Capítulo 4

Objetos distribuídos

A programação estruturada possui funções e procedimentos que são implementados globalmente na aplicação. Já na programação orientada a objetos, os métodos são aplicados aos dados de cada objeto. A junção de sistemas distribuídos e da programação orientada a objetos fez surgir o termo "objetos distribuídos", portanto, o objeto distribuído refere-se ao uso da programação orientada a objetos em um ambiente distribuído.

Neste capítulo, trataremos dos modelos de programação para aplicativos formados por programas que estão cooperando e executando diversos processos distintos. Esses programas precisam executar operações em outros processos, que estão em computadores distintos.

Para uma melhor compreensão, são apresentados neste capítulo os conceitos de objetos distribuídos e de arquitetura de objetos, além de operações com objetos remotos.

1 Conceito de objetos distribuídos e arquitetura de objetos

Para Tanenbaum e Van Steen (2008), a orientação a objetos é um paradigma importante em desenvolvimento de software e desde que foi proposta tornou-se fundamental e utilizada em sistemas distribuídos, pela sua capacidade de embutir os programas em componentes. Desse modo, os desenvolvedores podem se concentrar na implementação de funcionalidades específicas independentemente de outros desenvolvedores, utilizando o objeto distribuído para atender a esse objetivo.

Um objeto distribuído é um objeto que pode ser acessado remotamente, ou seja, de qualquer lugar da rede. Considera-se tradicionalmente que um objeto encapsula dados e comportamento, conforme definido por Tanenbaum e Van Steen (2008):

> A característica fundamental de um objeto é que ele encapsula dados, denominados estado, e as operações executadas nesses dados, denominadas métodos. Métodos são disponibilizados por meio de uma interface. É importante entender que não há nenhum modo legal pelo qual um processo possa acessar ou manipular o estado de um objeto, exceto pela invocação dos métodos disponibilizados para ele por meio de uma interface de objeto. Um objeto pode implementar várias interfaces. Da mesma forma, dada uma definição de interface, pode haver vários objetos que oferecem uma implementação dela. (TANENBAUM; VAN STEEN, 2008, p. 268)

Assim, torna-se possível que uma aplicação cliente orientada a objeto obtenha uma referência para o serviço pretendido e, através dessa referência, mesmo em um servidor remoto, possa invocar métodos desse objeto.

Segundo Coulouris, Dollimore e Kindberg (2007), a sustentação da plataforma de objetos distribuídos é suportada por uma arquitetura de objetos. Uma arquitetura tem como principal função o estabelecimento de regras, convenções e diretrizes de como as aplicações podem se comunicar entre si.

O conceito de desenvolvimento de objetos distribuídos na plataforma Java é oferecido por dois mecanismos: Java RMI e Java IDL. O RMI (remote method invocation – invocação remota de método) é um mecanismo de desenvolvimento de aplicações com sistemas distribuídos que trabalha exclusivamente com objetos Java.

Na aplicação prática do Java RMI, através do pacote java.rmi, são criados dois programas: um no servidor e outro no cliente. No programa servidor, um objeto remoto é criado, e a referência desse objeto é disponibilizada para o cliente. Já o programa cliente solicita os objetos remotos no servidor através de um método de invocação. A comunicação entre o cliente e o servidor é tratada por meio de dois objetos intermediários: objeto stub (do lado do cliente) e objeto skeleton (do lado do servidor).

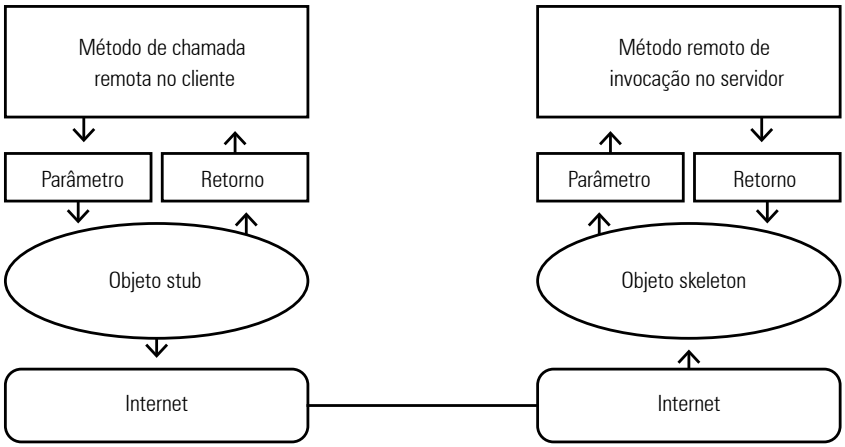
A figura 1 descreve o funcionamento do RMI; o objeto stub na máquina cliente constrói um bloco de informações e envia essas informações ao servidor. O bloco consiste de três parâmetros:

