RMI: Uma Visão Conceitual

Márcio Castro, Mateus Raeder e Thiago Nunes

11 de abril de 2007

Resumo

Invocação de Método Remoto (*Remote Method Invocation* - RMI) trata-se de uma abordagem Java para disponibilizar as funcionalidades de uma plataforma de objetos distribuídos, permitindo que um objeto que está rodando em uma Máquina Virtual Java invoque, remotamente, métodos de objetos que estão sendo executados em outra Máquina Virtual. Uma das vantagens oferecidas por esta abordagem é o fato de tornar transparente para o programador a comunicação das partes do sistema através de uma rede. Este artigo traz uma visão geral sobre os principais aspectos (do ponto de vista conceitual) desta tecnologia.

1 Introdução

Os primeiros computadores que foram produzidos, além de serem extremamente custosos, ocupavam prédios inteiros. Inicialmente eram somente utilizados em grandes laboratórios de universidades e indústrias. Sendo assim, para que o investimento com esta infraestrutura desse o retorno esperado, desejava-se que a máquina estivesse processando a maior parte do tempo possível. Estes computadores eram utilizados somente por operadores especializados e os *jobs* (trabalhos) eram processados seqüencialmente.

O problema do acesso restrito aos computadores, anteriormente citado, está vinculado a má utilização destes recursos: enquanto um *job* estava sendo desenvolvido, outros *jobs* não podiam ser processados. Além disso, a execução de *jobs* interativos, compostos de ações interdependentes, consumia muito tempo. A fim de superar estes problemas, surge então o conceito de tempo compartilhado (*time sharing*), compartilhando o uso do computador entre diferentes usuários, por determinada fatia de tempo [1].

O advento do conceito de tempo compartilhado foi o primeiro passo para a computação distribuída devido ao fato de prover dois importantes conceitos muito utilizados [1]: compartilhamento simultâneo de recursos por diversos usuários e o acesso a computadores localizados em lugares diferentes.

O uso de sistemas distribuídos tornou-se uma prática comum nas organizações que buscam compartilhamento de recursos, melhor comunicação e aumento de desempenho. Exemplos de sua utilização abrangem desde estações de tra-

balho em uma rede local até a Internet. Normalmente são caracterizados pela presença de atividades independentes em *software* e *hardware* heterogêneos [2].

2 Contexto

Por volta dos anos 80 desenvolveu-se uma tecnologia denominada Chamada de Procedimento Remoto (*Remote Procedure Call* - RPC), com o objetivo de permitir que as linguagens de programação procedurais (C, Pascal, entre outras) chamassem funções que residissem em outros computadores, tão convenientemente como se essa função fizesse parte do mesmo programa que executa no computador local. Esta tecnologia possibilitava aos programadores concentrarem-se apenas na programação, subdividindo o sistema em funções, não se preocupando com os mecanismos que permitem que as partes dos aplicativos se comuniquem. Pela facilidade apresentada pela tecnologia, a sua aplicação se difundiu na área de computação distribuída.

Resumidamente, o mecanismo de implementação RPC é composto por 5 elementos [1], como demonstrado na figura 1:

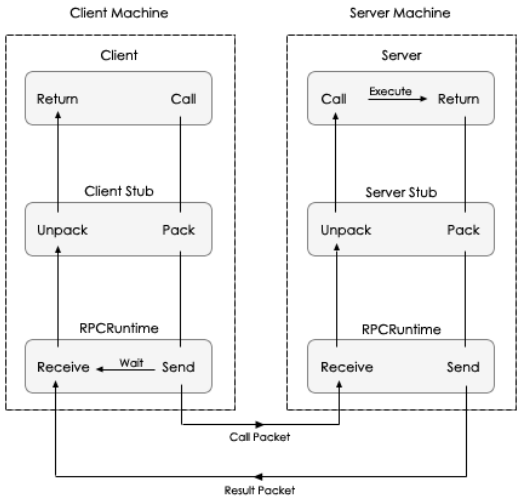

Figura 1: Mecanismo de implementação RPC.

- O cliente: é o processo que inicia a Chamada de Procedimento Remoto. É realizada uma chamada local que invoca o procedimento correspondente no seu *stub*.
- O *stub* do cliente: ao receber uma chamada do cliente, empacota a especificação do procedimento e os argumentos em uma mensagem para então

requisitar ao *RPCRuntime* local o envio ao *stub* do servidor. Ao receber o resultado da execução do procedimento, desempacota o resultado e o envia para o cliente.

- O *RPCRuntime*: lida com a transmissão de mensagens através da rede entre a máquina do cliente e do servidor. É responsável por retransmissões, confirmações, empacotamento, roteamento e criptografia. Cliente e servidor possuem um *RPCRuntime* local.
- O *stub* do servidor: ao receber um pedido do *RPCRuntime* local o *stub* do servidor, a desempacota e torna possível a invocação do procedimento adequado no servidor. Ao receber o resultado da execução do procedimento do servidor o *stub* do servidor empacota o resultado em uma mensagem e pede ao *RPCRuntime* local para enviá-la ao *stub* do cliente.
- O servidor: ao receber um pedido do seu *stub* o servidor executa o procedimento adequado e retorna o resultado da execução ao seu *stub*.

