**Objetos distribuidos e invocación remota**

**Conceptos**

**Aplicación distribuida** es un sistema compuesto de programas que trabajan juntos desde diferentes procesos, usualmente en distintos computadores.

**Middleware:** es un software que facilita este proceso, proporcionando herramientas para que los desarrolladores puedan programar sobre una estructura de procesos y comunicación por mensajes. Utiliza protocolos de mensajes para que los procesos puedan interactuar de forma más sencilla, ofreciendo funciones como invocaciones remotas y gestión de eventos. Además, el middleware permite que se pueda implementar la aplicación en varios lenguajes de programación.

**Un middleware ofrece varias ventajas en sistemas distribuidos:**

* **Transparencia frente de ubicación**: Permite que el cliente no note si el procedimiento que llama se ejecute en el mismo proceso o en un proceso diferente.
* **Protocolos de comunicación**: Funciona de manera independiente de los protocolos de transporte subyacentes.
* **Hardware de los computadores**: Se encarga de empaquetar y desempaquetar mensajes, adaptándose al hardware de los computadores.
* **Sistema operativo**: Proporciona abstracciones que funcionan sin depender del sistema operativo que se esté usando.
* **Utilización de diversos lenguajes**: Permite desarrollar aplicaciones distribuidas en diferentes lenguajes de programación.

**Los conceptos de middleware se enfocan en cómo los procesos se comunican en sistemas distribuidos. Existen tres formas principales (Paradigmas) de invocación remota:**

* **Protocolos de petición-respuesta**: Un método básico para ejecutar operaciones remotas, que es la base para modelos más avanzados como RPC y RMI.
* **Remote Procedure Call (RPC)**: Es el(el primer modelo) modelo más común y facilita la comunicación entre un cliente y un servidor en diferentes máquinas.
* **Remote Method Invocation (RMI)**: Una extensión de RPC, que permite llamar a métodos de objetos en otros nodos, ideal para programación orientada a objeto.

**Protocolo de petición-respuesta**

El protocolo de petición-respuesta es una forma de comunicación entre un cliente y un servidor, en la cual el cliente hace una solicitud y el servidor responde.

Normalmente, esta comunicación es síncrona, lo que significa que el cliente espera bloqueado hasta recibir la respuesta. Esto hace que la comunicación sea fiable, ya que la respuesta del servidor actúa como una confirmación para el cliente.

También existe la comunicación asíncrona, donde el cliente no espera de inmediato la respuesta y puede recuperarla más tarde, lo cual es útil en ciertos escenarios donde no se necesita una respuesta inmediata.

**Protocolo Petición Respuesta Primitivas**

**Diagrama

Descripción generada automáticamente**

**public byte[] doOperation (RemoteRef s, int operationId, byte[] arguments)**

Envía un mensaje de solicitud al servidor remoto y devuelve la respuesta. Los argumentos especifican el servidor remoto, la operación que se invocará y los argumentos de esa operación.

**public byte[] getRequest ();**

Adquiere una solicitud de cliente a través del puerto del servidor.

**public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);**

Envía el mensaje de respuesta al cliente a su dirección de Internet y puerto.

**Protocolo Petición Respuesta Modelo de Fallos**

Si las tres funciones doOperation, getRequest y sendReply usan datagramas UDP, enfrentarán problemas de comunicación comunes en UDP:

* **Fallos por omisión:** El servidor podría no responder o los mensajes podrían perderse en el camino.
* **Fallos bizantinos:** Los mensajes pueden llegar duplicados o en un orden incorrecto.

**Para manejar fallos por omisión en *doOperation() de timeout* se proponen dos opciones:**

1. Retornar un indicador de fallo cuando no hay respuesta.
2. Reintentar la operación hasta obtener una respuesta.

**Para los fallos bizantinos (como mensajes duplicados o desordenados):**

1. Usar un identificador único (id\_cliente # id\_peticion) para descartar mensajes duplicados mientras la operación sigue en curso.
2. Si la operación ya se realizó y es idempotente (puede repetirse sin problemas), volver a ejecutarla; si no, llevar un historial de resultados para evitar duplicados.

