Universidad Técnica Estatal de Quevedo

Facultad de Ciencias de la Ingeniería

Integrantes:

Miguel Ángel Piña Paredes

John Kleiner Vera Macias

Valeska Sofia Chica Valfre

Jordy Alejandro Vilcacundo Chiluisa

Asignatura:

Aplicaciones Distribuidas

Docente:

Ing. Gleiston Cicerón Guerrero Ulloa

Tema:

Remote Method Invocation (RMI)

UTEQ

UNIVERSIDAD TÉCNICA ESTATAL DE QUEVEDO

Contenido

Ilustraciones

1. Introducción

En la era actual, la integración de sistemas informáticos distribuidos se ha convertido en un componente esencial para enfrentar los desafíos de conectividad y escalabilidad. En este contexto, el paradigma de Remote Method Invocation (RMI) ha emergido como una solución robusta para facilitar la comunicación entre objetos distribuidos en el entorno Java [1].

El RMI ofrece a los desarrolladores la capacidad de superar las barreras físicas de los sistemas individuales, posibilitando la creación de aplicaciones que pueden operar de manera eficiente a través de nodos geográficamente dispersos. Este informe se sumerge en los cimientos, la arquitectura y la implementación del RMI, desglosando su papel central en el desarrollo de sistemas distribuidos [1].

A lo largo de este documento, exploraremos conceptos prácticos mediante un ejemplo que ilustrará la aplicabilidad directa de RMI en situaciones del mundo real. También abordaremos los desafíos de seguridad inherentes a la comunicación remota y reflexionaremos sobre la relevancia continua de RMI en el dinámico panorama del desarrollo de software distribuido.

1.1. Definición de RMI

La Invocación Remota de Métodos (RMI) constituye una tecnología fundamental en el ámbito de la computación distribuida, permitiendo la interacción entre objetos y posibilitando la ejecución de funciones de manera remota, sin depender de la infraestructura de red. Este proceso implica que un cliente realiza una solicitud, la cual es procesada por un servidor que automáticamente devuelve una respuesta [2].

El protocolo de funcionamiento propio de RMI facilita la comunicación efectiva entre componentes distribuidos, y su capacidad para ejecutarse en cualquier máquina virtual de Java confiere una gran flexibilidad a esta tecnología. Además, la capacidad

de RMI para atravesar cortafuegos y establecer conexiones de manera confiable contribuye a su utilidad en entornos diversos [2].

A pesar de sus ventajas, RMI presenta desafíos y limitaciones que los desarrolladores deben tener en cuenta. La necesidad de configuración y mantenimiento manual puede resultar una tarea laboriosa, aunque necesaria, para garantizar el correcto funcionamiento del sistema. Además, existe la posibilidad de que el rendimiento se vea afectado en comparación con otras tecnologías más eficientes [2].

Uno de los desafíos más notables para RMI es la amenaza de volverse obsoleto con el surgimiento de métodos alternativos de comunicación, como las API REST. Estos enfoques han ganado popularidad debido a su simplicidad, escalabilidad y la capacidad de adaptarse a diferentes lenguajes de programación. En este contexto, los desarrolladores deben evaluar cuidadosamente la idoneidad de RMI en comparación con las soluciones más modernas y considerar la posible transición a tecnologías que mejor se alineen con las necesidades y tendencias actuales en el desarrollo de software distribuido [2].

Autor	RMI
Fujie Gao	Sistema complejo, que incluye fenómenos como la interacción de la interfaz de choque, la inestabilidad hidrodinámica, el flujo multifásico y la mezcla turbulenta [27].
Ahmet Sayar	Tipo de llamada a procedimiento remoto que es independiente de la red, ligera y totalmente portátil [28].
Daniel Tejera	Se basa en un hilo que llama a un método en una computadora remota, en donde un servidor obtiene esta invocación, ejecuta el método apropiado y devuelve los resultados del método y el cliente espera hasta que se reciba esta información [29].
Sachin Bagga	Es una forma factible para llamar a un objeto que reside en alguna máquina remota [2].
Minh Le	Es una biblioteca y un marco de nivel relativamente bajo que se basa en llamadas a métodos [30].
Sachin Bagga	Método que permite la comunicación entre los sistemas maestro y esclavo, y proporciona un servicio de establecimiento de conexión confiable que incluso puede pasar a través del firewall [20].
Jan Vimmr	Tecnología que permite llamar a un método en una máquina virtual remota y, es fácilmente aplicable a una red informática heterogénea

Ilustración 1: Tabla de autores y su criterio sobre RMI [2]

1.2. Propósito y beneficios de utilizar RMI

La Invocación Remota de Métodos (RMI) se erige como una herramienta fundamental en el ámbito de la computación distribuida, cumpliendo con diversos propósitos y brindando beneficios clave para el desarrollo de sistemas distribuidos [2].