Esta tecnologia passou a apresentar dois problemas [3]. O primeiro está relacionado ao tipo de dados: com o advento das linguagens orientadas a objetos surgiu a necessidade de trabalhar com tipos de dados mais complexos (objetos, com atributos e métodos). Entretanto, o RPC foi projetado para trabalhar apenas com determinadas estruturas (compostas apenas por um conjunto de dados simples). O segundo problema está relacionado ao fato de que o programador deve compreender a Linguagem de Definição de Interface (*Interface Definition Language* - IDL), que é utilizada para descrever as possíveis funções a serem invocadas remotamente e gerar automaticamente os *stubs* do cliente e do servidor.

Neste contexto, surge a tecnologia de Invocação de Método Remoto (*Remote Method Invocation* - RMI), com a finalidade de resolver os problemas anteriormente citados. A RMI foi desenvolvida baseada nos conceitos de RPC aplicados especificamente à linguagem de programação Java.

3 Invocação de Método Remoto - RMI

RMI trata-se de uma solução Java para implementar comunicação entre aplicações distribuídas orientadas a objeto. A composição básica de uma aplicação que utiliza RMI é constituída por dois componentes principais: cliente e servidor. O servidor disponibiliza os serviços para serem acessados remotamente. O cliente, por sua vez, utiliza tais serviços de acordo com sua necessidade. Desta forma, é possível que objetos em diferentes máquinas virtuais Java interajam entre si, independentemente da localização física dessas máquinas.

A tecnologia RMI foi disponibilizada a partir da versão 1.1 do Kit de Desenvolvimento Java (Java Development Kit - JDK), e sua Interface de Programação de Aplicação (Application Programming Interface - API) é especificada no pacote **java.rmi**.

Aplicações distribuídas orientadas a objeto apresentam três características principais [4]: localização de objetos remotos, comunicação entre objetos remotos e definição de classes (dos objetos que são transmitidos).

3.1 Localização de objetos remotos

O cliente necessita de uma maneira para descobrir quais os objetos remotos providos pelo servidor. Para tanto, uma referência a estes objetos é fornecida ao cliente. Este mecanismo é denominado Serviço de Nomes RMI, implementado através do aplicativo *RMI Registry*.

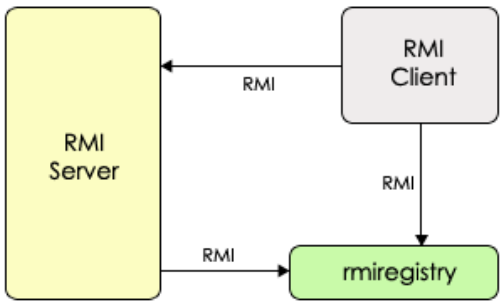

Figura 2: Localização de objetos remotos.

O processo de localização de objetos remotos no RMI é demonstrado na figura 2. O servidor RMI, que está sendo executado em uma máquina virtual, envia uma mensagem ao *RMI Registry*, associando um objeto (que contém os serviços disponibilizados pelo servidor) a um nome. O cliente, por sua vez, procura o objeto remoto através do seu nome consultando o *RMI Registry*. Desta forma, o cliente invoca o método desejado.

3.2 Comunicação entre objetos remotos

Os objetos (nas diferentes máquinas virtuais) devem ser capazes de comunicarem-se. Com a utilização de RMI, esta comunicação é realizada de forma transparente ao programador, dando a impressão de estar invocando métodos de objetos locais.

A arquitetura RMI oferece a transparência de localização através da organização de três camadas entre os objetos cliente e servidor: *Stub/Skeleton*, Gerenciador de Referências Remotas e Camada de Transporte RMI [5].

A camada *Stub/Skeleton* é a responsável por lidar com aspectos da rede, deixando transparentes todas as questões relacionadas a comunicação (empaco-

tamento e desempacotamento de dados). O cliente invoca métodos do *stub* que correspondem aos serviços implementados no servidor. Entretanto, os métodos no *stub* contém diversos aspectos relacionados a rede necessários para invocação remota. Por exemplo, se o servidor implementa o método **pow(int, int)**, o *stub* também terá a definição deste método. Porém, o *stub* não conterá a implementação propriamente dita, mas sim, o código necessário para acessar o *skeleton* remoto. A partir da versão 1.2 do *Java Development Kit* (JDK) as funções realizadas pelo *skeleton* foram incorporadas no próprio servidor.