1. identificador do objeto remoto a ser usado;
2. nome do método que deve ser invocado;
3. parâmetros para a máquina virtual Java remota.

O objeto skeleton, então, passa a solicitação do objeto stub para o objeto remoto, executando as seguintes tarefas:

- chama o método desejado no objeto real presente no servidor;
- encaminha os parâmetros recebidos do objeto stub para o método.

Figura 1 – Funcionamento do RMI



Já o Java IDL (interface definition language – interface de definição de linguagem) é similar ao Java RMI, porém utiliza a arquitetura CORBA para integrar aplicações Java com aplicações desenvolvidas em outras linguagens (BASHAM; SIERRA; BATES, 2005). O CORBA (Common Object Request Broker Architecture) basicamente utiliza IDL para definir interfaces e o compilador Java IDL para gerar stubs e skeletons. Veja a seguir um exemplo em Java de uma interface IDL chamada "Ola", dentro do módulo chamado "OlaApp". Essa interface criada, "Ola", receberá uma mensagem do tipo string e emitirá uma mensagem no cliente e também no servidor.

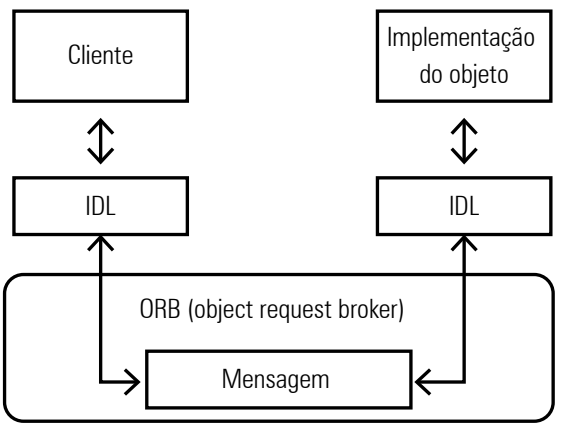
Ola.idl

```
module OlaApp
{
 interface Ola
 {
 string digaOla();
 oneway void shutdown();
 };
};
```

A arquitetura CORBA define como objetos devem interoperar em um ambiente distribuído, possuindo também uma linguagem para definição de interface (IDL) cuja principal função é especificar as interfaces dos objetos distribuídos de maneira que possam solicitar serviços a eles. O solicitante conhece somente uma interface de um objeto que está distribuído pela rede e após a execução apenas aguarda pela resposta. Toda a chamada entre os objetos deve ser feita pelo object request broker (ORB), que é responsável pela comunicação. Dessa maneira, para o solicitante, os serviços são atendidos de forma transparente, e, para o objeto distribuído, todas as solicitações comportam-se da mesma forma (COULOURIS; DOLLIMORE; KINDBERG, 2007).

Na figura 2, é possível visualizar a comunicação do solicitante, cliente, com o objeto distribuído através de IDL e ORB. Veja que o cliente faz a chamada para a interface IDL, e esta encaminha para o ORB, que, por sua vez, encaminha para a interface IDL do destinatário; este, após a execução, retorna para o solicitante, seguindo os mesmos passos definidos anteriormente.

Figura 2 – Funcionamento da CORBA



PARA SABER MAIS

CORBA é uma arquitetura criada pelo Object Management Group (OMG) e apoiada por diversas empresas para padronizar a comunicação entre sistemas distribuídos heterogêneos. No site oficial da OMG, é possível encontrar documentação, estudos de casos e implementações sobre o assunto. Recomendamos a leitura do material intitulado "Common Object Request Broker Architecture".