1.2.1. Propósitos

- **Invocación remota eficiente:** RMI posibilita que los clientes realicen llamadas a métodos en servidores remotos a través de la red. Esta capacidad es esencial para la creación de aplicaciones distribuidas, ya que permite la ejecución de operaciones en ubicaciones remotas de manera eficiente y coherente [2].
- **Facilitación de la comunicación entre sistemas distribuidos:** Uno de los propósitos fundamentales de RMI es facilitar la comunicación y la interacción entre sistemas distribuidos. Al permitir la invocación de métodos de forma remota, RMI contribuye a la creación de sistemas cohesivos en entornos donde los componentes pueden residir en diferentes ubicaciones geográficas o lógicas [2].

1.2.2. Beneficios

- **Abstracción de la comunicación remota:** RMI proporciona una abstracción efectiva de la complejidad asociada con la comunicación a través de la red. Esto permite que los clientes llamen a métodos en objetos remotos de manera transparente, sin necesidad de preocuparse por los detalles de implementación subyacentes. Esta abstracción simplifica el desarrollo y promueve una interfaz intuitiva [3].
- **Simplificación del desarrollo:** El modelo de programación orientado a objetos de RMI simplifica significativamente el desarrollo de aplicaciones distribuidas. Al tratar los objetos remotos de manera similar a los locales, los desarrolladores pueden concentrarse en la lógica de la aplicación sin verse abrumados por la complejidad de la comunicación a larga distancia [3].
- **Escalabilidad y modularidad:** La separación de la lógica del cliente y del servidor favorece la escalabilidad y modularidad en sistemas distribuidos.

Esto permite a los desarrolladores diseñar aplicaciones que pueden evolucionar de manera independiente, facilitando la adaptación a cambios y mejoras sin afectar toda la arquitectura [3].

1.3. Contexto en el que se utiliza comúnmente

La Invocación Remota de Métodos (RMI) desempeña un papel crucial en sistemas distribuidos al posibilitar que los clientes ejecuten métodos en objetos remotos a través de la red. Esta funcionalidad es esencial para facilitar la comunicación entre componentes de software que operan en entornos distintos, permitiendo una interacción fluida y colaborativa entre ellos de manera transparente [3].

En el ámbito del desarrollo de aplicaciones empresariales en Java, el RMI ha sido ampliamente adoptado. Aunque ha demostrado ser eficiente para la invocación remota de métodos, se ha enfrentado a desafíos significativos. Uno de estos desafíos radica en su limitada interoperabilidad con tecnologías no Java, lo que puede presentar obstáculos en entornos heterogéneos donde se utilizan múltiples lenguajes de programación [3], [4]

Otro desafío notable es la dificultad para atravesar proxies y firewalls, lo que puede afectar la conectividad en redes corporativas. Este problema puede requerir configuraciones adicionales y medidas de seguridad para garantizar una comunicación sin problemas entre los sistemas distribuidos [4].

Además, el RMI ha experimentado preocupaciones relacionadas con la seguridad debido a la descarga de código. La capacidad de descargar y ejecutar código de manera remota plantea riesgos de seguridad, y los desarrolladores deben abordar estos problemas mediante prácticas sólidas de codificación y configuración segura para mitigar posibles amenazas [4].

A pesar de estos desafíos, el RMI ha encontrado su lugar en entornos de aplicaciones empresariales debido a su eficiencia en la invocación remota de métodos. Los equipos de desarrollo han superado estos problemas mediante la implementación de soluciones específicas y prácticas de seguridad robustas. A medida que evolucionan las tecnologías, es esencial evaluar constantemente la idoneidad del RMI en comparación con alternativas emergentes y considerar cómo abordar los

desafíos específicos de interoperabilidad, seguridad y conectividad en el desarrollo de sistemas distribuidos [4]

2. Arquitectura de RMI

La arquitectura de RMI se basa en un modelo cliente-servidor, donde el cliente invoca un método en un objeto remoto, y el servidor ejecuta el método y devuelve el resultado al cliente. El cliente y el servidor se comunican a través de una red utilizando el mecanismo de serialización incorporado en Java, que permite convertir objetos en un flujo de bytes y transmitirlos a través de la red [5].

RMI proporciona varios componentes clave, incluyendo:

1. **Interfaz remota**: Esta es la interfaz que define los métodos que se pueden invocar de forma remota. La interfaz remota debe extender la interfaz java.rmi.Remote, y cada método debe declarar la excepción java.rmi.RemoteException [5].
2. **Objeto remoto**: Este es el objeto que implementa la interfaz remota y proporciona la implementación real de los métodos que se pueden invocar de forma remota. El objeto remoto debe extender la clase java.rmi.server.UnicastRemoteObject, que proporciona la infraestructura necesaria para la invocación remota de métodos [5].
3. **Registro**: Este es un repositorio central que permite a los clientes buscar objetos remotos por nombre. El registro se implementa como un servidor que se ejecuta en un puerto específico y escucha las solicitudes de los clientes [5].
4. **Stub y esqueleto**: Estos son los componentes del lado del cliente y del servidor que manejan el empaquetado y desempaquetado de los parámetros y valores de retorno del método. El stub es generado por el compilador de RMI y es responsable de transmitir las invocaciones de método a través de la red. El esqueleto se genera en tiempo de ejecución y es responsable de recibir las invocaciones de método e invocar el método correspondiente en el objeto remoto [5].

