Interoperabilidade de Aplicações

Alex Coelho

Anotações

Créditos

EQUIPE UNITINS

Organização de Conteúdos Acadêmicos	Alex Coelho
Coordenação Editorial	Maria Lourdes F. G. Aires
Revisão Linguístico-Textual	Ivan Cupertino Dutra
Revisão Didático-Editorial	Silvéria Aparecida Basniak Schier
Gestão de Qualidade	Sibele Letícia Rodrigues de Oliveira Biazotto
Gerente de Divisão de Material Impresso	Katia Gomes da Silva
Revisão Digital	Leyciane Lima Oliveira Vladimir Alencastro Feitosa
Projeto Gráfico	Albânia Celi Morais de Brito Lira Katia Gomes da Silva Márcio da Silva Araújo Rogério Adriano Ferreira da Silva Vladimir Alencastro Feitosa
Ilustração	Geuvar S. de Oliveira
Capas	Rogério Adriano Ferreira da Silva

EQUIPE EADCON

Coordenador Editorial	William Marlos da Costa
Assistentes de Edição	Cristiane Marthendal de Oliveira Jaqueline Nascimento Lisiane Marcele dos Santos Silvia Milena Bernsdorf Thaisa Socher
Programação Visual e Diagramação	Ana Lúcia Ehler Rodrigues Bruna Maria Cantador Denise Pires Pierin Kátia Cristina Oliveira dos Santos Sandro Niemicz

Apresentação

Caro estudante do curso de Análise e Desenvolvimento de Sistemas, seja bem-vindo à disciplina de Interoperabilidade de Aplicações. Iniciaremos agora o estudo de uma disciplina extremamente inovadora e diferenciada das demais técnicas e conceitos de programação vistos até aqui por você. O eixo central do conceito de interoperabilidade, com o passar dos anos, deixou de ser algo apenas teórico e retórico e passou a ser uma das técnicas mais utilizadas na criação de *softwares*, estabelecendo um novo foco e forma de fornecer sistemas, seguindo uma nova formatação, no caso, a disponibilização de serviços em sua maioria *on-line*.

O termo *on-line* nos remete à utilização da internet, o que, para a interoperabilidade entre sistemas, com certeza, foi um marco, já que possibilitou a comunicação e a troca de dados e informações entre sistemas, sem nos esquecermos, é claro, de linguagens e técnicas que se utilizam apenas das características das redes de computadores como CORBA e RMI na programação distribuída. Nossa disciplina consiste em um novo e interessante conceito na tecnologia que tem feito com que programadores e gestores de TI passem a considerar uma mudança no foco quanto ao desenvolvimento de suas aplicações ou disponibilização de serviços.

No primeiro capítulo, estudaremos os fundamentos da interoperabilidade de aplicações com *Web Services*. No segundo, veremos como se dá a criação de documentos XML, que é um padrão na troca de mensagens entre diferentes servidores e aplicações e é a base para a interoperabilidade entre aplicações. No terceiro capítulo, apresentaremos as principais APIS para a manipulação de documentos XML disponíveis no mercado, utilizados na linguagem Java. No quarto capítulo, analisaremos um dos mais importantes protocolos e responsável por possibilitar a comunicação entre *Web Services*, o protocolo SOAP.

No quinto capítulo, conheceremos a função dos arquivos WSDL para um *Web Service*. No sexto, colocaremos em prática os conhecimentos obtidos durante a disciplina, criaremos um *Web Service* Java. E, no último capítulo, examinaremos a UDDI para a publicação de serviços. Procure aproveitar ao máximo tudo o que será exposto em nossa disciplina.

Prof. Alex Coelho

Interoperabilidade com *Web Services*

1

Introdução

Agora abordaremos uma nova forma de trabalhar e de disponibilizar serviços, estudaremos um tema inovador, já que se trata de um conceito extremamente abrangente, a interoperabilidade. Abordaremos diversas tecnologias que, em conjunto, possibilitam uma nova forma de olhar a distribuição de aplicações conhecidas aqui como serviços. Esse novo paradigma é conhecido como SOA (*Service-Oriented Architecture*). Ele consiste em uma arquitetura de *software* baseada no conceito chave de aplicações que deixam de ser vistas como um todo e passam a considerar as suas funcionalidades como serviços específicos utilizados por diversas aplicações distribuídas, o que torna o processo de reutilização mais abrangente (CHAPPELL; JEWELL, 2002).

