

Sistema Servidor Cliente



Educação a Distância
Cruzeiro do Sul Educacional
Campus Virtual

Material Teórico



Ambiente de programação distribuída

Responsável pelo Conteúdo:

Prof Esp Marcio Funes

Revisão Técnica:

Prof. Ms. Luiz Carlos Reis

Revisão Textual:

Profa Ms. Luciene Oliveira da Costa Santos

UNIDADE

Ambiente de programação distribuída



- Sockets
- Corba
- Dcom
- Soap
- Java RMI
- Web Service



Objetivo de APRENDIZADO

Nesta unidade, falaremos sobre o ambiente necessário para a programação distribuída. Vamos entender quais são os conceitos relacionados ao ambiente que possibilitam desenvolvedores utilizarem os benefícios da computação distribuída. Isso é muito importante, pois você poderá utilizar tais conceitos em sua carreira.

Também é importante ver como as soluções de programação são criadas para um serviço, envolvendo máquinas que troquem informações e consigam manter um link e, principalmente, desenvolver a relação de Cliente e Servidor de forma confiável.

Nesta segunda unidade, você irá estudar os principais conceitos que ambientam a programação distribuída como sockets, corba, dcom, soap, Java RMI e Web Services.

O primeiro material a ser lido, que se encontra no material didático, é a Contextualização. Você deverá ler o conteúdo disponibilizado nesse link, cujo objetivo é apresentar um texto, situando o assunto que irá aprender e vinculando-o a situações cotidianas.

Depois, você deverá ler o conteúdo teórico. Nele, você encontrará os assuntos citados no parágrafo acima, com os quais serão definidos o conceito de cada tema e o modo de funcionamento geral de cada item. É importante que leia atentamente o conteúdo e consiga identificar a diferença entre cada ambiente de programação e seus aspectos de uso.

Após estudar o texto, você deverá analisar o slide disponível. Também por meio dele, você poderá estudar os pontos relevantes sobre o assunto desta primeira aula.

Com os conceitos já estudados, você estará pronto para participar do fórum e realizar as atividades. Primeiramente, você irá fazer a Atividade de Sistematização, depois, a Atividade de Reflexão.

Contextualização

Agora que já vimos os conceitos iniciais da tecnologia de sistemas cliente / servidor, podemos avançar e entender como os ambientes de programação distribuída funcionam. A compreensão de como a tecnologia distribuída em cliente / servidor acontece através das arquiteturas lhe permitirá agora entender como aplicamos esses conceitos dentro de ambientes de programação distribuída.

Para obter um bom aproveitamento com o estudo desta unidade, vamos conferir sua estrutura:

- Conteúdo Teórico: neste link, você encontrará o material principal de estudos na forma de texto escrito.
- Atividade de Sistematização: os exercícios disponibilizados são de autocorreção e visam a que você pratique o que aprendeu na disciplina e que identifique os pontos aos quais precisa prestar mais atenção, ou sobre os quais necessita pedir esclarecimentos a seu tutor. Além disso, as notas atribuídas aos exercícios serão parte de sua média final na disciplina.
- Atividade de Aprofundamento: é uma atividade dissertativa.
- Material Complementar: nestes links, você poderá ampliar seus conhecimentos.
- Vídeo-aula: nestes links, serão apresentadas algumas ferramentas na prática e também a resolução de alguns exercícios.

Fique atento(a) à importância de realizar todas as atividades propostas dentro do prazo estabelecido para cada unidade, pois, dessa forma, você evitará que o conteúdo se acumule e problemas ao final do semestre.

Uma última recomendação: caso tenha problemas para acessar algum item da disciplina, ou dúvidas em relação ao conteúdo, não deixe de entrar em contato com seu professor tutor através do botão mensagens.

Sockets



Você já participou de algum bate papo na Internet?

Provavelmente, a resposta será sim. Uma aplicação desse tipo é muito comum entre usuários de internet. Se você é mais veterano, deve se lembrar do ICQ, em 1996 (caso sinta saudades, aproveite, pois ele ainda está disponível para download), ou pelo Windows Messenger em 2008 ou somente MSN para os íntimos. Exemplos atuais de bate-papo nos trazem ao Facebook ou através de aplicativo como o WhatsApp para aplicativos móveis.