Además, la arquitectura RMI implica la exportación de objetos remotos, la implementación de interfaces remotas y la utilización de clases de utilidad como `UnicastRemoteObject` para facilitar la comunicación remota [6].

Esta arquitectura permite que los objetos en diferentes máquinas virtuales de Java se comuniquen de manera transparente, lo que facilita el desarrollo de aplicaciones distribuidas en Java [6].

2.1. Componentes clave de la arquitectura RMI

Estos componentes forman la base de la arquitectura de RMI y son fundamentales para la implementación de la comunicación de objetos distribuidos en entornos Java.

2.1.1. Interfaz remota

Una interfaz remota en RMI define los métodos que pueden ser invocados de forma remota por clientes distribuidos. Estas interfaces extienden la interfaz java.rmi.Remote y cada método declarado en la interfaz remota debe lanzar la excepción java.rmi.RemoteException para manejar posibles errores de comunicación remota. La interfaz remota actúa como un contrato que especifica qué operaciones pueden ser realizadas de forma remota en un objeto [5].

2.1.2. Objeto remoto

Un objeto remoto en RMI es una instancia de una clase que implementa una interfaz remota. Este objeto proporciona la implementación real de los métodos definidos en la interfaz remota y puede ser invocado de forma remota por clientes distribuidos. El objeto remoto debe extender la clase java.rmi.server.UnicastRemoteObject para habilitar la invocación remota de métodos y la comunicación a través de la red [5].

2.1.3. Registro

El registro RMI actúa como un repositorio central que permite a los clientes buscar y obtener referencias a objetos remotos por nombre simbólico. Funciona como un servicio de directorio que mapea nombres de objetos a sus ubicaciones en la red. Los objetos remotos se registran en el registro con un nombre único, lo que permite a los clientes buscar y acceder a los objetos remotos de manera eficiente [5].

2.1.4. Stub y Esqueleto

El stub y el esqueleto son componentes esenciales para la comunicación entre clientes y servidores en RMI. El stub es generado por el compilador de RMI y actúa como un representante local del objeto remoto en el cliente. Se encarga de empaquetar y transmitir las invocaciones de método a través de la red. El esqueleto se genera dinámicamente en tiempo de ejecución y se encarga de recibir las invocaciones remotas, desempaquetarlas y dirigirlas al objeto remoto correspondiente para su ejecución [5].

2.2. Descripción de roles: Cliente, Servidor y registro RMI

En la arquitectura de Remote Method Invocation (RMI), diferentes roles desempeñan funciones específicas para permitir la comunicación eficaz entre objetos distribuidos. A continuación, se detalla la descripción de los roles clave: Cliente, Servidor y Registro RMI.

2.2.1. Cliente

El cliente desempeña un papel central al iniciar la comunicación remota. Su responsabilidad principal es invocar métodos en objetos distribuidos, pero para hacerlo, el cliente necesita obtener una referencia al objeto remoto deseado. Este proceso generalmente implica la consulta al registro RMI, donde los objetos remotos están registrados con nombres únicos. Una vez que el cliente obtiene la referencia al objeto remoto, puede interactuar con él, invocando métodos como si fueran métodos locales. El cliente actúa como el catalizador de la comunicación, iniciando las solicitudes y gestionando las respuestas obtenidas de los objetos remotos en el servidor [1].

2.2.2. Servidor

El servidor es el proveedor de servicios remotos. Contiene objetos que implementan interfaces remotas y, por lo tanto, ofrecen funcionalidades específicas que los clientes pueden invocar de manera remota. Estos objetos se registran en el registro RMI, lo que permite a los clientes descubrirlos y acceder a sus servicios. El servidor, al recibir las solicitudes remotas de los clientes, ejecuta la lógica correspondiente en los objetos remotos y devuelve los resultados al cliente. En esencia, el servidor es la

entidad que responde a las solicitudes remotas y proporciona los servicios solicitados por los clientes [1].

2.2.3. Registro RMI

El registro RMI actúa como un punto de encuentro para clientes y servidores. Funciona como un servicio de directorio, permitiendo a los servidores registrar objetos remotos asociados con nombres únicos. Los clientes utilizan el registro para buscar objetos remotos de interés. Cuando un cliente necesita acceder a un servicio específico, consulta el registro RMI para obtener la referencia al objeto remoto. En resumen, el registro RMI facilita la conexión entre clientes y servidores al proporcionar un mecanismo centralizado para descubrir y obtener referencias a objetos remotos en un entorno distribuido [1].