**Protocolo Petición Respuesta Idempotencia**

Diagrama, Escala de tiempo

Descripción generada automáticamente

**Protocolo Petición Respuesta Estilos de protocolos RPC**

En los tipos de RPC (Remote Procedure Call) existen tres protocolos principales según cómo manejan los errores de comunicación:

* **Petición (R)**: El cliente envía la solicitud sin esperar confirmación.
* **Petición-Respuesta (RR)**: El cliente recibe una respuesta del servidor como confirmación.
* **Petición-Respuesta-Confirmación de la Respuesta (RRA)**: Intercambio en tres pasos donde el cliente confirma haber recibido la respuesta.

Diagrama

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

**Protocolo petición-respuesta HTTP: un ejemplo de protocolo RR**

HTTP es un protocolo de petición-respuesta donde el cliente (navegador) se comunica con el servidor (WebServer). Define los tipos de mensajes (como GET, PUT, POST)(son métodos) que permiten la interacción con todos los recursos. Además, HTTP permite:

* **Negociación de contenido:** El cliente puede especificar preferencias (como idioma o tipo de archivo), sus peticiones pueden tener información.
* **Autenticación:** Se usan credenciales para verificar la identidad del usuario.

**En HTTP sobre TCP, la comunicación cliente-servidor sigue estos pasos básicos:**

1. El cliente se conecta al puerto del servidor.
2. El cliente envía su petición.
3. El servidor responde.
4. La conexión se cierra.

Este ciclo se repite para cada interacción.

**Los métodos HTTP permiten diferentes acciones en los recursos de un servidor:**

1. **GET**: Solicita datos o ejecuta programas en el servidor según el URL.

* Si el URL se refiere a datos, entonces el servidor responderá enviando de

vuelta los datos indicados por el URL.

* Si el URL se refiere a un programa, entonces el servidor web ejecutará el

programa y devolverá su salida al cliente.

* Admite argumentos al URL

1. **HEAD**: Igual a GET, pero solo devuelve la información sobre los datos.
2. **POST**: especifica el URL de un recurso (por ejemplo, un programa) que puede tratar los datos proporcionados en el cuerpo de la petición.

* Proporcionar un bloque datos (por ejemplo, los obtenidos en un formulario) aun proceso gestión de datos como un servlet o un programa cgi.
* Enviar un mensaje a un tablón de anuncios, lista de correo o grupo de noticias.
* Modificar una base de datos con una operación de añadir registro.

1. **PUT**: indica que los datos aportados en la petición deben ser almacenados con la URL aportada como su identificador, ya sea como una modificación de datos existentes o como la creación de un recurso nuevo.
2. **DELETE**: Borra el recurso especificado por el URL (si está permitido).
3. **OPTIONS**: Informa al cliente de los métodos disponibles para un URL (por ejemplo, GET, HEAD, PUT) y sus requisitos especiales.
4. **TRACE**: Devuelve la petición para ayudar en la depuración.

Estos métodos son la base de cómo los clientes interactúan con recursos en HTTP.

**Protocolo petición-respuesta Mensaje HTTP**

Tabla

Descripción generada automáticamente

Códigos de estado de respuesta HTTP indican el estado (exitoso o no) de una petición

HTTP. Las respuestas se agrupan en cinco clases:

Respuestas informativas (100–199),

Respuestas satisfactorias (200–299),

Redirecciones (300–399),

Errores de los clientes (400–499),

Interfaz de usuario gráfica, Aplicación, Tabla

Descripción generada automáticamente con confianza mediaErrores de los servidores (500–599).

**Llamada a Procedimiento Remoto RPC**

La llamada a procedimiento remoto (RPC) permite que la programación en sistemas distribuidos se sienta como la programación normal. Al extender las llamadas de procedimiento a redes, RPC logra ocultar la complejidad de la distribución y simplificar el desarrollo de sistemas distribuidos.

