RMI multicast asincrónico

Gaston Gustavo Rios - okason1997@hotmail.com

Introducción

Java Remote Method Invocation (RMI) es una API de Java que permite la invocación de métodos en objetos remotos, esto permite la creación de sistemas distribuidos donde la comunicación entre nodos se realiza mediante invocación a métodos de objetos que pueden estar en otros nodos. RMI se utiliza de forma similar pero orientado a objetos a remote procedure calls (RPC), donde se realizan llamadas a procedimientos de otros nodos del sistema distribuido.
RMI es muy útil en muchas ocasiones, pero cuenta con algunas limitaciones.

- La obtención del objeto remoto debe ser de algún sistema remoto conocido. Debido a esto, si este sistema remoto dejará de funcionar o se saturara sería necesario poseer algún otro sistema remoto conocido del cual importar el objeto remoto. Esto genera un problema de disponibilidad si el sistema remoto conocido no está funcionando correctamente necesitando sistemas conocidos de apoyo los cuales son difíciles de incluir en el programa.
- Otra limitación que posee RMI es que la conexión entre objeto local y remoto es siempre de uno a uno. Esto es debido a que en RMI las conexiones son implícitamente unicast por TCP haciendo que las conexiones solo puedan ser entre 2 computadoras a la vez. Si se desea hacer llegar un mismo mensaje a objetos localizados en n computadoras sería necesario realizar n invocaciones de métodos remotos. También sería necesario conocer la dirección ip y el puerto en el cual fueron expuestos los objetos remotos de cada computadora.

Estas limitaciones previenen el uso de RMI en varios problemas donde podría ser muy útil contar con un RMI que alcance fácilmente múltiples computadoras, por ejemplo en el caso de querer obtener el estado de cada objeto remoto de un sistema distribuido.
Para solucionar esto se propuso e implementó RMI multicast asincrónico (RMI-MA), el cual, sin agregar muchas más complicaciones que RMI permite la obtención de un objeto remoto mediante multicast y la invocación de métodos remotos de forma asincrónica mediante multicast a múltiples objetos localizados en diferentes computadora para posteriormente obtener los resultados de cada objeto remoto.
RMI-MA permitirá la utilización de RMI en múltiples nuevos problemas. Un ejemplo es la recolección de información de múltiples objetos de un sistema distribuido en una sola operación. Otra campo en el que utilizar RMI-MA sería ventajoso es en las aplicaciones colaborativas, donde múltiples nodos de un sistema distribuido trabajan con un objetivo común RMI-MA permitiría un llamado eficiente a métodos de los objetos de los nodos.
También es útil en casos donde no se conoce el tamaño del problema, ya que es posible agregar nuevas computadoras fácilmente al grupo multicast. En casos donde los argumentos puedan tomar tamaños excesivamente grandes RMI-MA otorga una forma eficiente de enviar estos argumentos a los distintos objetos remotos.

Remote Method Invocation

La creación de sistemas distribuidos en Java es simplificada en gran medida gracias a la existencia de RMI. RMI permite a los programadores crear sistemas distribuidos de aplicaciones con tecnología Java a aplicaciones con tecnología Java de forma tal que los métodos de objetos remotos puedan ser invocados por otras máquinas virtuales Java (JVM) las cuales podrían estar en diferentes hosts. Así es como RMI permite la comunicación entre programas escritos en Java de forma sencilla al ser una librería nativa de Java que se mantiene en el paradigma orientado a objetos propio de Java. [1]
Es posible ver desde un nivel básico a RMI como un mecanismo de RPC. Pero RMI posee numerosas ventajas sobre RPC al ser orientado a objetos. Los sistemas de RPC tradicionales son de lenguaje neutral, lo que evita que este posea funcionalidad no disponible en todas las posibles plataformas. RMI en cambio posee todas las ventajas del lenguaje Java. Es orientado a objetos pudiendo pasar objetos como argumentos y valores de retorno sin realizar truncamiento de clase, por lo que el comportamiento de los objetos enviados no cambiará. La utilización de patrones de diseño de la programación orientada a objetos se mantiene, ya que se sigue trabajando con objetos.[2]
Usualmente en RMI se cuenta con dos roles en la comunicación: un cliente y un servidor. El servidor exporta el objeto remoto escuchando llamadas a métodos remotos de los clientes para ejecutar estas llamadas localmente y luego devolver mediante TCP el valor de retorno el cual puede ser un objeto. Adicionalmente, el servidor podrá exportar el stub del objeto remoto para que los clientes lo descarguen.
El cliente, por otro lado, utilizara este stub descargado como un proxy para realizar las llamadas a los métodos de los objetos remotos como si este objeto fuera un objeto local. RMI provee los mecanismos para realizar este intercambio entre el cliente y el servidor otorgando funcionalidades adicionales como un garbage collector distribuido que elimina los objetos remotos no referenciados por clientes de la memoria del servidor.