3. Como funciona RMI

El funcionamiento de RMI se basa en la capacidad de serializar objetos para convertirlos en una representación serial adecuada para su transmisión a través de una conexión de red y luego reconstruirlos al recibirlos. Esto es fundamental para los métodos remotos que toman objetos como parámetros, así como para los objetos que tienen objetos o valores de retorno [7].

3.1. Explicación del proceso de invocación de métodos remotos

En el contexto del desarrollo de aplicaciones distribuidas en Java, la Invocación Remota de Métodos (RMI) emerge como una herramienta esencial. Este enfoque permite la invocación eficiente de métodos en objetos situados en sistemas remotos, ofreciendo una abstracción que independiza a los desarrolladores de las complejidades específicas de la plataforma. El proceso de implementación, detallado paso a paso en este artículo, abarca desde la concepción de interfaces y la implementación de clases que las siguen, hasta la creación de componentes esenciales para servidores y clientes [8].

En primer lugar, se destaca la importancia de la creación de interfaces, que actúan como contratos que especifican los métodos que podrán ser invocados de manera remota. Posteriormente, la implementación de clases que sigan estas interfaces es esencial, ya que constituyen la lógica real detrás de los métodos remotos. El artículo

subraya la necesidad de compilar estas clases y generar stubs y esqueletos, componentes cruciales para facilitar la comunicación entre el cliente y el servidor [8].

En el desarrollo del componente del servidor, se aborda el registro de objetos remotos en un servicio de nombres, lo que permite a los clientes descubrir y acceder a estos objetos. El acceso a interfaces y referencias a objetos remotos desde el lado del cliente se presenta como una parte fundamental del proceso, estableciendo la conexión necesaria para la invocación remota de métodos [8].

Para proporcionar una comprensión práctica, el artículo incluye un ejemplo detallado que guía a los lectores a través de cada fase del proceso de desarrollo e implementación de una aplicación RMI. Desde la conceptualización hasta la ejecución práctica, este recurso sirve como un valioso compañero para aquellos que buscan no solo comprender los principios teóricos de RMI, sino también adquirir habilidades prácticas para su implementación efectiva en proyectos reales [8].

3.2. Comunicación entre objetos distribuidos

La Comunicación entre Objetos Distribuidos (COD) constituye un aspecto fundamental en el ámbito de la programación, referente a la interacción entre objetos que operan en distintos sistemas, generalmente a través de una red. En el contexto específico de Java, una tecnología clave para facilitar esta comunicación es la Invocación de Método Remoto (RMI)[9] .

RMI representa un modelo que posibilita que un objeto en una Máquina Virtual de Java (JVM) invoque un método en un objeto que se está ejecutando en una JVM remota, permitiendo así la transmisión de resultados entre ellos. Este modelo involucra una estructura cliente-servidor, en la cual el servidor crea un objeto y lo expone para su acceso remoto, mientras que el cliente busca dicho objeto en un registro y obtiene una referencia para interactuar con él [9].

La arquitectura de RMI se compone de varios componentes esenciales, tales como stubs y esqueletos, la interfaz remota, el objeto remoto y las capas de referencia y transporte. Los stubs y esqueletos actúan como intermediarios o proxies para el cliente y el servidor respectivamente, gestionando el código relacionado con la red

y los sockets para liberar al cliente y al servidor de la complejidad directa de la comunicación a través de la red [9].

En la implementación práctica de RMI, se definen interfaces remotas que especifican los métodos disponibles para los clientes como un servicio. Estas interfaces siempre extienden la interfaz java.rmi.Remote, y los métodos deben indicar que lanzan una RemoteException para manejar posibles problemas de comunicación [9].

Adicionalmente, RMI ofrece funcionalidades como la carga dinámica de stubs y esqueletos, así como la gestión de argumentos y valores de retorno en los métodos remotos. Es importante destacar que los argumentos deben ser serializables y que los valores de retorno también deben cumplir con esta condición para asegurar la transferencia efectiva de información entre las JVMs [9].

3.3. Paso de parámetros y resultados

En el contexto de Remote Method Invocation (RMI), la habilidad para invocar métodos en objetos remotos situados en sistemas distintos constituye una característica fundamental. Una vez que se adquiere una referencia al objeto remoto, los métodos de dicho objeto pueden ser invocados de manera análoga a los métodos de objetos locales. La fortaleza de RMI radica en su capacidad para gestionar los detalles de red necesarios para estas invocaciones remotas, ofreciendo así una experiencia casi transparente al programador Java [10].

En el trasfondo de esta interacción remota, RMI emplea flujos de bytes para la transferencia de datos y la invocación de métodos. La infraestructura RMI automatiza el proceso de serialización y deserialización de los parámetros que se transmiten entre el cliente y el objeto remoto, simplificando la complejidad de la comunicación a través de la red [10].