**RPC**

La llamada a procedimiento remoto (RPC) facilita la programación en sistemas distribuidos, haciendo que se sienta como la programación tradicional. Esto se logra al extender las llamadas de procedimiento a redes, logrando así transparencia en la distribución.

En RPC, un cliente puede llamar a un procedimiento que se ejecuta en un servidor, de forma similar a cómo invoca un método en su propio sistema. Además, los servidores también pueden actuar como clientes para llamar a otros servidores, formando así cadenas de RPC.

El servidor define una interfaz de servicio que detalla los procedimientos que pueden ser llamados de forma remota. RPC se basa en un protocolo de petición-respuesta y permite implementar distintas semánticas de invocación (como “al menos una vez” o “como máximo una vez”) para manejar los errores de comunicación.

Los mensajes de RPC son similares a los de RMI, pero no incluyen el campo de referencia a objetos, ya que RPC se centra en llamadas a procedimientos y no en objetos remotos.

El soporte para RPC es parecido al de RMI, pero sin módulos de referencias remotas, ya que RPC no usa objetos. En el lado del cliente, cada procedimiento en la interfaz del servicio tiene un procedimiento de resguardo (como un proxy) que actúa como si fuera un método local. Sin embargo, en lugar de ejecutarlo directamente, empaqueta el identificador del procedimiento y los argumentos en un mensaje que envía al servidor.

En el servidor, un distribuidor selecciona el procedimiento adecuado según el mensaje recibido. El procedimiento de resguardo del servidor desempaqueta los argumentos, llama al procedimiento correspondiente y empaqueta el resultado para enviarlo de vuelta al cliente.

En RPC, el distribuidor del servidor utiliza el identificador del procedimiento en el mensaje del cliente para elegir el procedimiento de resguardo adecuado. Este procedimiento de resguardo, similar a un esqueleto, desempaqueta los argumentos del mensaje, llama al procedimiento de servicio correspondiente (que implementa las funciones de la interfaz de servicio) y luego empaqueta el resultado para enviarlo de vuelta al cliente como respuesta.

Diagrama

Descripción generada automáticamente

**Interfaces**

En sistemas distribuidos, los módulos se ejecutan en procesos separados, por lo que no pueden acceder directamente a las variables de otros módulos en diferentes procesos. Las interfaces permiten la comunicación entre módulos, pero en RPC o RMI no permiten el acceso directo a variables.

En CORBA, las interfaces en IDL pueden incluir atributos, pero estos no se acceden directamente; en su lugar, se crean automáticamente procedimientos de lectura y escritura para manejarlos, manteniendo la separación entre procesos.

En RPC, **los parámetros de entrada** se envían al módulo remoto dentro del mensaje de petición y se usan como argumentos para la operación en el servidor.

**Los parámetros de salida** se devuelven al cliente en el mensaje de respuesta, ya sea como resultado o reemplazando valores de los argumentos originales. Es importante recordar que los punteros no son válidos entre procesos, por lo que no pueden pasarse ni como argumentos ni como valores de retorno en RPC.

**Interfaces de servicio:** En el modelo cliente-servidor, c/ servidor proporciona procedimientos disponibles para clientes. El término se emplea para referirse a la especificación de los procedimientos que ofrece un servidor.

Por ejemplo, un servidor de archivos proporcionará procedimientos para leer y escribir archivos. Define los tipos de los argumentos de entrada y salida.

**Interfaces remotas:** En el modelo de objetos distribuidos, una interfaz remota especifica los métodos de un objeto que están disponibles para su invocación por objetos de otros procesos, y define los tipos de los argumentos de entrada y de salida.

Sin embargo, la gran diferencia es que los métodos en las interfaces remotas pueden pasar objetos como argumentos y como resultados de los métodos.

**Lenguajes de definición de interfaces.**

* Mecanismo RMI con un lenguaje de programación concreto.
* Incluye una notación apropiada para definir interfaces,
* Debe permitir relacionar los parámetros de entrada y de salida con el uso habitual de los parámetros en ese lenguaje.