A camada responsável por gerenciar as referencias remotas é o *middleware* entre a camada de *Stub/Skeleton* e o protocolo de transporte. Em outras palavras, quando ocorre uma consulta ao *RMI Registry*, os objetos resultantes desta requisição são gerenciados por esta camada.

A última camada, denominada Camada de Transporte RMI, oferece o meio necessário para a comunicação entre os objetos cliente e servidor através da rede.

Abaixo, figura 3, é ilustrada a organização destas três camadas em uma aplicação RMI.

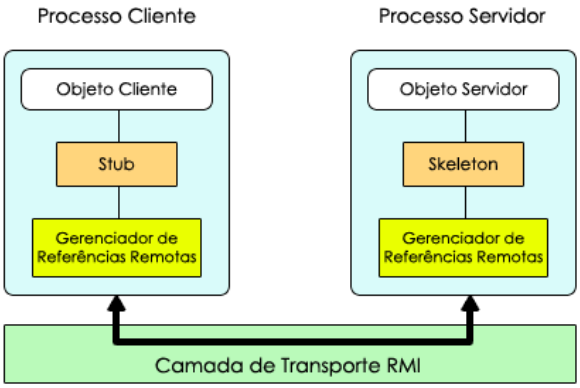

Figura 3: Arquitetura básica de uma aplicação RMI.

3.3 Definição de classes

Para desenvolver uma aplicação utilizando Java RMI, é necessário que o programador defina uma interface de serviços, os quais serão disponibilizados pelo objeto servidor. Esta interface criada deverá, obrigatoriamente, extender a interface *Remote* do pacote **java.rmi**. Por tratar-se de operações de acesso a métodos remotos, estes métodos devem ser capazes de disparar uma exceção em caso de erro. Esta exceção é chamada *RemoteException* e deve constar na defi-

nição de todos os métodos contidos nesta interface (*throws RemoteException*). Este conceito de interface é próprio da linguagem Java, isentando o programador do conhecimento de uma determinada IDL.

Criada a interface do servidor, é necessário o desenvolvimento de uma classe que implemente os métodos definidos nela. O objetivo desta é codificar as funcionalidades que devem ser providas pelo servidor. Além de implementar a interface de serviços, esta classe deve extender a classe *UnicastRemoteObject*, tornando-a disponível para receber chamadas remotas.

Por fim, deverá ser desenvolvida a parte principal do servidor que instanciará a implementação da interface anteriormente dita, associando o objeto criado a um nome no registro de nomes mantido pelo *RMI Registry*. Com este vínculo, os serviços deste objeto tornam-se disponíveis para que os clientes acessem-os remotamente.

Uma vez que a interface e a classe do serviço tenham sido criadas e compiladas utilizando-se um compilador Java convencional, é necessário criar os correspondentes *stubs* e *skeletons*. Para tanto, utiliza-se o aplicativo compilador denominado **rmic**, disponibilizado juntamente com o JDK.

Do outro lado, deverá ser criada a aplicação cliente que irá acessar os serviços remotos oferecidos pelo servidor. Para tanto é necessário a localização destes serviços, realizada através de uma busca no registro de nomes do *RMI Registry*. Assim, uma referência ao objeto do servidor é obtida e seus métodos podem ser acessados quando necessário. O cliente tem a impressão de estar realizando chamadas a métodos locais. Entretanto, ao receber estas chamadas, o mecanismo Java RMI invoca-as remotamente, tornando transparente toda a comunicação entre o cliente e o servidor.

4 Conclusão

A utilização de RMI em sistemas distribuídos orientados a objetos mostra-se como uma solução eficaz para comunicação entre processos remotos. O uso da tecnologia soluciona problemas relacionados ao uso de RPC, o qual não permite utilizar tipos de dados complexos (objetos) e força o programador a conhecer uma IDL específica.

Dentre as vantagens oferecidas destacam-se a facilidade de uso e a transparência de comunicação. Por utilizar o conceito de interface da linguagem Java, a criação de aplicações RMI é facilitada. E, por tratar a invocação de métodos remotos como aparentemente locais, o programador não precisa se preocupar com as questões de comunicação em baixo nível.

Referências

[1] SINHA, P. K. *Distributed Operating Systems*. Piscataway, NJ: IEEE Press, 1997. ISBN 0780311191.

[2] ZOMAYA, A. Y. *Parallel and Distributed Computing Handbook*. New York, NY: McGraw Hill, 1996. ISBN 0070730202.

[3] DEITEL, H. M.; DEITEL, P. J. *Java how to program*. 4th. ed. Upper Saddle River, NJ: Prentice Hall, 2002. ISBN 0130341517.

[4] Trail: RMI (The Java Tutorials).
Disponível em: <http://java.sun.com/docs/books/tutorial/rmi/index.html>.
Acesso em: 2 de abril 2007.

[5] POO: Programação Distribuída.
Disponível em: <http://www.dca.fee.unicamp.br/cursos/PooJava/progdist.html>.
Acesso em: 2 de abril 2007.