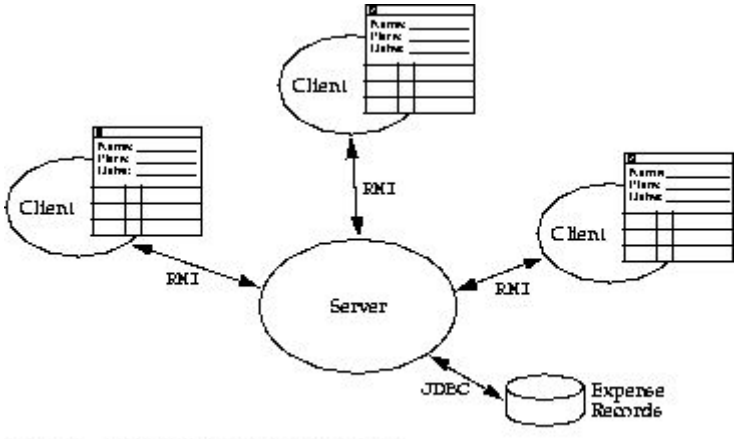
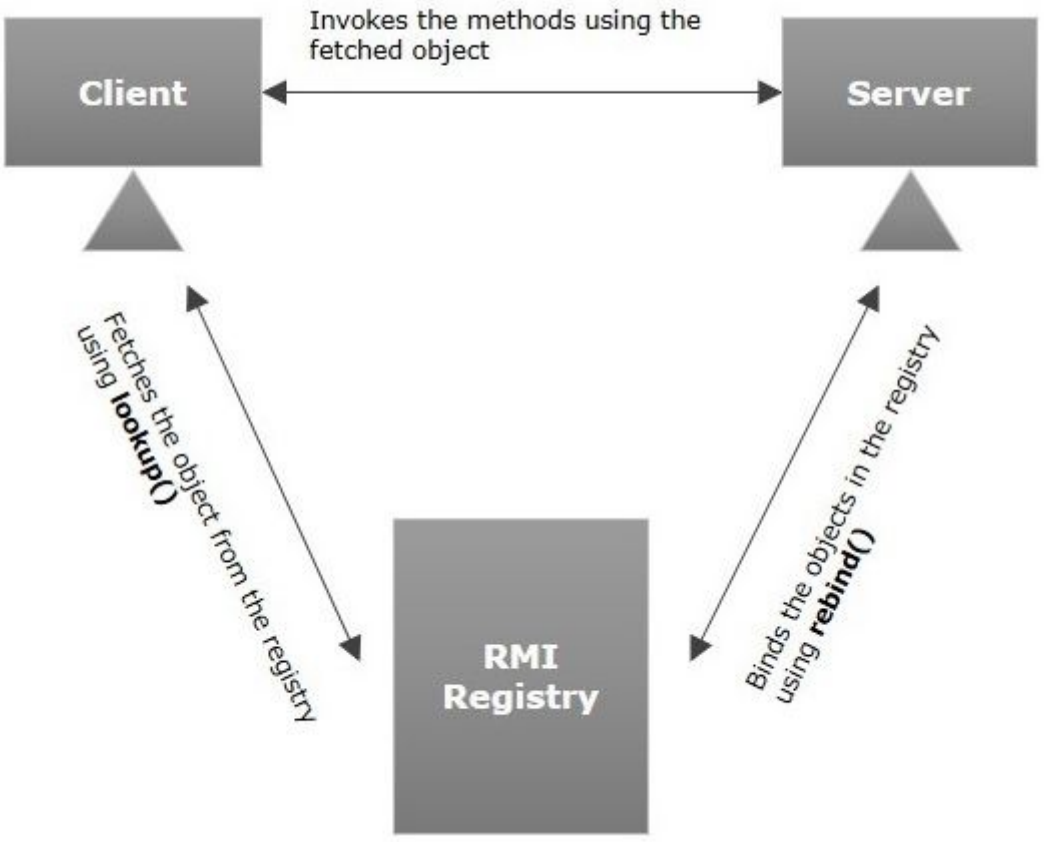