Sabemos que comunicação de forma instantânea na internet é uma das suas maiores vantagens e essa possibilidade não irá desaparecer. O que pode ocorrer é a mudança do aplicativo ou serviço que utilizamos para comunicação.

Quero convidá-lo(a) a conhecer o que existe por trás do aplicativo que nos permite bater um papo e combinar uma festa no final de semana. Como programadores conseguem fazer com que mensagens saiam do meu aplicativo e cheguem ao seu destino?

A resposta será sockets, sim, sockets! E essas belezinhas não são usadas apenas em bate-papo. Sockets são usados quando existe a necessidade de comunicar dois pontos e necessitam de estabilidade, rapidez e desempenho.

O que é um Socket?

É um serviço de transporte que permite a comunicação através do protocolo TCP entre clientes e servidores. Cada cliente e servidor possuem seu socket que irá possibilitar uma conexão fim a fim.

Podemos entender que sockets são terminais em uma conexão bidirecional, a mensagem sai de um computador ou aplicativo A e chega ao seu fim em um computador ou aplicativo B. Todo socket possui dois tipos de endereçamento:

Endereço Local: Endereço da porta de comunicação para a camada de transporte.

Endereço Global: Endereço do computador na rede.

Vamos entender como os sockets funcionam. Primeiro, vamos analisar a visão do servidor.

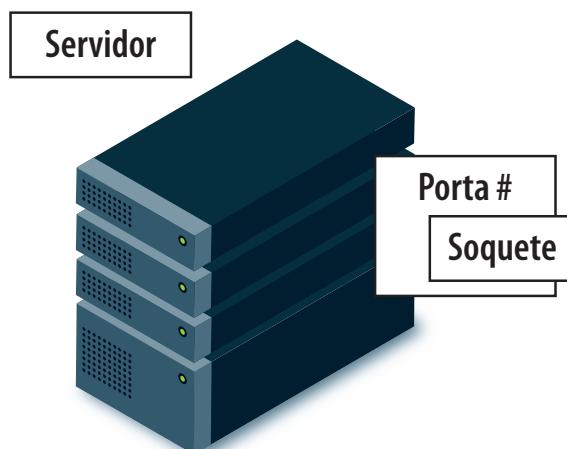


Figura 1 – Socket associado a uma Porta

Temos, na Figura 1, um servidor que oferece algum serviço através de uma de suas portas. O socket associado a essa porta ficará “escutando” a porta, esperando que algum cliente ansioso faça sua solicitação.

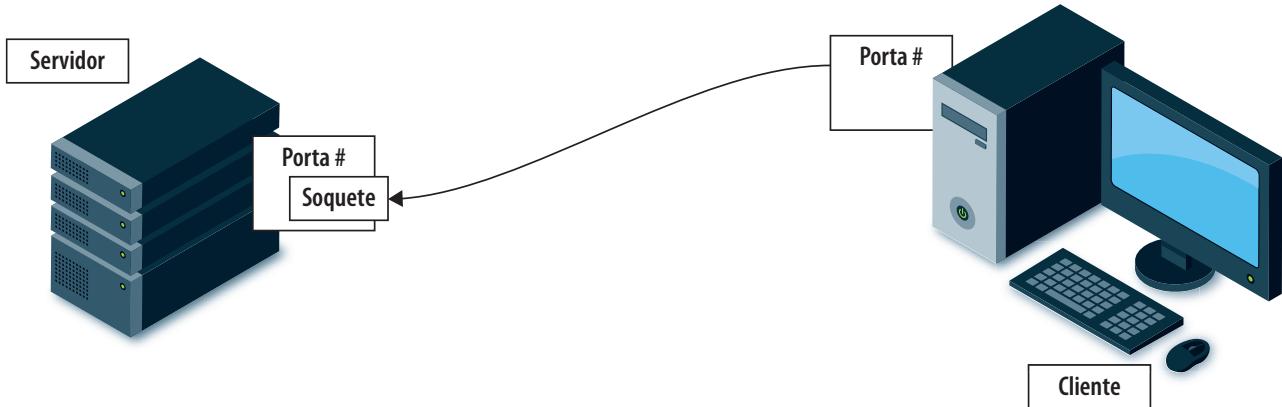


Figura 2 – Cliente e suas requisições

O cliente então deve saber identificar o servidor e o número da porta em que o servidor aguarda comunicação. Para facilitar, imagine a ação de enviar um e-mail, quando abrimos o Gmail, por exemplo, o servidor fica atento que algum cliente solicite conexão; por sua vez, o servidor através de uma porta e um socket responde à solicitação, exigindo login e senha para poder fornecer os e-mails contidos em seu repositório.