Se você se lembra, o processo de reutilização de código é algo comum para programadores Java, mas a forma como ele é trabalhado, com a utilização dos conceitos de SOA, faz com que o código deixe de ser tão importante, e as funcionalidades ou os serviços passem a ser mais importantes. Mas como são tratatas e trabalhadas tais perspectivas na arquitetura SOA? Poderíamos citar diversos elementos dessa arquitetura que possibilitam a interoperabilidade de sistemas, mas em geral as aplicações que trabalham baseado em SOA têm um *frontend*, ou seja, uma aplicação que faz o acesso aos serviços que estão armazenados e disponibilizados em um repositório de serviços e que são fornecidos ou expostos para utilização na forma de barramento de serviços (*service bus*), que nada mais são que interfaces para acesso e execução dos serviços (DAUM; MERTEN, 2002). Em sua maioria, essas aplicações são conhecidas como *Web Services*, tecnologia que conheceremos neste capítulo.

Para o entendimento desse conteúdo, é interessante que você tenha conhecimento de detalhes comuns em aplicações para internet, como os servidores *web*, uma vez que aprenderemos conceitos sobre a interoperabilidade, bem como sobre a nova metodologia que é adotada por esse paradigma ou arquitetura de aplicações.

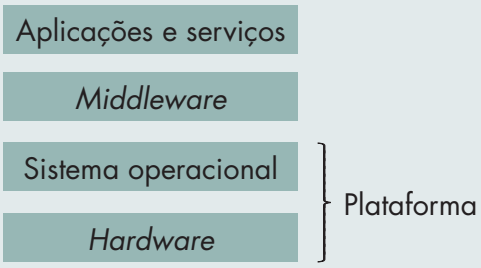
1.1 *Web Services*

Poderíamos muito bem começar este capítulo mencionando que os *Web Services* são a sétima maravilha do mundo e que todos os seus problemas estariam resolvidos a partir do momento em que você passasse a dominar os conceitos

ligados a essa tecnologia. Muito disso é verdade, já que os *Web Services* são interessantes, considerando as vantagens que tal arquitetura fornece.

Mas é necessário que entendamos os conceitos e princípios tecnológicos para uma melhor utilização desse padrão. Poderíamos começar nosso capítulo mencionando que os conceitos trabalhados pelos *Web Services* não são novos, já que remetem aos antigos sistemas distribuídos que fornecem o acesso a diversas funcionalidades por meio de protocolos e APIs. Entre diversos sistemas, podemos citar o CORBA, DCOM, SPREAD e JAVA/RMI que também são conhecidos como *middleware*, pois consistem em uma camada responsável por isolar características diferentes como arquitetura de máquinas e rede, bem como **sistemas operacionais das aplicações**, conforme é apresentado na figura 1.

Figura 1 *Camadas de software.*



Fonte: Chappell e Jewell (2002).

Portanto, quando mencionamos a interoperabilidade, estamos, consequentemente, citando os sistemas distribuídos e suas características trabalhadas. Claro que, no decorrer dos anos, tais serviços foram se aperfeiçoando e sendo trabalhados de uma forma mais natural e simples, já que foram introduzidos conceitos chave para possibilitar sua utilização na *web* (GALBRAITH e outros, 2002).

Se repararmos na figura 1, é interessante verificar que a camada de *middleware* é a reponsável por garantir o funcionamento de aplicações e serviços, ou seja, ela é a base para nossa arquitetura voltada à disponibilização de serviços por meio de um item chave que consiste na comunicação entre as partes. Os protocolos e APIs citadas eram interessantes em um ambiente conhecido como uma rede de computadores de uma pequena empresa. Mas agora imagine como seria disponibilizar funcionalidades em uma rede como a internet? No caso, a programação com CORBA e JAVA/RMI, por exemplo, são naturalmente complexas, o que fez com que programadores e responsáveis por TI considerem sempre outras perspectivas e soluções para disponibilizar suas funcionalidades e aplicações, sendo utilizadas geralmente em último caso, quando as características do *software* requerem tais protocolos e APIs, sendo considerados para isso os *Web Services*.

Assim, com certeza, uma das maiores vantagens dos *Web Services* está em sua disponibilidade e características que tornam possível utilizar a internet, bem como

sua neutralidade de plataforma. O segredo para comunicação e interoperabilidade dos *Web Services*, fazendo com que os ambientes e os sistemas se tornem heterogêneos, consiste na linguagem de marcação XML (DEITEL; DEITEL; NIETO, 2003). Então vamos conhecer e trabalhar com os documentos que podem ser criados com a XML e que serão importantes para o restante de nossa disciplina.

1.1.1 XML (*eXtensible Markup Language*)

A XML é uma linguagem de marcação criada e padronizada pelo consórcio W3C (*World Wide Web Consortium*), em 1996, que combina as potencialidades e a extensibilidade de sua linguagem-mãe, a SGML (*Standard Generalized Markup Language*), padrão para armazenamento e intercâmbio de informações e dados em todo o mundo, que foi adotado em 1986 (NEWCOMER, 2002).