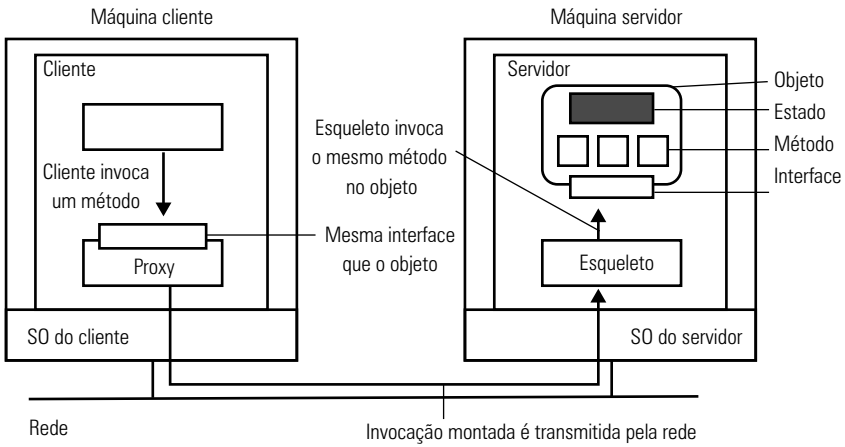
2 Operações com objetos remotos

Como vimos, a separação entre interfaces e objetos permite colocar uma interface em uma máquina e o objeto em outra, o que chamamos de objeto distribuído. Em termos operacionais, quando um cliente se conecta a um objeto distribuído, carrega-se no espaço de endereço do cliente um proxy, que é a implementação de uma interface do objeto. O proxy tem como principal objetivo criar as chamadas a métodos em mensagens e desmontar mensagens de resposta para retornar o resultado.

A característica principal dos objetos distribuídos é que seu estado não é distribuído, mas reside em uma única máquina, sendo somente as interfaces implementadas pelo objeto disponibilizadas em outras máquinas; esses objetos são denominados objetos remotos (TANENBAUM; VAN STEEN, 2008).

Na figura 3 é possível visualizar a operação de um objeto remoto com o cliente, através da máquina do cliente e do servidor. Do lado do cliente, existe um proxy com a mesma interface do objeto. O apêndice do lado do servidor, chamado de esqueleto, também é responsável por montar respostas e enviar mensagens de resposta para o proxy do lado do cliente. Ele é denominado esqueleto pois fornece o mínimo necessário para que o middleware do servidor acesse os objetos definidos pelo usuário.

Figura 3 – Operação de um objeto remoto com o cliente



Fonte: Tanenbaum e Van Steen (2008).

NA PRÁTICA

OmniORB é um CORBA ORB de alto desempenho para C++ e Python. Ele está disponível gratuitamente sob os termos da GNU Lesser General Public License (para as bibliotecas) e da GNU General Public License (para as ferramentas). No site oficial da OmniORB, existe um material de autoria de Duncan Grisby detalhando a implementação da CORBA em Python. Busque por "The omniORBpy version 4.2 User's Guide" e tente praticar os exemplos oferecidos neste guia.

Considerações finais

Neste capítulo, aprendemos que a união de sistemas distribuídos e programação orientada a objetos fez surgir o termo "objetos distribuídos" e que este refere-se ao uso da programação orientada a objetos em um ambiente distribuído.

Além do conceito de objetos distribuídos, foi possível entender também sua arquitetura. Em especial, vimos a arquitetura CORBA e verificamos as operações com objetos remotos por meio de um exemplo didático.

Referências

BASHAM, Bryan; SIERRA, Kathy; BATES, Bert. **Use a Cabeça!**: servlets & JSP. Rio de Janeiro: Alta Books, 2005.

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. **Sistemas distribuídos**: conceitos e projeto. 4. ed. Porto Alegre: Bookman, 2007.

TANENBAUM, Andrew S.; VAN STEEN, Maarten. **Sistemas distribuídos**: princípios e paradigmas. São Paulo: Prentice Hall, 2008.