destinatarios autorizados puedan leerlos, y además asegurar que la información no haya sido alterada durante el tránsito.En Java, existen varias APIs y bibliotecas estándar que facilitan la implementación de criptografía, como: Java Cryptography Architecture (JCA) y Java Cryptography Extension (JCE), que proveen herramientas para cifrado simétrico (AES), asimétrico (RSA), generación y verificación de firmas digitales, y gestión de certificados.

Estas propuestas representan trabajos futuros que potenciarían la escalabilidad, eficiencia y seguridad del sistema, adecuándolo mejor a entornos reales y exigentes.

3. Instrucciones de uso

1. Descargar la carpeta con la implementación, anexada en la entrega junto a este informe.
2. Ejecutar los archivos java `ServidorHoroscopoRMI`, `ServidorPronosticoRMI` y `ServidorCentralRMI` 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 clientes.ClienteRMI`
4. Ingresar los datos que pide el cliente. Se utiliza `ENTER` como botón de confirmación.
5. Esperar unos segundos para obtener la respuesta.

4. Conclusiones

En este trabajo se desarrolló un sistema distribuido en Java utilizando RMI que consta de un cliente, un servidor central y dos servidores secundarios especializados en horóscopos y pronósticos del clima. Se diseñaron siete clases principales que implementan la estructura teórica del sistema.

La implementación de interfaces remotas y su correspondiente registro en el RMI Registry permitió una comunicación remota eficiente y modular entre componentes, facilitando la invocación de métodos distribuidos sin preocuparse por los detalles de la red.

El servidor central no solo actúa como intermediario para las solicitudes del cliente, sino que también incorpora una caché sincronizada que almacena respuestas recientes para reducir la carga y mejorar el tiempo de respuesta, evitando consultas repetidas a los servidores secundarios.

Para mejorar la flexibilidad y la facilidad de mantenimiento, se utilizó un archivo externo de configuración (config.properties), que permite cambiar parámetros como el host y puerto de los servidores sin necesidad de recompilar el código. En nuestro caso, se configuró el sistema para ejecutarse sobre localhost, facilitando las pruebas en un entorno controlado.

Bibliografía

[1] Java cryptography architecture (jca) reference guide, 2014. Último acceso: 27 de abril de 2025. URL: https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html.

[2] Java secure socket extension (jsse) reference guide, 2014. Consultado el 27 de abril de 2025. URL: https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html.