FIGURE 1 An Expense Reporting Architecture

La funcionalidad de RMI puede ser dividida en 2 grandes tareas que incluyen la interacción entre el servidor y el cliente.

Localización del objeto remoto

Para obtener la referencia a un objeto remoto, las aplicaciones pueden registrar sus objetos remotos mediante la funcionalidad de RMI Registry, en donde las referencias a objetos remotos serán registrados con un nombre para poder ser localizados por clientes fácilmente. El cliente solo necesitará localizar el servidor de registro de nombres y buscar el objeto por el nombre con el que fue registrado, así obtendrá su stub el cual incluirá la referencia con la dirección en la que el objeto remoto se encuentra. [4]



Por otro lado, las referencias a objetos remotos pueden ser pasadas por las aplicaciones como partes de su funcionamiento normal. Solo es necesario poseer el stub generado en el servidor para poder acceder al objeto remoto que este posee.
En cada ciclo de vida de un objeto remoto en el cliente solo es necesario acceder a su referencia una vez, esta será reutilizada cada vez que se invoque un método en dicho objeto remoto.

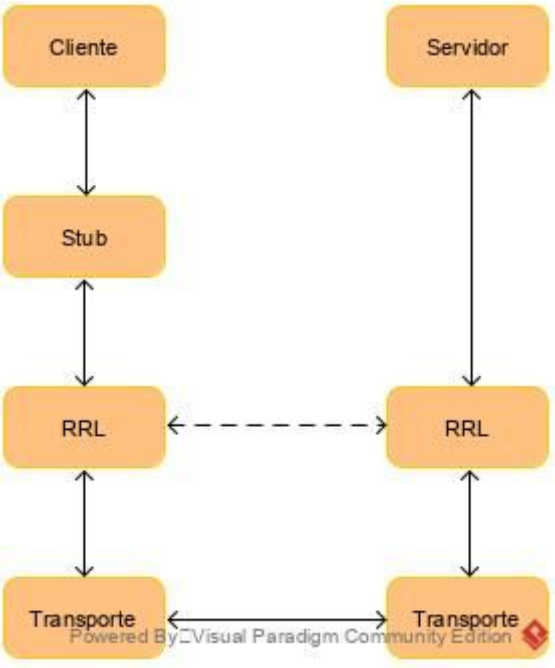
Comunicación con los objetos remotos

La comunicación con objetos remotos localizados en los servidores se realiza de forma unicast mediante TCP. El cliente posee las direcciones ip y los puertos de los objetos remotos en sus referencias en sus stubs.
De esta forma, cada vez que se realiza la invocación de un mensaje a un objeto remoto, el stub actuando como proxy abrirá una conexión TCP con el servidor, enviará el método a ejecutar y mediante una operación de marshall enviará los argumentos necesarios para la

ejecución del método. La operación de marshall transforma la representación de un objeto en memoria a un formato de datos apto para su transmisión.
Por el lado del servidor, este recibirá por TCP un stream el cual idealmente poseerá toda la información necesaria para la invocación del método. Primero, obtendrá la operación a realizar por parte del cliente, la cual será la ejecución de un método. Luego obtendrá el hash del método en cuestión, con el cual ubicará el método a ejecutar en el objeto remoto y realizará el unmarshal de los argumentos para así poder invocar dicho método. Finalmente, con el resultado o el Throwable (excepción o error) obtenido como resultado de la ejecución del método se realizará un marshal de este para su envío hacia el cliente.
El thread del cliente que haya realizado la invocación al método remoto se bloquea hasta obtener la respuesta del servidor con el resultado de esta invocación. Una vez obtenido el mensaje TCP del servidor con el resultado realizará unmarshal de este para obtener finalmente el objeto resultante de la operación. Este objeto puede ser el resultado esperado o un Throwable , de esta forma el cliente sabrá si en el objeto remoto se obtuvo una excepción o un error como resultado de la invocación al método. Por último se retorna a la ejecución normal del cliente habiendo obtenido el resultado del objeto remoto como si este fuera un objeto local.
Esta operación de comunicación podrá ser realizada cuantas veces se desee siempre y cuando se cuente con la referencia al objeto remoto.