Assim, ocorre uma ligação bidirecional, por meio de portas e sockets, se estabelece uma conexão e ambos podem se comunicar, como mostra a Figura 2.

Quando um servidor está disponível para receber solicitações e um cliente aceita a conexão oferecida são criados sockets em ambas as pontas, estabelecendo a comunicação e definindo as diretrizes que podem ser lidas e escritas em ambos os sockets, como podemos ver na Figura 3.

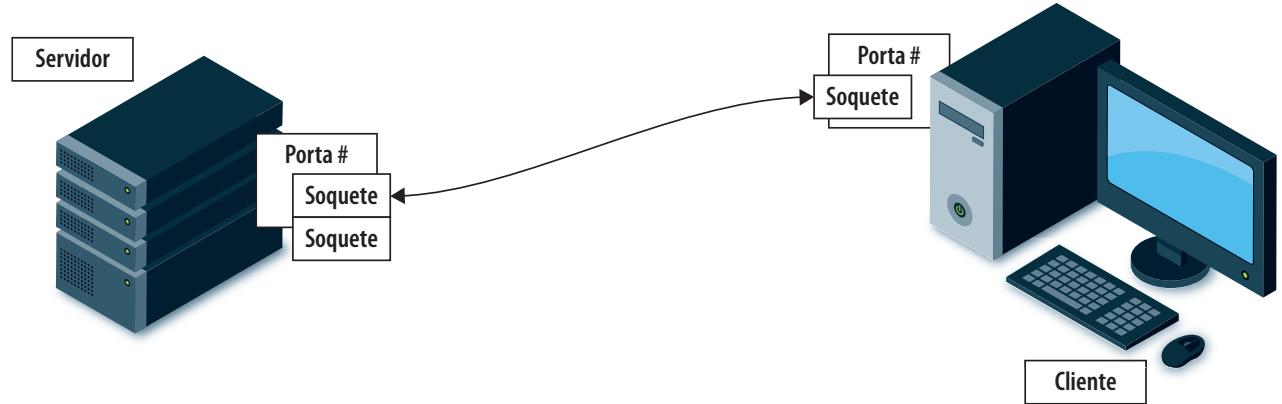


Figura 3 – Sockets conectados em ambas as pontas

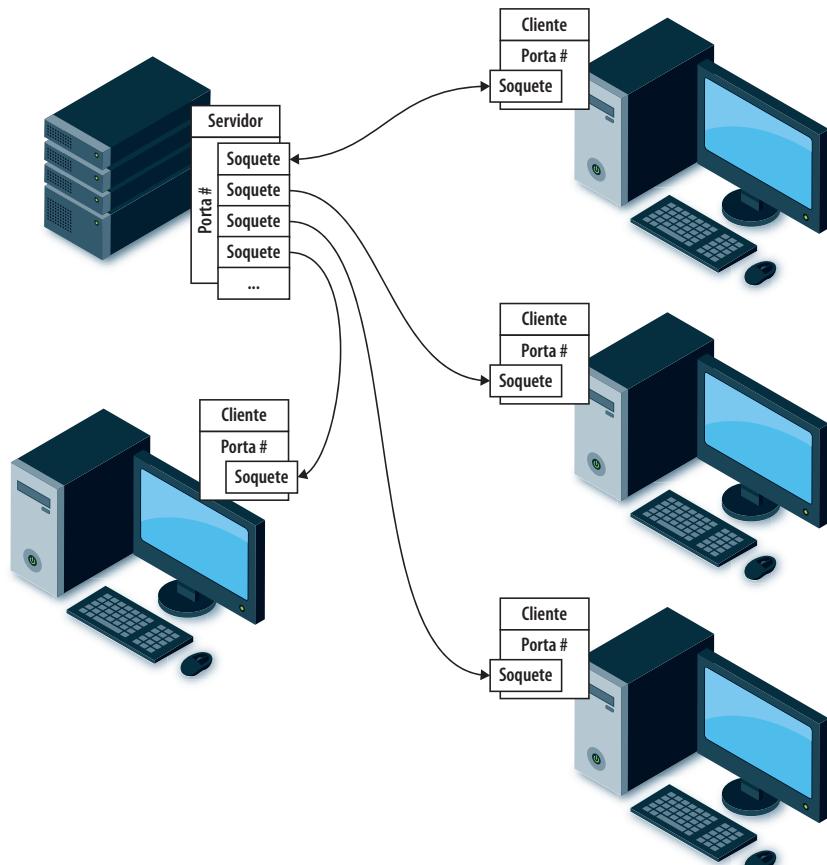


Figura 4 – Exemplo de sockets em um bate-papo

Voltando ao exemplo do bate-papo, quando vários clientes necessitam acessar o mesmo servidor, são criados vários sockets para estabelecer uma conexão única e estável com cada cliente, como demonstrado na Figura 4.



Acesse: <https://goo.gl/xzGmD>

Esse tutorial disponibilizado pela Oracle, além de trazer várias informações sobre sockets, traz um tutorial passo a passo de programação de sockets em Java.

CORBA



Em meados de 1990, a tecnologia de comunicações e serviços via rede cada vez mais se expandiam, novos serviços se espalhavam, fazendo com que a procura por soluções distribuídas aumentasse cada vez mais.

Porém, havia um empecilho que poderia comprometer o crescimento dessas soluções. Não havia uma padronização única que permitisse que diversos tipos de sistemas distribuídos pudessem se comunicar. Muitas vezes, uma solução era implementada em uma determinada linguagem diferente de outra solução, impedindo ambas de não se comunicarem.