JavaRMI es un ejem. en el que se ha añadido un mecanismo RMI a un LP

**Los lenguajes de definición de interfaces (IDL)**

* Permiten que los objetos implementados en lenguajes diferentes se invoquen

unos a otros.

* Un IDL proporciona notación para definir interfaces en la cual cada uno de los parámetros de un método se podrá describir como de entrada o de salida además de su propia especificación de tipo.

**Semánticas de invocación**

Las medidas de tolerancia a fallos en sistemas distribuidos ayudan a asegurar la confiabilidad de la comunicación entre clientes y servidores. Aquí se presentan tres mecanismos clave:

**Reintento del mensaje de petición:**

* Consiste en reenviar el mensaje de solicitud desde el cliente al servidor repetidamente hasta que se reciba una respuesta.
* Si no se recibe respuesta tras varios intentos, se asume que el servidor ha fallado.

**Filtrado de duplicados:**

* Al usar retransmisiones, es posible que el servidor reciba múltiples copias de la misma petición.
* Este mecanismo descarta las solicitudes duplicadas para evitar que el servidor procese la misma operación varias veces.

**Retransmisión de resultados:**

* El servidor guarda un historial de los resultados de las operaciones realizadas.
* Si el cliente no recibe el resultado, el servidor puede retransmitirlo desde el historial sin necesidad de volver a ejecutar la operación.
* Estos mecanismos permiten mejorar la confiabilidad y consistencia en aplicaciones distribuidas, manejando posibles fallos de comunicación o errores en el servidor.

La semántica de invocación "pudiera ser" se refiere a un escenario en sistemas distribuidos donde el cliente no puede determinar si una operación en el servidor se ha ejecutado una vez o no se ha ejecutado en absoluto. Esto ocurre cuando no se implementan medidas de tolerancia a fallos.

**Problemas asociados:**

1. **Fallos de omisión**:
   * Ocurren si se pierde el mensaje de solicitud o el mensaje de respuesta, evitando que el cliente reciba el resultado.
2. **Fallos por caída**:
   * Suceden si el servidor que aloja el objeto remoto falla antes de completar la ejecución del método.
3. **Incógnita**:
   * Si el cliente no recibe respuesta tras un tiempo de espera (timeout), no puede saber si el método no se ejecutó o si el mensaje de respuesta se perdió.
   * Esto deja al cliente en incertidumbre sobre el estado de la operación, ya que podría haber fallado la invocación o simplemente haberse perdido el resultado.

La semántica de invocación "al menos una vez" asegura que, si el cliente recibe un resultado, el método en el servidor se ha ejecutado al menos una vez. Si no se obtiene respuesta, se lanza una excepción indicando que no se recibió el resultado.

**Características y Mecanismos:**

* **Retransmisión de mensajes de petición**:
  + Se utiliza para evitar fallos por omisión, reenviando la solicitud hasta recibir una respuesta, garantizando así que el método se intente ejecutar al menos una vez.

**Problemas asociados:**

1. **Fallos por caída**:
   * Ocurren si el servidor falla antes o durante la ejecución del método, impidiendo que se complete la operación.
2. **Fallos arbitrarios**:
   * Pueden surgir si el mensaje de solicitud se retransmite varias veces, lo que podría hacer que el servidor ejecute el método múltiples veces.
   * Esto puede causar resultados incorrectos o inconsistentes si la operación no es idempotente (es decir, si repetirla cambia el estado del sistema).

**Semántica como máximo una vez:** el invocante recibe bien un resultado, en cuyo caso el invocante sabe que el método se ejecutó exactamente una vez, o una excepción que le informa de que no se recibió el resultado, de modo que el método se habrá ejecutado o una vez o ninguna en absoluto.

La semántica de invocación como máximo una vez puede obtenerse utilizando todas las medidas de tolerancia frente a fallos.

Tanto Java RMI como CORBA observan la semántica de invocación como máximo una vez, pero CORBA permite emplear la semántica pudiera ser para los métodos que no devuelven resultados.