Funcionamiento de RMI

La implementación de RMI puede ser separada en 3 capas de abstracción. La capa de stub, la capa de referencia remota y finalmente la capa de transporte.

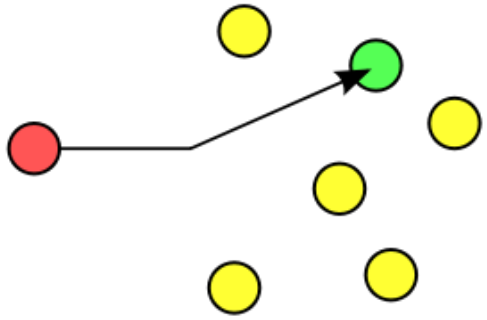


El stub es una representación del objeto remoto en el cliente. Funciona como un proxy, cuando un método es llamado en el stub este se encarga de redireccionar esta llamada para que llegue al objeto remoto y retornar el valor obtenido del objeto remoto al cliente. El stub posee la referencia remota y su trabajo será llamar invoke() a este objeto delegando el trabajo de realizar la conexión con el servidor y obtener los resultados. Antiguamente existía

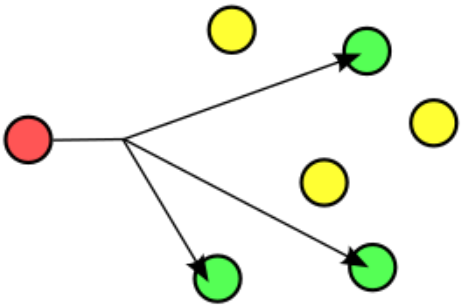
un objeto skeleton del lado del servidor que funcionaba como proxy al objeto remoto del servidor, pero en Java 2 SDK se eliminó la necesidad de utilizarlo al introducir un nuevo protocolo de stub.
La capa de referencia remota es la capa encargada de administrar e interpretar las referencias del cliente al objeto remoto. Del lado del cliente, esta capa posee un objeto UnicastRef que implementa la interfaz RemoteRef, el cual posee el método invoke() que será llamado por el stub cuando un método sea invocado. El objeto UnicastRef pasara el llamado a la referencia remota del servidor mediante la capa de transporte. En el servidor, el objeto UnicastServerRef que implementa las interfaces ServerRef y Dispatcher compone la capa de referencia remota. Este objeto, por parte de ServerRef, es encargado de exportar el stub creando la referencia remota que utilizará el cliente y indicando a la capa de transporte que escuche a llamadas de métodos. También, por parte de Dispatcher, es encargado de implementar dispatch() donde realizará la llamada al método indicado por el cliente en el objeto local del servidor y retorna el valor obtenido a UnicastRef en el cliente mediante la capa de transporte.
La capa de transporte posee toda la información necesaria para realizar las conexiones entre el cliente y el servidor mediante TCP/IP. Esta capa implementa todos los protocolos necesarios para realizar la conexión y también posee estrategias de penetración de firewall.