Em 1991, um grupo de empresas estabelece uma especificação de uma arquitetura chamada CORBA (Common Object Request Broker Architecture) que define padrões, permitindo que aplicações feitas por diferentes desenvolvedores se comunicassem sem nenhum problema.

Desde então, ela é patrocinada pelo OMG (Object Management Group) grupo formado em 1989, com o objetivo de estimular a adoção de sistemas de objetos distribuídos para usufruir das vantagens da programação orientada a objeto no desenvolvimento de softwares para sistemas distribuídos (COULOURIS et al., 2007, p.711).

Qual o segredo do CORBA?

Vimos acima que o grande sucesso do CORBA é possibilitar a diferentes soluções a comunicação em uma rede entre si. Mas como ele faz isso?

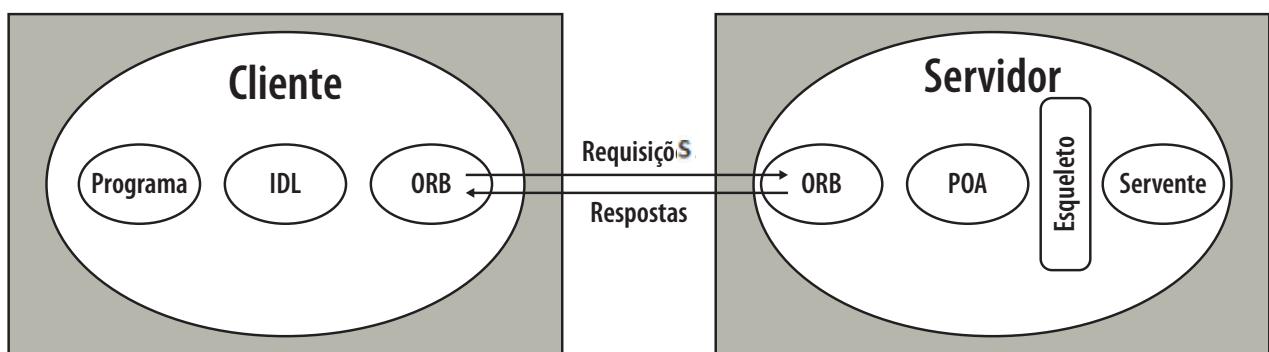


Figura 5 – Principais componentes CORBA.

Analizando a Figura 5, podemos entender que o objetivo principal é permitir que o programa que está sendo executado na máquina do cliente possa solicitar requisições do servidor e usufruir de seus serviços como é característico do modelo Cliente / Servidor. Porém, tanto o programa quanto o servidor são diferentes em termos de desenvolvimento. Nesse contexto, temos que criar um Middleware, ou seja, um conjunto de elementos que irão mediar a comunicação entre o cliente e o servidor.