Uma linguagem de marcação tem como principal função descrever outras linguagens. Logo uma das principais características da XML é também fornecer suporte para definir outras linguagens ou protocolos trabalhando com *tags* para identificar suas estruturas de dados. Entre essas linguagens, pode ser citada a RDF (*Resouce Description Framework*), muito utilizada na *web* semântica. Para nossa disciplina, a XML é reponsável por definir a SOAP (*Simple Object Access Protocol*), bem como o WSDL (*Web Service Definition Language*) e o UDDI (*Universal Description, Discovery and Integration*), que consideraremos em capítulos futuros. A XML é geralmente utilizada como uma ferramenta de intercâmbio de informações entre diferentes aplicações.

As aplicações utilizam a **XML** devido às características que a diferenciam das demais. Segundo Chappell e Jewell (2002), dessas **características**, podemos citar, como exemplo:

- os arquivos XML, que têm uma estrutura rígida, ou seja, pré-estabelecida;
- opcionalmente, mas recomendado, uma DTD (*Document Type Definition*), que define a estrutura do arquivo XML;
- opcionalmente, informações de folha de estilo que definem como os dados serão apresentados e formatados;
- Parser XML, que é usado para manipulação dos arquivos XML.

A XML tem uma linguagem considerada neutra que pode ser utilizada para representar dados, além de fornecer um suporte corporativo que garante que um maior número de novas tecnologias, em sua maioria *web*, forneçam uma nova forma de disponibilizar serviços nos próximos anos com a utilização de metadados. Quando combinadas, essas características da linguagem XML possibilitam a intergração e a interoperabilidade de programas seguindo o modelo dos *Web Services* (CHAPPELL; JEWELL, 2002).

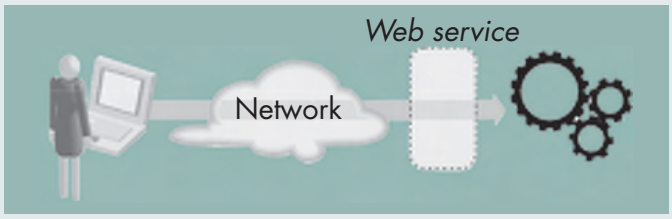
No próximo capítulo, detalharemos mais os conceitos relacionados à confecção de documentos XML e outros correlatos citados anterioremente. Mas o que

realmente nos importa momentaneamente é que você saiba que essa linguagem de marcação fornece e trabalha como a camada de *middleware* desempenhada até então pelos protocolos e APIs dos sistemas distribuídos. Pode se dizer, então, que a linguagem de marcação XML consiste no *core* dos *Web Services*, já que consiste no pilar para as tecnologias que estão envolvidas na disponibilização de serviços como os protocolos SOAP, WSDL e UDDI (ABINADER, 2006).

Uma integração dinâmica de sistemas e serviços requererá, naturalmente, a combinação e envolvimento do SOAP, WSDL e UDDI para prover tais condições, mantendo toda a infraestrutura de negócios ativa. Combinadas, essas tecnologias tornam o *Web Service* revolucionário porque consistem na primeira padronização de tecnologias para fornecer serviços dinâmicos na *web*. Como já mencionado, no passado, existiam tecnologias que proviam características semelhantes ao SOAP, WSDL e UDDI em diversas linguagens, mas deixavam a desejar quanto ao suporte coorporativo, sendo limitadas quanto ao alcance de seus serviços, bem como quanto a sua flexibilidade, já que não trabalhavam com uma linguagem flexível como a XML como camada de *middleware* entre a plataforma e as interfaces das aplicações.

Então, resumidamente, poderíamos dizer que a definição para os *Web Services* consiste em uma rede acessível de funcionalidades de aplicações, também conhecidas como serviços, construídas com uso de tecnologias padrão para a internet. Uma noção básica da estrutura do processo de interoperabilidade utilizando os *Web Services* pode ser observada na figura 2.

Figura 2 *Estrutura padrão de interoperabilidade com Web Services.*

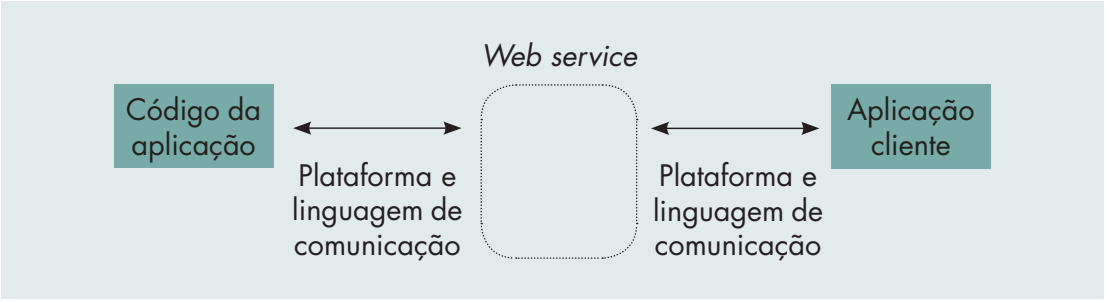


Fonte: Chappell e Jewell (2002).