Multicast

Las aplicaciones colaboradoras o que manejan información de media requieren una forma eficiente de realizar comunicación de grupos (comunicación de uno a muchos o muchos a muchos). Una solucion que puede surgir en principio es la utilizacion de multiples conexiones unicast donde se envían duplicados de los datos a cada objetivo. El problema de esta solución es que no posee buena escalabilidad al aumentar en gran medida la cantidad de miembros del grupo o la cantidad de datos a ser enviados.
Multicast es una forma más eficiente de solucionar este problema, otorgando una manera de enviar en paquetes a grupos. IP-multicast es un método sumamente eficiente de comunicación de grupos, requiriendo que el origen solo envie un paquete el cual será replicado y distribuido utilizando protocolos de ruteos multicast. Los routers multiplicaran y redireccionar los paquetes a cada rama del árbol de distribución multicast llamado Spaning Tree (ST). [5]
El origen no necesita conocer todos los destinos a los que llegará el paquete, solo necesita conocer la dirección IP que se encuentre dentro del rango de direcciones IP reservadas para multicast. [7] El origen y los receptores deberán conocer esta dirección IP multicast y asociarse a esta informando a la red que están interesados en recibir paquetes destinados a los miembros del grupo asociado a la dirección IP dada. De esta forma, si un paquete es enviado a un grupo asociado a una dirección IP, todos los que se hayan asociado a tal dirección recibirán el paquete. [6]



Unicast



Multicast

Funcionamiento de RMI multicast asincrónico

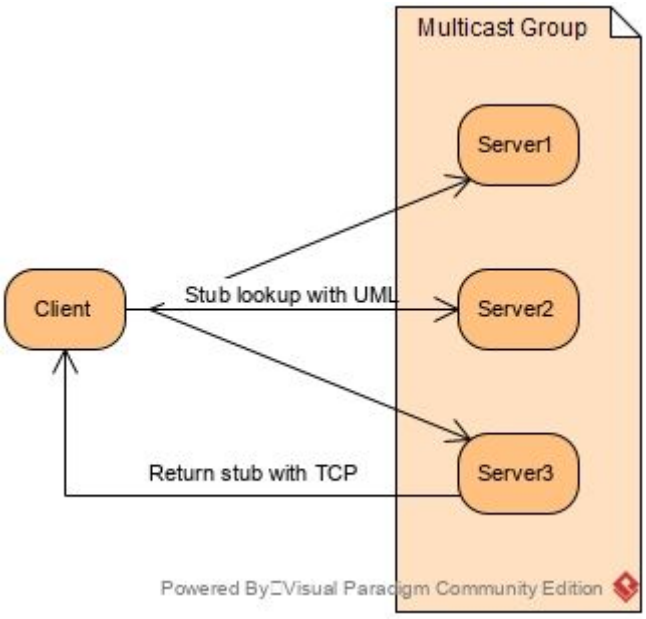
En RMI-MA un cliente podrá enviar un mensaje en una sola operación a múltiples objetos remotos, los cuales podrían estar localizados en diferentes hosts del sistema distribuido. El cliente no necesita conocer a estos hosts de antemano ya que al ser la comunicación por medio de multicast, los mensajes serán redireccionados a todos los hosts que se hayan unido al grupo multicast con la misma IP. Posteriormente el cliente obtendrá la respuesta del mensaje de cada objeto remoto y almacenará tal respuesta en un arreglo a medida que estas lleguen. De esta forma el cliente puede seguir realizando operaciones mientras espera que lleguen las respuestas a la invocación del método.
La utilización de RMI-MA no requiere grandes cambios por parte del programador con respecto a RMI.
En RMI-MA, la interfaz que implementa el objeto remoto y que usará el cliente para acceder a este objeto debe devolver un valor de clase Object, de esta forma es posible castear este valor en el cliente a clase List para obtener la lista que almacena los valores de retorno y en el servidor no será necesario realizar cambios a la implementación del objeto remoto ya que todas las clases en Java heredan de Object el valor de retorno puede poseer cualquier clase en el objeto remoto.
Para exportar el objeto remoto y escuchar mensajes enviados por clientes referidos a tal objeto, MulticastRemoteObject implementa el método exportObject() que toma como argumentos el objeto remoto y el puerto en el que escuchara por mensajes enviados a través de multicast. También es posible volver al objeto remoto una subclase de MulticastRemoteObject, entonces este objeto se exportará inmediatamente al ser instanciado.
El sistema de registro de nombres para la obtención del stub por parte del cliente fue reemplazado por una nueva clase StubExporter la cual permitirá al servidor exportar fácilmente los stubs. StubExporter implementa el método export(), el cual exportará un stub a una direccion IP multicast asignandole un nombre. Los clientes que deseen acceder al objeto remoto deberán utilizar el método lookup de StubExporter con el nombre asignado al objeto que desean acceder. Una vez un cliente quiera acceder a un objeto remoto en una dirección IP multicast, si un servidor está exportando un objeto con el nombre indicado en esa dirección, se abrirá una conexión TCP entre el cliente y el primer servidor que responda para enviar el stub al cliente por unicast.