Vamos analisar agora cada elemento que compõe o CORBA.

- **Programa:** é a aplicação que o usuário tem acesso, ou seja, a aplicação que o usuário deseja utilizar.
- **IDL (Interface Definition Language):** uma linguagem para definição de interface que permite padronizar as chamadas aos métodos CORBA com o IDL não importando qual tecnologia seja executada, pois, quando ela passa pela IDL, é padronizada pelas definições CORBA.
- **ORB (Object Request Broker):** define as funções que um cliente irá utilizar e um servidor irá receber, ele terá a tarefa de interpretar aquilo que o cliente deseja e aquilo que é respondido pelo usuário, fazendo assim que servidor e cliente “falem” a mesma língua.
- **POA (Portable Object Adapter):** presente apenas no servidor, tem a função de ajudar o

ORB na tarefa de entregar as requisições ao servente. Permite que sejam criadas aplicações e servidores portáveis internos para atender a qualquer tipo de requisição do ORB.

- **Esqueletos:** segundo Couloris et al. (2007), esqueletos são classes geradas na linguagem do servidor por um compilador de IDL, que tem por função desempacotar os argumentos nas mensagens de requisições (vindas dos clientes) e empacotar exceções e resultados nas mensagens de respostas geradas por um servente. Nesse desempacotar e empacotar, consegue estipular qual o tipo da requisição e enviá-la a um servente correto, ou estabelecer a qual cliente a resposta deverá ser enviada pelo ORB.
- **Servente:** implementa as operações que o servidor possui através de chamada de métodos, permitindo ao servidor entender a requisição do cliente e responder aquilo que lhe foi pedido.

DCOM



Entendendo as tecnologias ligadas aos sistemas distribuídos e os benefícios que ela pode trazer, a Microsoft percebe que deve investir recursos voltados a esse tema e, em 1996, lança no mercado seu modelo de ambiente de programação distribuído chamada DCOM (Distributed Component Object Model).

Muito semelhante ao modo de trabalho do CORBA, o DCOM também permite que objetos se tornem acessíveis, possibilitando acesso a serviços localizados em servidores, caracterizando, assim, uma comunicação cliente/servidor.

Como principal ponto, a Microsoft fornece soluções transparentes dentro do DCOM. Vejamos abaixo alguns exemplos:

- Transparência de localização: através de ponteiros, os clientes conseguem localizar objetos que estão inseridos em servidores e, assim, eles podem utilizar os serviços disponíveis apenas invocando o método da interface como se essa invocação fosse local.
- Transparência de plataforma: sabemos que sistemas distribuídos operam em multiplataformas devido à gama de sistemas operacionais e hardwares que assumem papéis de cliente e servidores. Possibilitar que todos se comuniquem e troquem requisições e respostas é uma característica dos ambientes de programação distribuída, o DCOM implementa essa funcionalidade.
- Transparência de linguagem de programação: assim como plataformas podem ser diferentes, as linguagens de programação que estão presentes em clientes podem ser diferentes em servidores. Se clientes tivessem a obrigatoriedade de possuir a mesma codificação de servidores, os serviços fornecidos seriam muito restritos e, em muitos casos, a solução seria inviável.

SOAP



Como você já deve ter percebido até agora, o segredo dos sistemas cliente/ servidor e os sistemas distribuídos em geral está em seus bastidores, nas programações, protocolos e arquiteturas que são criadas para aperfeiçoar o modo de comunicação entre hardware e software distribuído.

Muitos desenvolvedores criam soluções fantásticas para clientes finais, ou seja, criam sites, sistemas, serviços e soluções que agradam ao usuário final e o fazem interagir com suas criações. Você se lembra de quando acessou sua primeira rede social? Ou quando usou algum serviço como compra, ou se comunicou com alguém pela internet e pensou “Uau, isso é realmente muito legal”?

Vamos ver agora um protocolo chamado SOAP que permite justamente a comunicação ao usar um sistema distribuído que você tanto gosta, mas talvez não saiba bem como ele funciona em sua parte mais técnica.