Conforme pode ser observado na figura 2, os *Web Services* consistem em uma interface posicionada entre o código da aplicação, que está disponível para os usuários, e o código dos serviços disponibilizados, acessados por meio de uma rede. Segundo Chappell e Jewell (2002), essa camada de abstração, que chamamos de *Web Service*, é a responsável por separar a plataforma, no caso arquitetura de *hardware* e sistemas operacionais, bem como linguagens de programação específica, de detalhes de como o código de uma aplicação funciona ou como foi desenvolvida. Essa camada de padronização significa que qualquer linguagem de programação que suporte um *Web Service* pode acessar

as funcionalidades de uma aplicação, as quais, geralmente, estão disponíveis por meio da internet e de documentos HTML, conforme é apresentado na figura 3.

Figura 3 *Abstração de um Web Service entre cliente e funcionalidades.*



Fonte: Chappell e Jewell (2002).

Poderíamos, ainda que grosseiramente, dizer que as páginas e os *web* sítios que nós acessamos consistem em *Web Services*, que fornecem serviços e aplicações como publicação, gerenciamento, pesquisa e apresentação de conteúdo, que são acessados por meio do protocolo padrão HTTP (MENDES, 2004). Aplicações clientes como os *browsers* e ferramentas P2P fazem o uso exatamente de padrões semelhantes aos protocolos HTTP e HTML para possibilitar a interação com serviços para desempenhar tarefas.

Devido a essa abstração fornecida por interfaces padronizadas, independente de linguagens de programação como Java e C, ou mesmo sistemas operacionais como o Windows ou Linux, os *Web Services* permitem o cruzamento de plataformas possibilitando a interoperabilidade, pois torna tais detalhes de arquitetura e de sistema operacional irrelevantes.

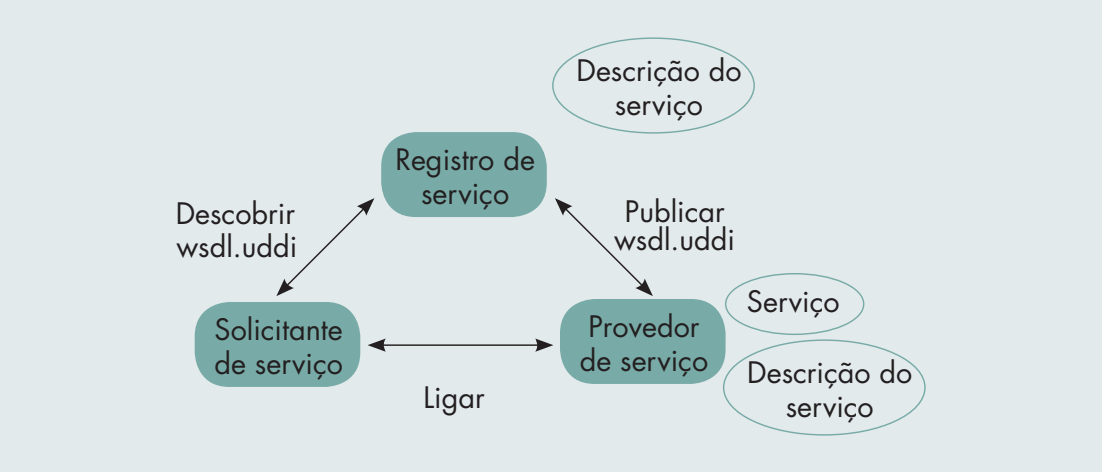
Assim a interoperabilidade é a chave central dos benefícios obtidos com a implementação dos *Web Services*. Uma prova clara do potencial dos *Web Services* é a possibilidade de interoperabilidade entre soluções Java e baseadas no Microsoft Windows, como C#, o que naturalmente é algo complexo, sendo uma camada de intercâmbio que minimiza as diferenças existentes entre ambas as soluções e linguagens de programação (GALBRAITH e outros, 2002). Para um melhor entendimento das características e dos benefícios fornecidos pelos *Web Services*, é interessante que sua arquitetura seja conhecida. Então, vamos a ela?

1.2 Arquitetura dos *Web Services*

Os serviços fornecidos por meio da interoperabilidade de aplicações só são possíveis devido à arquitetura formulada e adotada para os *Web Services*. A arquitetura dos *Web Services* descreve três pilares que são responsáveis por tratar aspectos desde oferta, descoberta e utilização dos serviços disponibilizados, isso por meio da utilização dos protocolos baseados em XML para mensagens, transporte e localização dos serviços. A Figura 4 apresenta a arquitetura por trás

do *Web Service* e descreve o comportamento e as etapas para a utilização dos serviços por ele disponibilizados.