Sun RPC proporciona la semántica de llamadas al menos una vez.

**El modelo de objetos distribuido**

RMI (Remote Method Invocation) es una extensión de RPC (Remote Procedure Call) al mundo de los objetos distribuidos. Permite que un objeto en un proceso pueda invocar métodos en otro objeto que podría estar en un proceso remoto, de forma similar a como se haría con una llamada local.

**Características:**

* **Invocaciones de métodos remotas:**
  + Un objeto puede ejecutar métodos en otro objeto que se encuentra en un proceso distinto, potencialmente en una máquina diferente.
  + Los detalles técnicos de la comunicación subyacente (como la serialización de datos y la transmisión de mensajes) están ocultos para el usuario, facilitando la programación.
* **Invocaciones de métodos locales:**
  + También permite que los objetos ejecuten métodos en otros objetos dentro del mismo proceso, como lo harían en un entorno de programación tradicional.

Diagrama

Descripción generada automáticamente

**Objetos Remotos:** Objetos que pueden recibir invocaciones remotas.

B y F son objetos remotos. Todos los objetos pueden recibir invocaciones locales, a pesar de que sólo puedan recibirlas de otros objetos que posean referencias ellos.

Por ejemplo, el objeto C puede tener una referencia al objeto E de modo que puede invocar uno de sus métodos.

**Los 2 conceptos fundamentales siguientes son el corazón del modelo objetos distribuidos:**

* **Referencia de objeto remoto:** otros objetos pueden invocar los métodos de un objeto remoto si tienen acceso a su “referencia de objeto remoto”
* **lnterfaz remota:** c/ objeto remoto tiene una interfaz remota que especifica cuáles de sus métodos pueden invocarse remotamente.

Referencias a objetos remotos. permite que cualquier objeto que pueda recibir un RMI tenga una referencia a objeto remoto.

Una referencia a objeto remoto “es un identificador que puede usarse a lo largo de todo un sistema distribuido para referirse a un objeto remoto particular único.”

Las referencias a objetos remotos son análogas a las locales en cuanto a que:

* El objeto remoto donde se recibe la invocación de método remoto se especifica mediante una referencia a objeto remoto.
* Las referencias a objetos remotos pueden pasarse como argumentos y resultados de las invocaciones de métodos remotos.

En **RMI (Remote Method Invocation)**, una **interfaz remota** define los métodos que pueden ser invocados por objetos en otros procesos.

**Características:**

* La clase del objeto remoto implementa los métodos de su interfaz remota, permitiendo que estos sean accesibles para clientes remotos.
* Los objetos remotos solo permiten la invocación de métodos definidos en su interfaz remota, mientras que los objetos locales pueden acceder tanto a estos métodos como a otros métodos adicionales del objeto.

**Ejemplos:**

* En Java RMI, las interfaces remotas se definen como cualquier interfaz Java, pero deben extender la interfaz Remote para ser consideradas remotas.
* En CORBA, se utiliza IDL (Interface Definition Language) para definir interfaces remotas.

Esto facilita la comunicación entre objetos distribuidos, permitiendo llamadas a métodos remotos de manera segura y estructurada.

**Acciones en un sistema de objetos distribuido.** Como en el caso no distribuido, una acción se inicia mediante la invocación de un método, que pudiera resultar en consiguientes invocaciones sobre métodos de otros objetos.

**Excepciones.** Cualquier invocación remota puede fallar por razones relativas a que el objeto invocado está en un proceso o computador diferente de la del objeto que lo invoca. Por ejemplo, el proceso que contiene el objeto remoto pudiera malograrse o estar demasiado ocupado para responder, o pudiera perderse el mensaje resultante de la invocación.

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

**Implementación de RMI**

El módulo de comunicación en RMI es responsable de gestionar el protocolo de petición-respuesta entre el cliente y el servidor, transmitiendo mensajes de solicitud y respuesta.