CONHEÇA O SIMPLE OBJECT ACESST PROTOCOL

O SOAP tem seu foco no envio de qualquer informação através de um sistema distribuído. Ele surgiu em meados de 1999 e inicialmente foi projetado apenas para o protocolo HTTP. Atualmente, define esquemas para outros protocolos de transporte e especialmente define meios de uso para o uso da linguagem XML.

Segundo Coulouris et al. (2007), para suportar comunicações entre cliente e servidores, o SOAP define como o método de envio da mensagem de requisição e da mensagem de resposta como deve se comportar.

Lembre-se de que o cliente e o servidor podem estar a quilômetros de distância um do outro. Uma requisição que parte do cliente em direção ao servidor passa por diversos middlewares pelo caminho e isso requer certa segurança. Afinal, não queremos que qualquer um possa ler nossas informações.

Uma mensagem SOAP é transportada em um envelope que, por sua vez, possui cabeçalho e um corpo em sua composição. Vejamos a função de cada elemento:

Cabeçalho: permite que intermediário entre cliente e servidor possam tomar ações sobre o que fazer com o envelope, analisando seu conteúdo e para onde o envelope deverá ser encaminhado. Assim como um carteiro lê o destinatário de uma carta e sabe onde deve entregá-la, o cabeçalho de um envelope no protocolo SOAP funciona, a diferença é que os intermediários podem acrescentar e editar informações.

Corpo: o corpo contém informações sobre a validade do conteúdo da mensagem e garante que não existem erros e que o conteúdo está correto para ser entregue.

EXEMPLO

Abaixo, temos um exemplo de requisição de um cliente a um servidor apenas iniciando uma simples comunicação:

```
POST /EnviandoHelloWord/endpoint HTTP/1.1
Host: (aqui irá o endereço do destinatário)
Content-Type: text/XML; charset="utf-8"
Content-Length: 322
```

Cabeçalho:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://hello">
```

Corpo:

```
<soapenv:Body>
<ns1:sayHello>
  <ns1:name>Olá</ns1:name>
</ns1:sayHello>
</soapenv:Body>
</soapenv:Envelope>
```

Vejamos agora a resposta do Servidor:

```
HTTP/1.1 200 OK
Content-Type: text/XML; charset="utf-8"
Content-Length: 367
```

Cabeçalho:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://hello">
```

Corpo:

```
<soapenv:Body>
<ns1:sayHelloResponse>
  <ns1:return>Olá, Tudo bem?</ns1:return>
</ns1:sayHelloResponse>
</soapenv:Body>
</soapenv:Envelope>
```

Java RMI



Sabemos que, quando se trata de ambiente de desenvolvimento em Java, um dos conceitos fundamentais é a possibilidade de criação de objetos e manipulação dos mesmos de diversas formas. Porém, quando se trata de desenvolvimento Java em um ambiente distribuído, onde temos objetos em clientes e objetos em servidores?

Nesse caso, utilizamos o conceito de Java RMI (Remote Method Invocation), que possibilita invocar qualquer objeto localizado em um cliente ou servidor mesmo que esse cliente ou servidor não possua conhecimento do objeto que está invocando. Vamos ver como isso funciona:

Tudo começa entendendo a ligação de Máquinas Virtuais Java ou conhecidas como JVM (Java Virtual Machine). Perceba que um cliente pode possuir uma interface Java e deseja acessar algum serviço localizado em outra JVM que esteja, em um servidor, por exemplo. Através do RMI, a Máquina Virtual pode invocar um método que está armazenado em um cliente apenas esperando ser acessado via rede.

Os métodos que são invocados pelo cliente podem inclusive passar objetos localizados em servidores que nunca foram vistos pelo cliente antes, tirando assim a preocupação do desenvolvedor de implementar objetos em clientes. Podemos entender isso como carregamento dinâmico de classes e objetos.

Considere um desenvolvedor expandindo as funcionalidades de sua solução Java. Nesse cenário, imagine aplicadas a criação e a modificação de classes a todo o instante, adicione também um cliente que queira acessar essas funcionalidades, mas não tenha cópia dessas novas classes (sejam públicas ou privadas) no mesmo instante em que elas são atualizadas pelo desenvolvedor inicial.

A solução nesse cenário é a utilização do cliente RMI que busca automaticamente as novas classes do servidor onde é compartilhada a classe e, consequentemente, a solução que ela traz. A nova classe é carregada na memória e o cliente RMI pode utilizá-la tranquilamente.