Figura 4 *Abstração de um Web Service entre cliente e funcionalidades.*



Fonte: Chappell e Jewell (2002).

Conforme apresentado na figura 4, as interações entre as aplicações clientes e os serviços fornecidos dependem dos três pilares da abstração que consistem em um Provedor de Serviços, que é o responsável por disponibilizar as funcionalidades que, a qualquer momento, podem ser acessadas por um Solicitante de Serviços, no caso o cliente, que, por meio de mensagens, em sua maioria SOAP, faz uso dos serviços publicados em um Registro de Serviço, no qual podem ser consultadas as diversas funcionalidades publicadas pelos Provedores de Serviço.

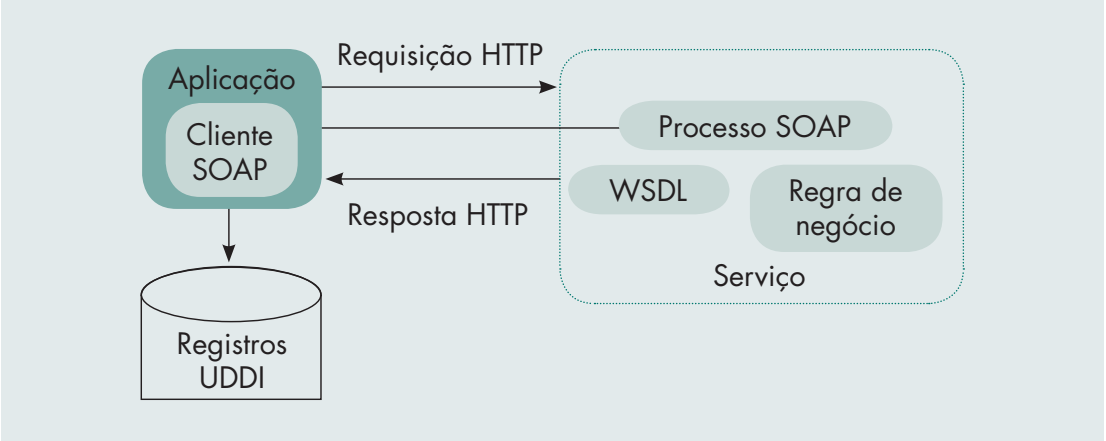
Todos os serviços citados na arquitetura dos *Web Services* apresentados na figura 4 utilizam o padrão XML para a comunicação e a troca de dados. Conforme Chappell e Jewell (2002), três **tecnologias** são extremamente utilizadas nos ***Web Services***. Vejamos quais são elas.

- **WSDL** (*Web Services* Description Language): responsável por descrever os serviços que estão disponíveis.
- **SOAP** (*Simple Object Access Protocol*): protocolo responsável por publicar, localizar e invocar *Web Services* que estão disponíveis nos registros UDDI.
- **UDDI** (*Universal Description, Discovery and Integration*): registro que pode ser acessado por clientes que localizam os serviços a serem utilizados.

Como podemos reparar, cada uma das tecnologias citadas é a responsável por uma das atividades da arquitetura dos *Web Services*. Nos próximos capítulos, consideraremos cada uma dessas tecnologias mais detalhadamente. A

figura 5 apresenta a interação simples entre tais tecnologias baseadas na arquitetura apresentada na figura 4.

Figura 5 *Interação simples entre as tecnologias de um Web Service.*



Fonte: Chappell e Jewell (2002).

O relacionamento entre todas as tecnologias que compõem os *Web Services*, no caso SOAP, WSDL e UDDI, apresentadas na figura 5, pode ser descrito observando a seguinte cronologia: uma aplicação cliente que necessita de um serviço localiza outra aplicação, por meio do protocolo HTTP disponível na *web*. Repare, ainda na figura 5, que os clientes, por meio do protocolo SOAP, consultam um registro UDDI para obter o identificador, nome, categoria e especificações de suporte para utilização das funcionalidades. Com isso, o cliente pode obter de cada serviço localizado informações por meio de documentos WSDL, que contém informações de como contatar o *Web Service* e o formato das mensagens de requisição utilizando um esquema XML. Assim os clientes criam mensagens SOAP que obedecem ao esquema definido no documento WSDL e então são enviadas as requisições para o *host* definido nos registros UDDI, no caso o local no qual os serviços estão disponibilizados.

Saiba mais

Diversos serviços utilizados em sítios na *web* já fazem uso de tecnologias como os *Web Services*, o que possibilita que funcionalidades sejam acessadas e suas características sejam transparentes. Exemplos de tais aplicações são os serviços Google Adsense e Google Maps, que podem ser incorporados a qualquer *web* sítio. Ambas fazem o uso de mensagens baseadas na linguagem XML, que são associadas aos documentos que serão apresentados para os clientes em *browsers*, independentemente das tecnologias adotadas. Você pode acessar mais informações sobre essas tecnologias por meio do endereço eletrônico da Google.