RMI presenta soporte tanto para el paso de parámetros por valor como por referencia. En el caso del paso por valor, los parámetros son serializados y enviados al objeto remoto, permitiendo que la información viaje de manera eficiente a través de la red. Por otro lado, en el paso por referencia, el objeto remoto recibe una referencia al objeto original, posibilitando una conexión directa con la instancia original [10].

Un aspecto destacado de RMI es su capacidad para gestionar el paso de objetos complejos como parámetros. Sin embargo, para que esto sea posible, estos objetos deben ser serializables, es decir, deben implementar la interfaz Serializable. Este enfoque garantiza que la estructura del objeto pueda ser convertida en una secuencia de bytes y reconstruida de manera efectiva en el extremo receptor, asegurando así una comunicación eficiente y precisa entre el cliente y el objeto remoto [10].

4. Implementación de RMI

La implementación del RMI (Remote Method Invocation) en Java es un enfoque para la comunicación entre procesos que permite a los programas Java invocar métodos en objetos remotos de manera transparente [11]. Aquí hay algunos aspectos clave sobre la implementación del RMI:

1. Definición de interfaces remotas: La clave para desarrollar sistemas basados en RMI es definir las interfaces para los objetos remotos. En RMI, todos los objetos que proporcionan una interfaz remota deben extender la interfaz predefinida `Remote` y declarar que lanzan `RemoteException` [11].

2. Creación de objetos remotos: Se deben crear objetos remotos que implementen las interfaces remotas definidas. Estos objetos deben extender `java.rmi.server.UnicastRemoteObject` y lanzar `RemoteException` en sus métodos [11].

3. Generación de stubs y esqueletos: El compilador de RMI genera stubs y esqueletos para facilitar la comunicación entre el cliente y el servidor. El stub actúa como un representante local del objeto remoto en el cliente, mientras que el esqueleto maneja las llamadas remotas en el servidor [11].

4. Registro RMI: El registro RMI (RMI registry) actúa como un servicio de nombres para objetos remotos. Permite a los clientes buscar objetos remotos registrados y obtener referencias a ellos [11].

5. Comunicación cliente-servidor: El cliente obtiene una referencia remota al objeto en el servidor a través del registro RMI y luego invoca métodos en el objeto remoto como si fueran locales [11].

6. Manejo de excepciones: Dado que la comunicación remota puede fallar, es crucial manejar las excepciones de `RemoteException` adecuadamente en la implementación del RMI [11].

4.1. Uso de interfaces remotas

El concepto de interfaces remotas en RMI se basa en el uso de interfaces en Java. Las interfaces son una forma de definir un conjunto de métodos que una clase debe implementar. En RMI, las interfaces remotas se utilizan para definir los métodos que un objeto remoto debe proporcionar [12].

El uso de interfaces remotas en RMI tiene varias ventajas. En primer lugar, permite que el código que define el comportamiento y el código que implementa el comportamiento se ejecuten en máquinas virtuales Java separadas. Esto significa que el cliente y el servidor pueden estar en diferentes máquinas y aun así comunicarse de manera efectiva [12].

En segundo lugar, el uso de interfaces remotas permite que el cliente haga llamadas a métodos en el objeto remoto como si fuera un objeto local. Esto significa que el cliente no necesita conocer los detalles de la implementación del objeto remoto, lo que simplifica el código del cliente y lo hace más fácil de mantener [12].

En tercer lugar, el uso de interfaces remotas en RMI permite que el servidor proporcione servicios a múltiples clientes simultáneamente. Esto se logra mediante la creación de múltiples objetos remotos que implementan la misma interfaz remota. Cada objeto remoto se ejecuta en su propia máquina virtual Java y puede atender a múltiples clientes simultáneamente [12].

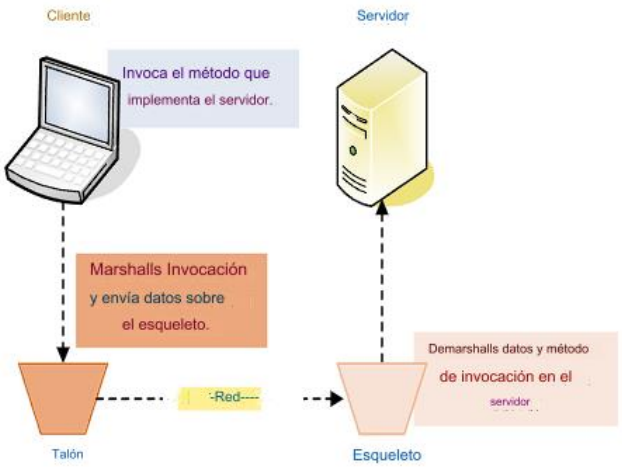

Ilustración 2: Una llamada RMI básica con un código auxiliar y un esqueleto mediante Interfaces remotas [11]

4.2. Creación de objetos remotos

La creación de objetos remotos en RMI implica varios pasos clave. Primero, se define una interfaz remota que especifica los métodos que el objeto remoto proporcionará. Luego, se implementa esta interfaz en una clase que se ejecutará en el servidor. Una vez que el objeto remoto está implementado, se registra en el registro RMI para que los clientes puedan encontrarlo y acceder a él [12].