Vejamos um exemplo de código para a criação simples de um servidor em Java RMI. Note como é simples a criação do novo objeto e como podemos compilar o mesmo:

```
import java.rmi.Naming;  
  
public class ServerTeste //considere a criação da classe ServerTeste anteriormente  
{  
    public static void main(String[] args)  
    {  
        if (args.length != 1)  
        {  
            System.err.println("\nUsage:\tjava ServerTeste objname\n");  
            System.exit(1);  
        }  
    }  
}
```

```
try
{
    Teste ateste = new Teste(); // criação do objeto servidor
    String objname = "//localhost/" + args[0];
    System.out.println("Registrando" + objname + "...");
    Naming.rebind(objname, awalk); // Registro do RMI
    System.out.println("Registrado");
}
catch (Exception e) // catch caso ocorra erro no registro do RMI
{
    System.err.println("Erro em main()" + e);
    e.printStackTrace();
    System.exit(2);
}
System.out.println("Esperando Objeto");
}}
```

Para compilar o servidor acima utilizamos a expressão:

```
servhost> javac ServerTeste.java
```

Web Service



Como último tópico, temos o conceito de Web Service que podemos entender como a solução final implementada, podendo utilizar vários ambientes de programação distribuída trabalhando em conjunto.

Ao entender o conceito de SOAP, Java RMI e diversos ambientes de programação distribuída, temos um olhar focado naquela solução que desejamos desenvolver. Porém, na integralização de sistemas, um olhar mais amplo, ou seja, ou olhar macro sobre a solução é necessária e é através deste olhar que podemos ver Web Service sendo criados.

Um WebService é um fragmento de lógica de negócio, localizado em algum lugar na Internet, que é acessível através de protocolos de Internet baseados em padrões, como HTTP ou SMTP. Usar um Web Service pode ser tão simples quanto efetuar um login em um site ou tão complexo quanto facilitar uma negociação multiorganizacional. (CHAPPELL, D. JEWELL, T., p. 01, tradução nossa).

Em tradução livre, David considera um Web Service como um pedaço da lógica de um negócio, que está localizada em algum lugar da internet e pode ser acessada por protocolos como HTTP ou SMTP. David também traça um paralelo, dizendo que Web Service pode ser visto desde um simples login em algum site, até a complexa negociação de multinacionais através de algumas tecnologias.

As principais tecnologias inseridas em Web Service são:

- **SOAP:** como vimos anteriormente, o SOAP se importa com a estrutura de empacotamento para transportar informações em XML através de protocolos de Internet (HTTP, SMTP, FTP).
- **WSDL:** tecnologia que padroniza a descrição da interface de um web service. Ele define como serão os parâmetros de entrada e saída de uma chamada externa.
- **UDDI:** repositório mundial de Web Services, permite consultar diversas soluções já criadas, mais informações em: <http://uddi.xml.org/>

Material Complementar

Como complemento desta unidade, sugiro a leitura completa do capítulo I do livro:

- KALIN, M. **Java Web Services: Implementando.** Alta Books, 2009.

Referências

CHAPPELL, D.; JEWELL, T. **Java Web Services.** O'reilly, 2002.

COULORIS, G.; DOLLIMORE, J. AND KINDBERG, T. **Sistemas Distribuídos: Conceitos e Projeto 4/E.** Bookman, 2007.

HANSEN, M. D. **Soa Using Java Web Services.** New Jersey: Prentice Hall, 2007.

KOCHMER, C. **Jsp And Xml: Integrating Xml And Web Services In Your Jsp Application.** Boston: Addison-Wesley, 2002.

TANENBAUM, A.S.; VAN STEEN, M. **Sistemas Distribuídos: Princípios e Paradigmas 2/E.** Prentice Hall, 2007.

Anotações





Educação a Distância
Cruzeiro do Sul Educacional
Campus Virtual

www.cruzeirodosulvirtual.com.br
Campus Liberdade
Rua Galvão Bueno, 868
CEP 01506-000
São Paulo SP Brasil
Tel: (55 11) 3385-3000



Universidade
Cruzeiro do Sul



UNICID
Universidade
Cidade de S. Paulo



UNIFRAN
Universidade
de Franca



UDF
Centro
Universitário



Módulo
Centro
Universitário