Porém, antes da execução de todo o processo de um *Web Service*, é necessário que possamos acessá-los, e isso é extremamente dependente de tecnologias e técnicas que veremos a seguir.

1.3 Acesso aos *Web Services*

Anteriormente, descrevemos que uma série de passos é necessária para que seja estabelecida uma conexão entre cliente e serviços. Logo fica nítida a importância do processo de comunicação para os *Web Services*. Existem duas maneiras de realizar a **comunicação** entre as partes envolvidas em um *Web Service*, o RPC (*Remote Procedure Calls*) e por meio de mensagens (DAUM; MERTEN, 2002). As principais diferenças entre essas duas tecnologias de comunicação estão tanto na abordagem quanto na arquitetura definida no processo de implementação. O protocolo de comunicação para as duas tecnologias é o mesmo, o SOAP, que suporta ambas.

Diversas tecnologias e linguagens de programação dão suporte à utilização do RPC e das mensagens como o Microsoft SOAP Toolkit, .Net *Web Services*, .NET Remoting, DCOM, Java *Web Services*, Java RPC e Java Message Service. Mas em termos de utilização, como se comportam essas duas formas de se comunicar? Entenderemos, a seguir, como cada uma dessas importantes partes do *Web Service* trabalha.

A ideia por trás do RPC é que o cliente comece a pensar em termos de invocar procedimentos de maneira remota, ou seja, em um servidor de serviços. Ok, até então algo natural, já que essa é a ideia natural dos *Web Services*. Porém, em termos de implementação, isso significa a instanciação de objetos remotos e invocação de métodos e/ou propriedades disponibilizadas por esses objetos. Logo isso faz com que concentremos nossos esforços sempre na camada de objetos e suas interfaces, utilizando, assim, seus atributos e métodos públicos por meio de parâmetros para acesso aos serviços de um *Web Service*.

Por sua vez, as mensagens são tipicamente comuns no processo de comunicação de qualquer procedimento computacional, como no caso dos sistemas distribuídos. Nesse tipo de procedimento de comunicação, uma aplicação cliente define seus procedimentos e então cria mensagens que são enviadas a um servidor contendo todas as diretivas e dados necessários para a execução de uma tarefa (DAUM; MERTEN, 2002). Do lado do servidor, quando a mensagem do cliente é recebida, é realizado o processamento do serviço requisitado e, como resultado desse processo, é criada uma mensagem de resposta que é enviada para o cliente que solicitou o serviço.

Logo o processo de utilização de mensagens está obviamente concentrado na definição de regras que são respeitadas por clientes e servidores nas trocas das mensagens para a utilização dos serviços remotos. Nesse caso, ao definirmos a utilização de mensagens para a comunicação entre as partes, temos uma característica que torna esse processo menos atraente, que consiste em um

menor acoplamento entre o cliente e o servidor de serviço que utiliza o RPC. Isso acontece porque no processo de mensagem cada parte que não se relaciona tem um canal de comunicação direto, já que cada um cria suas mensagens assincronamente, ou seja, enquanto o cliente cria as mensagens, o servidor não está realizando nenhuma tarefa para aquele cliente. O mesmo processo acontece com o cliente enquanto aguarda a resposta do servidor a suas mensagens.

Outra característica interessante dos sistemas RPC é que muitos deles tentam, de alguma maneira, prover uma noção transparente para os serviços disponibilizados, tratam seus objetos remotos e interfaces como se eles estivessem sendo executados localmente e escondem o que deve ser enviado, fazendo com que o cliente não tenha de se preocupar com o fato de o servidor de objetos, ou no caso, que os objetos estejam em outra máquina (CHAPPELL; JEWELL, 2002).

Poderíamos dizer que, nesse caso, as implementações de soluções que utilizam RPC para *Web Services* são idênticas, seja para os serviços fornecidos local ou remotamente, considerando obviamente apenas o *host* definido a ser acessado. Na contramão do RPC, os sistemas que se utilizam de mensagens permitem ter controle sobre o que é apresentado sem que a outra extremidade do par de comunicação seja conhecida, dando maior segurança e transparência ao processo.