**Características:**

* Utiliza tres elementos clave del mensaje:
  1. Tipo de mensaje
  2. idPeticion (identificador de la petición)
  3. Referencia remota del objeto a invocar.
* El idMetodo y el proceso de empaquetado/desempaquetado de datos son gestionados por el software RMI.
* Este módulo colabora para garantizar una semántica de invocación segura (por ejemplo, como máximo una vez).
* En el servidor, el módulo de comunicación identifica el distribuidor adecuado para el objeto a través de su referencia local, obtenida mediante el módulo de referencia remota.

El módulo de referencia remota en RMI se encarga de gestionar las referencias entre objetos locales y remotos.

**Funciones:**

* Traducción de referencias entre objetos locales y remotos.
* Creación de referencias para objetos remotos.
* Gestión de una tabla de objetos remotos en cada proceso, que mapea referencias locales a referencias remotas accesibles en todo el sistema.

**Tabla de objetos remotos:**

* Contiene una entrada para cada objeto remoto implementado por el proceso (ej., el objeto remoto B registrado en el servidor). ver figura 5.6 arriba
* Incluye una entrada para cada proxy local que actúa como representante de un objeto remoto (ej., el proxy para B registrado en el cliente). ver figura 5.6 arriba

El módulo de referencia remota en RMI gestiona la creación y resolución de referencias a objetos remotos durante la comunicación.

**Acciones principales:**

1. **Creación de referencias:**
   * Cuando un objeto remoto se utiliza por primera vez como argumento o resultado, el módulo crea una referencia y la agrega a su tabla de objetos remotos.
2. **Resolución de referencias:**
   * Al recibir una referencia a un objeto remoto en un mensaje de petición o respuesta, el módulo busca la referencia correspondiente en su tabla.
   * Si el objeto remoto no está en la tabla, se crea un nuevo proxy y se añade a la tabla.
3. **Empaquetado y desempaquetado:**
   * Durante el empaquetado/desempaquetado de mensajes, el módulo utiliza su tabla para identificar el objeto local (proxy o remoto) que debe invocarse según la referencia recibida.

**EI software de RMI consiste en una capa de software entre:**

* los objetos del nivel de aplicación y
* los módulos de comunicación y de referencia remota.

Los papeles de los objetos de middleware mostrados en la Figura 5.6 son como

sigue:

**Proxy:** hace que la invocación al método remoto sea transparente para los clientes.

Para ello se comporta como un objeto local para el que invoca; pero en lugar de

ejecutar la invocación, dirige el mensaje al objeto remoto.

Hay un proxy para cada objeto remoto del que el cliente disponga de una referencia

de objeto remoto. La clase de un proxy implementa los métodos de la interfaz

remota del objeto remoto al que representa. Esto asegura que las invocaciones al

método remoto son adecuadas según el tipo del objeto remoto.

Cada Servidor tiene un Distribuidor y un Esqueleto para cada clase que

represente a un objeto remoto.

**Distribuidor:** El distribuidor recibe el mensaje de petición desde el módulo de

comunicación. Emplea el idMetodo para seleccionar el método apropiado del

esqueleto, pasándole el mensaje de petición. El distribuidor y el proxy

emplean los mismos métodos de asignación de cada idMetodo para los

métodos de la interfaz remota.

**Esqueleto:** la clase de un objeto remoto tiene un esqueleto, que implementa

los métodos interfaz remota. Se encuentran implementados de forma muy

diferente de los métodos del objeto remoto. Un método del esqueleto

desempaqueta los argumentos del mensaje de petición, invoca el método

correspondiente en el objeto remoto.

**Referencias a objetos remotos.**

Cuando un cliente invoca un método en un objeto remoto en RMI, se envía un mensaje de invocación al proceso servidor que aloja dicho objeto.

**Detalles:**

* El mensaje debe identificar específicamente qué objeto remoto se va a invocar.
* Se incluye una referencia a objeto remoto, que es un identificador único válido en todo el sistema distribuido, permitiendo localizar el objeto exacto en el servidor para ejecutar el método solicitado.

**Apache Kafka**

**Escala de tiempo

Descripción generada automáticamente**