Una vez obtenido el objeto remoto, el cliente puede llamar a todos los métodos que implemente la interfaz del objeto remoto. De estos métodos obtendrá como valor de retorno un objeto el cual deberá ser casteado a List<Object>. A esta lista se irán agregando los resultados obtenidos por cada objeto remoto que haya recibido el paquete con la información de la invocación del método por multicast. Estos resultados podrán ser los retornos esperados o throwables en el caso de que la invocación del método en el objeto remoto haya generado una, por lo tanto el cliente deberá comprobar si algún objeto retornado es una excepción o error antes de intentar usar el valor esperado.
Debido a la naturaleza asincrónica de RMI-MA, si se intentara utilizar la lista inmediatamente a haber invocado el método, se encontraría con que esta está vacía. Por lo tanto existen 2 formas de utilizar los resultados: esperar a que la lista se llene con las respuestas de los objetos remotos, o asignar un lambda o referencia a método que se ejecute cada vez que un valor de retorno sea agregado a la lista.
Al igual que en RMI, en RMI-MA su funcionamiento puede ser clasificado en 2 grandes tareas: la localización del objeto remoto y la comunicación con los objetos remotos.

Localización del objeto remoto

Para la localización del objeto remoto se realiza un intercambio de mensajes inspirado en la implementación de Vinay Ujjinihavildar [3]. Se realiza una comunicación entre el cliente y el servidor en la que el cliente busca por multicast mediante UDP un servidor el cual exponga un objeto remoto con el nombre indicado. Una vez un servidor que haya expuesto un objeto remoto bajo tal nombre reciba el pedido del stub por parte del cliente, el servidor abrirá una coneccion unicast por TCP/IP con el cliente para enviar el stub.
El cliente comenzará abriendo un puerto para aceptar conexiones TCP. Esta conexión será por la cual el servidor le enviará el stub al cliente. Posteriormente preparará el datagrama que enviara por medio de multicast. Este datagrama poseerá el número de operación, por el cual el servidor podrá saber que la operación recibida se refiere a una operación de búsqueda de objeto remoto, también tendrá el nombre asociado del objeto remoto que busca el cliente y el puerto donde el cliente espera recibir por medio de una conexión unicast TCP el stub correspondiente al nombre asociado al objeto remoto. El datagrama será construido como un arreglo de bytes utilizando ObjectOutputStream, de esta forma se facilita su creación y la posterior obtención de los componentes del datagrama. El cliente solo recibirá el stub de un servidor, a pesar de que el requerimiento del objeto remoto llegue a múltiples servidores solo se abrirá una conexión TCP, de esta forma el stub llegara del primer servidor que abra la conexión con el cliente. Previamente a recibir el stub, el cliente comprobará que la operación indicada por el servidor sea la correcta.
El servidor construirá un stub para el cliente mediante el método exportObject() de MulticastRemoteObject. exportObject() creará un objeto que implementara el patrón proxy y en cada invocación a un método invocará este método en su referencia asociada por medio de invoke(). Esta referencia será asignada al proxy por MulticastRemoteObject y es un objeto MulticastRef. Una vez creado el stub, el servidor expondrá el objeto remoto utilizando el método export() de StubExporter. Mediante este método, el servidor se unirá a un grupo multicast de una IP determinada donde escuchara por requerimientos de objetos remotos de clientes. Al recibir un datagrama este es convertido a un ObjectInputStream. Con esta entrada, el servidor comprobará la operación del datagrama para descubrir si su objetivo es