El registro RMI es un servicio que permite a los clientes buscar objetos remotos por nombre y obtener referencias a ellos. Una vez que el objeto remoto está registrado en el registro RMI, los clientes pueden obtener una referencia a él y llamar a sus métodos como si fuera un objeto local [12].

La creación de objetos remotos en RMI también implica la generación de clases de stub y esqueleto. El stub es una clase que se ejecuta en el cliente y actúa como un proxy para el objeto remoto. El esqueleto es una clase que se ejecuta en el servidor y actúa como un proxy para el cliente. Estas clases se generan automáticamente a partir de la interfaz remota y se utilizan para facilitar la comunicación entre el cliente y el servidor [12].

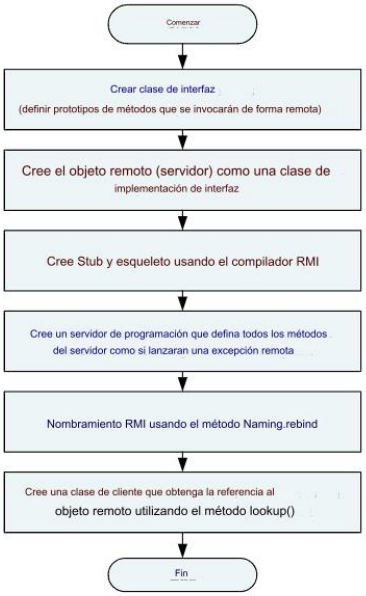

Ilustración 3: Pasos de diseño de RMI [11]

Para crear objetos remotos en RMI, se deben seguir los siguientes pasos:

1. Crear una interfaz remota: La interfaz remota define los métodos que se pueden invocar de forma remota. Todos los métodos de la interfaz deben lanzar `RemoteException`. Por ejemplo:

```
```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface MyRemoteInterface extends Remote {
    public String sayHello() throws RemoteException;
}
``
```

2. Implementar la interfaz remota: La implementación de la interfaz remota debe extender `java.rmi.server.UnicastRemoteObject` y proporcionar una implementación para cada método de la interfaz. Por ejemplo:
```

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class MyRemoteObject extends UnicastRemoteObject implements MyRemoteInterface {
    public MyRemoteObject() throws RemoteException {
        super();
    }
    public String sayHello() throws RemoteException {
        return "Hello, world!";
    }
}
```

3. Crear un objeto remoto: Para crear un objeto remoto, se debe instanciar la implementación de la interfaz remota y registrarla en el registro RMI. Por ejemplo:

```
import java.rmi.Naming;
import java.rmi.RemoteException;
public class MyRemoteServer {
    public static void main(String[] args) {
        try {
            MyRemoteInterface obj = new MyRemoteObject();
            Naming.rebind("MyRemoteObject", obj);
            System.out.println("MyRemoteObject bound in registry");
        } catch (RemoteException e) {
            System.out.println("Error: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

4.3. Registro de objetos remotos en el registro RMI

El registro RMI es un servicio que permite a los clientes buscar objetos remotos por nombre y obtener referencias a ellos. Cuando se registra un objeto remoto en el registro RMI, se le asigna un nombre único que se puede utilizar para buscarlo. Esto permite a los clientes encontrar y acceder a objetos remotos de manera sencilla y eficiente [12].

El registro RMI se ejecuta en una máquina específica y mantiene una lista de nombres de objetos remotos y sus referencias. Cuando un cliente necesita acceder a un objeto remoto, puede consultar el registro RMI utilizando el nombre del objeto y obtener una referencia a él. Esta referencia se puede utilizar para llamar a los métodos del objeto remoto como si fuera un objeto local [12].

El registro RMI es una parte fundamental de la infraestructura de RMI, ya que facilita la comunicación entre clientes y objetos remotos. Proporciona un mecanismo centralizado para la gestión de objetos remotos y permite a los clientes encontrar y acceder a estos objetos de manera transparente [12].

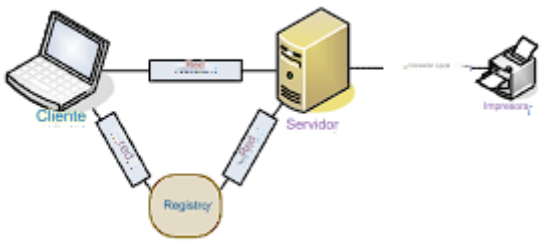

Ilustración 4: Diagrama de arquitectura de registro RMI [11]

Para registrar objetos remotos en el registro RMI, se utiliza la clase `java.rmi.Naming`. Aquí está la información relevante sobre el registro de objetos remotos en el registro RMI:

1. Uso de `java.rmi.Naming`: La clase `java.rmi.Naming` proporciona métodos estáticos para asociar nombres con objetos remotos y para buscar objetos remotos por nombre en el registro RMI .