Assim finalizamos o primeiro capítulo. Nele, estudamos os principais conceitos relacionados à interoperabilidade. Além disso, você foi apresentado a uma nova forma de distribuir suas aplicações, o paradigma SOA (*Service-Oriented Architecture*), que consiste em uma arquitetura de *software* baseada no conceito chave de que as aplicações deixam de ser vistas como um todo e passam a considerar as suas funcionalidades como serviços específicos que podem ser utilizados por diversas aplicações distribuídas, o que torna o processo de reutilização mais abrangente. Vimos protocolos e tecnologias que se utilizam da linguagem de marcação XML para a definição de processos importantes como a troca de mensagens, divulgação e utilização dos serviços, no caso o protocolo SOAP, os registros UDDI e os documentos WSDL. Por fim, consideramos um aspecto vital para os *Web Services*: a comunicação entre clientes e servidores por meio do uso das tecnologias como RPC e mensagens para a solicitação e execução das aplicações.

No próximo capítulo, conheceremos os conceitos básicos da tecnologia XML, bem como a construção dos demais protocolos e tecnologias que possibilitam a comunicação e a disponibilização de serviços pelos *Web Services*.

Referências

ABINADER, Jorge Abílio. ***Web Services* em Java**. Rio de Janeiro: Brasport, 2006.

CHAPPEL, David A.; JEWELL, Tyler. **Java *Web Services***. Sebastopol, USA: O'REILLY, 2002.

DAUM, Berthold; MERTEN, Udo. **Arquitetura de sistemas com XML**. Rio de Janeiro: Campus, 2002.

DEITEL, Harvey M.; DEITEL, Paul J.; NIETO, Lin. **XML**: como programar. Porto Alegre: Bookman, 2003.

GALBRAITH et al. **Profissional Java *Web Services***. Rio de Janeiro: Alta Books, 2002.

MENDES, Antônio. **Programando com XML**. Rio de Janeiro: Campus, 2004.

NEWCOMER, Eric. **Understanding *Web Services***: XML, WSDL, SOAP and UDDI. Indianapolis, USA: Addison-Wesley Professional, 2002.

Anotações

Construindo documentos XML

2

Introdução

Para o acompanhamento deste capítulo, é importante que os conceitos apresentados no capítulo 1 tenham sido assimilados por você. Esse conhecimento é necessário, já que a linguagem XML, que é o tema deste capítulo, é a base das demais tecnologias que são aplicadas e utilizadas na criação dos *Web Services*, bem como na implementaçao da interoperabilidade entre as aplicações.

Na sua maioria, a troca de mensagens e dados entre as aplicações utilizam o padrão XML. Além disso, é interessante que você já tenha algum conhecimento sobre o trabalho com *tags*, o que já foi apresentado a você na disciplina de programação com foco para *web*: assim como a linguagem HTML, a XML trabalha com o conceito de marcações. A compreensão desses conteúdos auxiliarão você a compreender os fundamentos da linguagem XML e criar documentos de metadados baseados em XML.

Como pode ser observado por você no capítulo anterior, seria praticamente impossível comentarmos sobre interoperabilidade de aplicações sem que, em algum momento, nos deparássemos com a linguagem de marcação XML. Vimos que diversos protocolos e tecnologias estão baseados nessa linguagem que é conhecida como um padrão para o intercâmbio de dados. Entre essas tecnologias e protocolos, estão o SOAP, o WSDL e o UDDI que, conforme vimos, são utilizados para possibilitar o trabalho dos *Web Services*.

Quando falamos da linguagem XML, não há como deixar de falar de sua estrutura que, apesar de flexível e bem definida, tornou o XML uma das tecnologias mais utilizadas em qualquer que seja a aplicação ou mesmo armazenamento de dados computacionalmente. Neste capítulo, verificaremos o processo de construção de documentos XML e conheceremos sua estruturação hierárquica, bem como outros documentos que auxiliam na definição e na apresentação dos dados mantidos em um arquivo XML. Então, vamos ao que interessa?

2.1 Estrutura dos documentos XML

No início do capítulo, mencionamos que os documentos XML consistem em um documento com estrutura bem definida e extremamente flexível. A *web* é uma rede mundial por meio da qual milhões de pessoas se comunicam e interagem por intermédio de diversas aplicações. Com o passar dos anos, a linguagem

HTML, coirmã da XML, já que ambas são frutos da SGML, passou a ser o padrão para a criação de documentos para a *web*. Porém o HTML tem sérias limitações, sendo apenas um padrão para a representação visual, ou seja, apresentação de conteúdo (NEWCOMER, 2002).

A XML surgiu, em 1996, com o intuito de fornecer um padrão para codificar conteúdo de maneira a possibilitar a criação de estruturas de dados mais flexíveis. Para tanto, os documentos XML fazem uso de regras, ou de um conjunto delas, para garantir uma definição formal de estrutura e dados, o que torna a XML computacionalmente atraente. Com o passar dos anos, todos os *browsers* disponíveis no mercado passaram a dar suporte à apresentação dos documentos (NEWCOMER, 2002).