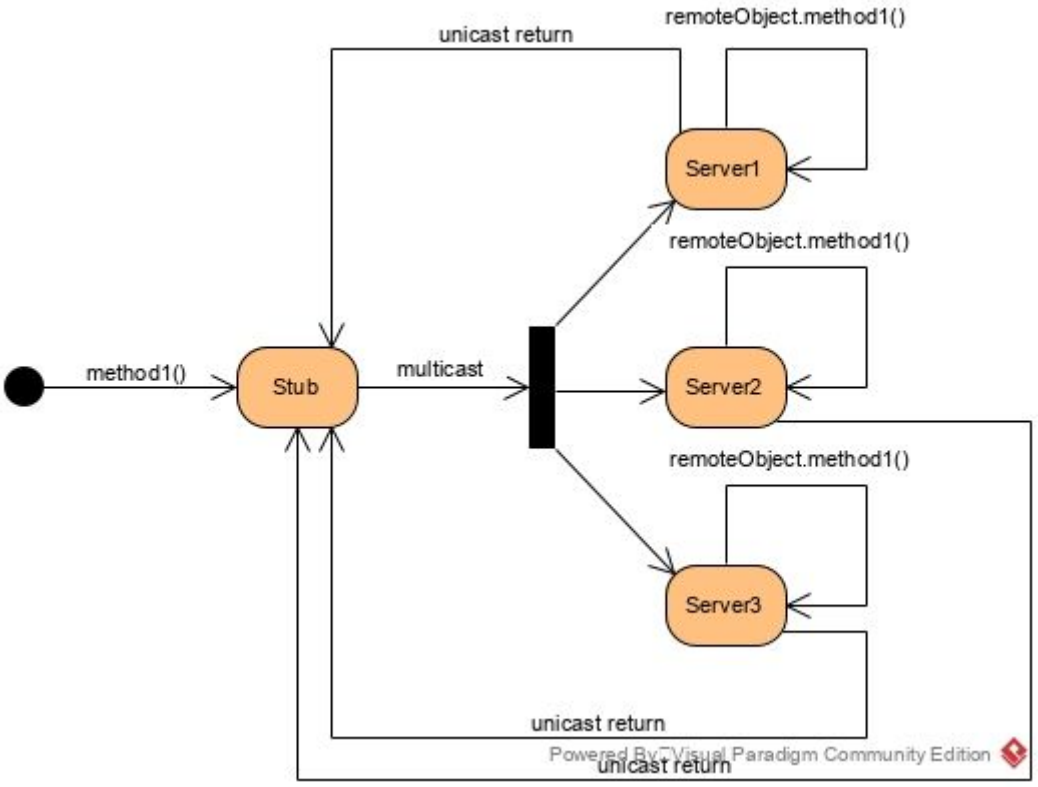
la obtención del stub por parte del cliente. Si la operación es correcta, comprobará que el nombre indicado en el datagrama coincida con el nombre asignado al objeto remoto. En caso de que los nombres coincidan se procederá a intentar crear una conexión TCP con el cliente al puerto indicado por el cliente para recibir la respuesta, el cual fue enviado en el datagrama, y la ip asignada al datagrama recibido por el servidor. Una vez conectado con el cliente, el servidor procede a enviar el identificador de operación con la cual el cliente y finalmente el stub. Este proceso funciona en su propio thread, el cual loopea esperando clientes, por lo que cualquier cantidad de clientes podrán acceder al objeto remoto.



Comunicación con los objetos remotos

Una vez obtenido el stub con la referencia asociada por el servidor MulticastRef, es posible utilizarlo llamando a los métodos indicados en la interfaz utilizada. El stub funciona aplicando el patrón proxy. El patrón proxy permite crear un substituto o placeholder para otro objeto, de forma tal que este objeto proxy controle el acceso al objeto original permitiendo realizar operaciones antes o después de que los llamados lleguen al objeto original. Cuando se realiza la invocación de un mensaje en el stub este realizara la operación invoke() en su referencia MulticastRef. En esta operación, se realizará toda la lógica que controla que el mensaje llegue al servidor y que el resultado de cada servidor llegue a la lista de resultados.
invoke() comienza abriendo una conexión con un grupo multicast de una IP multicast predefinida y abrirá un puerto donde esperará las respuestas de los servidores por TCP. Luego procederá a enviar a este grupo toda la información necesaria para que el mensaje sea ejecutado en los servidores del grupo que hayan expuesto el objeto remoto. Esta información está compuesta por el número de operación, el puerto donde el cliente esperará las respuestas, el identificador de la invocación, el hash del método a ejecutar y los parámetros que este método requiera. Con el número de operación el servidor podrá saber, una vez recibido el datagrama, que la operación se refiere a una invocación a un método. El puerto lo usara el servidor para saber donde debe retornar los valores de retornos de la invocación. El identificador de invocacion sera utilizado por el cliente para comprobar que las respuestas que reciba de los servidores vayan referidas a la invocación correcta. El hash