2. Registro de un objeto remoto: Para registrar un objeto remoto en el registro RMI, se utiliza el método `rebind` de la clase `java.rmi.Naming`. Por ejemplo:

```java
MyRemoteInterface obj = new MyRemoteObject();
Naming.rebind("MyRemoteObject", obj);
```

En este ejemplo, el objeto remoto `obj` se registra en el registro RMI con el nombre "MyRemoteObject".

3. Búsqueda de un objeto remoto: Para buscar un objeto remoto en el registro RMI, se utiliza el método `lookup` de la clase `java.rmi.Naming`. Por ejemplo:

```java
MyRemoteInterface obj = (MyRemoteInterface) Naming.lookup("rmi://localhost/MyRemoteObject");
```

5. Seguridad de RMI

El tema de la seguridad es muy importante para cualquier sistema, en especial los sistemas distribuidos, ya que estos sistemas suelen manejar información sensible de los usuarios o su vez registros cruciales para la empresa u organización que utilice este sistema.

La seguridad de los métodos remotos debe contemplar aspectos como la autenticación, autorización, integridad y confidencialidad. Es así, como la implementación estándar de RMI admite la utilización de sockets personalizados a través de fábricas de sockets tanto para el servidor como para el cliente. También, se ofrecen sockets SSL personalizados para RMI, que permiten lograr integridad y confidencialidad. Es decir, la autenticación y autorización del cliente deben ser determinadas por el objeto remoto que ofrece el servicio [6], [13].

El protocolo no permite la inclusión de credenciales, como nombre de usuario y contraseña, en una llamada a un método. Es decir, cualquier cliente que conozca el nombre público del control remoto puede conectar fácilmente el servicio de objeto y el puerto del registro RMI con el objeto remoto. Por lo tanto, la autenticación debe

ser gestionada por el registro RMI, que devuelve el código auxiliar del objeto solo a clientes legítimos, mientras que la autorización debe ser manejada por el objeto remoto [6].

La aplicación del patrón de diseño de proxy en RMI plantea interrogantes sobre la seguridad [6]. Según las directrices de RMI, se requiere tener en cuenta las siguientes características para garantizar la seguridad:

- **Secreto:** La información contenida en el sistema debe ser accesible únicamente para usuarios autorizados [6].
- **Privacidad:** Se debe evitar el uso indebido de la información, asegurando la confidencialidad y la protección de la privacidad [6].

5.1. Desafíos de seguridad en sistemas distribuidos

La seguridad es un objetivo dinámico, influenciado por la interacción entre las nuevas tecnologías y las aplicaciones que las facilitan. En la actualidad, la mayoría de las computadoras están conectadas, al menos en algún momento, a algún tipo de red de comunicaciones. Muchas aplicaciones, desde el correo electrónico hasta el comercio electrónico, dependen de esta conectividad. En este contexto, la seguridad informática se refiere a la seguridad de los sistemas distribuidos. Asimismo, la seguridad de los sistemas distribuidos ya no se considera simplemente una especialización dentro de la seguridad informática, sino que está intrínsecamente vinculada a ella [7], [13].

La seguridad en aplicaciones distribuidas proporciona una visión más precisa de los avances actuales y también representa una apuesta adecuada por las preocupaciones de seguridad contemporáneas. En la actualidad, las aplicaciones distribuidas están adoptando cada vez más tecnologías web, las cuales, a su vez, están sujetas a una evolución constante. Por lo cual, existe la posibilidad de la que se creen o encuentre nuevas vulnerabilidades para el acceso a los datos y el robo de información [13].

5.2. Desafíos de seguridad en el contexto de RMI.

Los sistemas distribuidos actualmente se enfrentan a varios desafíos de seguridad, como la autenticación, la autorización, la integridad de los datos y la confidencialidad. En el contexto de RMI, algunos desafíos específicos incluyen:

- **Autenticación de Clientes y Servidores:** Garantizar que tanto el cliente como el servidor sean quienes dicen ser, evitando la suplantación de identidad [13].
- **Integridad de los Datos:** Proteger la integridad de los datos transmitidos entre sistemas distribuidos para evitar manipulaciones no autorizadas [13].
- **Confidencialidad:** Asegurar que la información sensible no sea accesible para usuarios no autorizados durante la transmisión [13].

5.3. Métodos para garantizar la seguridad RMI

La seguridad es muy importante, en especial para los sistemas distribuidos, es por ello por lo que RMI hace uso de los componentes de seguridad implícitos de Java, lo que permite que su estructura esté protegida cuando los clientes realizan descargas de ejecuciones. También, utiliza el modelo de seguridad definido para asegurar los marcos de los subprogramas adversarios, protegiendo así sus sistemas y los sistemas de códigos descargados que podrían ser potencialmente hostiles. En situaciones extremas, un servidor tiene la opción de rechazar la descarga de cualquier aplicación [13].