del método le indicará al servidor cual es el método a ejecutar. MulticastRef realizará un marshal de los datos para transformarlos en un arreglo de bytes y así poder enviar el datagrama correctamente.
Una vez formado el datagrama, previo a su envío para evitar perder respuestas, el cliente comenzará a escuchar de forma asincrónica las respuestas de los servidores por un periodo de tiempo finito. El cliente aceptará las conexiones TCP entrantes al puerto indicado y comprobará que el identificador de invocación enviado por el cliente coincida con la invocación realizada. En caso de no coincidir, se procederá a cerrar la conexión. Si el identificador de invocación fuera el correcto, se procederá a leer el byte success. success indica si la operación en el objeto remoto ubicado en el servidor fue exitosa o generó un Throwable. Utilizando success se casteará el valor de retorno a Object en caso de haber obtenido una ejecución exitosa o Throwable en caso de que la ejecución en el objeto remoto haya generado una excepción o error, por lo que su valor de retorno será dicha excepción o error. En caso de haberle asignado una expresión lambda o una referencia a un método a MulticastRef, esta operación asignada será ejecutada utilizando como entrada el resultado. Finalmente el resultado será agregado a la lista de resultados utilizando un indicador de exclusión mutua para que la escritura sea segura.
Al ejecutar exportObject() de MulticastServerRef el servidor comenzará a escuchar en el grupo multicast predeterminado por invocaciones a métodos del objeto remoto utilizando al objeto MulticastServerRef como dispatcher para esas invocaciones. Cada vez que reciba por UDP de este grupo multicast un número de operación que indique invocación a método remoto se ejecutará el método dispatch() de MulticastServerRef. dispatch() se encarga de realizar el unmarshal de la información recibida por UDP, realizar la invocación del método al objeto remoto y devolver el resultado al cliente por TCP. Para esto comienza obteniendo del ObjectInput, generado del datagrama obtenido por UDP, el hash del método a ejecutar. MulticastServerRef, en la ejecución de exportObject() almaceno los métodos del objeto remoto en un objeto Map donde la clave para cada método será su hash, de esta forma es posible acceder al método indicado si se posee el hash correspondiente. Sabiendo el método se realiza unmarshal de los argumentos de entrada para este y utilizándolos se realiza la invocación del método. Con la invocación del método se obtiene el resultado que será enviado al cliente. Si se generó alguna excepción o error en la ejecución de dispatch(), este throwable será el resultado enviado al cliente. El servidor procederá a abrir una conexión unicast TCP con el cliente al puerto y dirección que el cliente envio en el datagrama conteniendo la información de la invocación del método remoto. A través de esta conexión enviará al cliente el identificador de la invocación que recibió junto a los datos necesarios para la invocacion del metodo, el indicador de exito y finalmente el resultado.



Referencias

[1] Remote Method Invocation Home - https://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html

[2] Java Remote Method Invocation - Distributed Computing for Java - https://www.oracle.com/technetwork/java/javase/tech/index-jsp-138781.html

[3] Vinay Ujjinihavildar, B.E.- UNIVERSITY OF NORTH TEXAS - The feasibility of multicasting in RMI. - 2003 - https://www.semanticscholar.org/paper/The-Feasibility-of-Multicasting-in-RMI-Ujjinihavildar/27a59cea2475ca3970184f59b97c9656d54b01b9

[4] David Reilly - Introduction to Java RMI - 2006 - http://www.javacoffeebreak.com/articles/javarmi/javarmi.html

[5] Dasgupta, Subhasish - Encyclopedia of Virtual Communities and Technologies - 2005

[6] S. E. Deering, Stanford University - RFC 988 - 1986

[7] M. Cotton, L. Vegoda, ICANN, D. Meyer - RFC 5771- 2010