A continuación, se describen algunos de los métodos más usados para garantizar la seguridad RMI:

- **Implementación de Certificados SSL/TLS:** El uso de Secure Sockets Layer (SSL) o Transport Layer Security (TLS) proporciona una capa de seguridad adicional al cifrar la comunicación entre el cliente y el servidor, garantizando la confidencialidad y la integridad de los datos [13].
- **Configuración de Mecanismos de Autenticación:** Establecer mecanismos robustos de autenticación, como el uso de tokens de seguridad o la integración con sistemas de autenticación centralizados, contribuye a verificar la identidad de los participantes en la comunicación RMI [13].

- **Autorización basada en Roles:** Definir roles y permisos específicos para los usuarios y sistemas involucrados en la RMI. Esto asegura que solo las operaciones autorizadas se lleven a cabo, reduciendo el riesgo de acceso no autorizado [13].
- **Uso de Firewalls y Configuraciones de Red Seguras**: Implementar firewalls y configuraciones de red seguras ayuda a prevenir ataques externos y proteger la infraestructura de RMI contra accesos no autorizados [13].

6. GitHub

Enlace de GitHub: https://github.com/SrJordy/RMI.git

7. Conclusión

La implementación de Remote Method Invocation (RMI) en Java es un enfoque fundamental para la comunicación entre procesos en entornos distribuidos. A través de la definición de interfaces remotas, la creación de objetos remotos, la generación de stubs y esqueletos, y el registro en el RMI registry, se facilita la conexión entre clientes y servidores de manera transparente. La capacidad de RMI para gestionar la invocación eficiente de métodos en objetos remotos, así como el manejo de la serialización de objetos para su transmisión a través de la red, lo posiciona como una herramienta esencial en el desarrollo de aplicaciones distribuidas en Java.

8. Referencias

[1] David. Reilly and M. (Michael S. Reilly, *Java network programming and distributed computing*. Addison-Wesley, 2002.

[2] M. E. Guaraca Moyota, J. Jácome León, A. Pinchao Pujota, J. Silva Moran, and F. Imbaquingo, “Invocación de métodos remotos (RMI): una revisión de la literatura,” *INNOVATION & DEVELOPMENT IN ENGINEERING AND APPLIED SCIENCES*, vol. 5, no. 1, p. 9, Jul. 2023, doi: 10.53358/ideas.v5i1.869.

[3] M. Król, K. Habak, D. Oran, D. Kutscher, and I. Psaras, “Rice: Remote method invocation in ICN,” in *ICN 2018 - Proceedings of the 5th ACM Conference on Information-Centric Networking*, Association for Computing Machinery, Inc, Sep. 2018, pp. 1–11. doi: 10.1145/3267955.3267956.

[4] R. Jolly, “Monadic Remote Invocation,” Aug. 2017, [Online]. Available: http://arxiv.org/abs/1708.03882

[5] Institute of Electrical and Electronics Engineers. and IEEE Industrial Electronics Society., *ETFA 2010 : 15th IEEE International Conference on Emerging Techologies and Factory Automation : September 13-16, 2010 @ Bilbao, Spain*. IEEE, 2010.

[6] V. K. Dabhi and H. B. Prajapati, “Dissection of the internal workings of the RMI, possible enhancements and implementing authentication in standard Java RMI,” 2010.

[7] K. Deep and S. Toor, “Remote Method Invocation and Remote Procedure Call,” vol. 2, 2015.

[8] M. Mihailescu and S. Loredana Nita, “Remote Method Invocation (RMI) Articles » Languages » Java » General Remote Method Invocation (RMI),” 2015. [Online]. Available: http://www.codeproject.com/Articles/884158/Remote-Method-Invocatio...

[9] “Advanced Java Programming,” 2009.

[10] J. Graba, “Remote Method Invocation (RMI),” in *An Introduction to Network Programming with Java*, London: Springer London, 2013, pp. 129–149. doi: 10.1007/978-1-4471-5254-5_5.

[11] A. B. Sallow, “Design And Implementation Distributed System Using Java-RMI Middleware,” *Academic Journal of Nawroz University*, vol. 9, no. 1, p. 113, Feb. 2020, doi: 10.25007/ajnu.v9n1a550.

[12] N. Akter, F. Ahmed, and N. J. Shanta, “Effective Techniques to Improve Network Load Balancing for Parallel Computation Using RMI,” *International Journal of Innovative Research in Computer Science & Technology*, vol. 6, no. 6, pp. 117–120, Nov. 2018, doi: 10.21276/ijircst.2018.6.6.1.

[13] D. Gollmann, “Security in Distributed Applications.”