Segundo Abinader (2006), os documentos XML podem ser disponibilizados e utilizados nas mais diversas aplicações, demonstrando o porquê de sua importância na interoperabilidade de aplicações. A XML possibilita que sejam criadas soluções tanto do lado cliente quanto do servidor. Por exemplo, no lado cliente, os documentos XML podem ser utilizados para a apresentação de conteúdo personalizado, o que está entre as maiores limitações da HTML. Tais limitações foram supridas com a criação das folhas de estilo que são baseadas na XML.

Do lado dos servidores, seu impacto é mais claro, e isso é obviamente mais nítido, já que os documentos que obedecem a esse padrão consistem no core dos *Web Services* que, no caso, são as mensagens. Esse sistema de mensagens é a possibilidade de intercâmbio de dados entre aplicações e computadores, no qual é estabelecido um formato e regras claras para os pares.

Outra clara e forte fonte de utilização dos documentos XML está no armazenamento de dados, em que são utilizados para criar bancos de dados e armazenar os dados seguindo uma estrutura de árvore. Nessa estrutura os dados podem ser acessados de seu nó raiz e assim navegar pelo documento, tendo acesso aos dados contidos nas *tags* do documento. Graças à aplicação de regras rigorosas de boa formação do XML, é possível se assegurar que qualquer analisador XML disponível tenha a capacidade de ler e compreender seu conteúdo (GALBRAITH e outros, 2002). Apesar de esse tipo de persistência apresentar suas vantagens, é importante mencionar que obviamente essa não é a maneira mais eficiente de se persistirem dados.

Assim, para a formatação dos documentos XML, é necessário que algumas regras sejam seguidas. Na próxima seção, trabalharemos com essas regras e criaremos os primeiros documentos.

2.1.1 Regras dos documentos XML

Uma das características da linguagem HTML é que, para a apresentação de conteúdo, não é necessário que seus documentos tenham uma estruturação formal, ou seja, não importa o quanto seja estranha ou ruim uma formatação HTML, ela será sempre apresentada nos *browsers*. Na contramão, os analisadores e os processadores de documentos XML não permitem que ocorra uma

má formação em sua estrutura, o que gerará uma mensagem de erro impossibilitando sua utilização (DEITEL; DEITEL; NIETO, 2003).

Deitel, Deitel e Nieto (2003) mencionam que isso auxilia no processo de garantir que tais documentos possam ser interpretados da mesma maneira, independente do processador XML e/ou aplicação, ou seja, seguindo uma mesma padronização. Para tanto, existem algumas regras. Um documento que segue tais regras é considerado como bem formado. Segundo Deitel, Deitel e Nieto (2003), um documento XML bem formado é aquele que:

- obedece à sintaxe XML, sem que sejam utilizadas marcações incorretas, ou incompletas;
- não utiliza caracteres indevidos ou especiais dentro do arquivo;
- nenhum atributo deve aparecer mais de uma vez em uma marcação.

O texto dos documentos XML pode ser classificado em duas categorias: marcação e dados de caracteres. A primeira consiste em qualquer coisa que esteja entre os sinais "<" e ">", assim como as *tags* HTML. Todo o restante é considerado como dados de caracteres. Vamos à estrutura básica de nosso primeiro documento XML? Observe a semântica utilizada no quadro 1.

Quadro 1 *Estrutura básica de um documento XML bem formado.*

```
1 <?xml version="1.0"?>
2 <lista-veiculo>
3  <veiculo>
4  <nome>FUSCA</nome>
5  <tipo>PASSEIO</tipo>
6  <marca>VOLKSWAGEM</marca>
7  </veiculo>
8  <veiculo>
9  <nome>CORSA</nome>
10 <tipo>PASSEIO</tipo>
11 <marca>CHEVROLET</marca>
12 </veiculo>
13 </lista-veiculo>
```

Conforme pode ser observado no quadro 1, a primeira coisa a ser feita na confecção de um documento XML é a utilização da marcação que auxilia na identificação do arquivo como sendo uma estrutura XML. Logo após a declaração XML, são

definidos os **elementos ou nós**. Os elementos são a forma mais comum de marcação utilizada nos documentos XML. Note que eles seguem uma estrutura padrão – abertas e fechadas e obedecendo às regras XML para documentos bem formados. Os elementos nome, tipo e marca são considerados filhos do elemento veículo.

Como mencionado, o que não é uma marcação consiste nos dados, então tudo o que se encontra entre as *tags* de abertura e fechamento são dados ou conteúdo do elemento e podem ser acessados utilizando *parsers* para navegação na estrutura. Abordaremos mais sobre os *parsers* no próximo capítulo. Salve o arquivo, por exemplo, como "listaVeiculo.xml" retirando os números ao lado para identificação das linhas e verifique sua apresentação em qualquer *browser*. Porém isso não é tudo quando falamos da XML.

Um documento bem formado não consiste naturalmente em um arquivo válido. E aqui surge outro importante detalhe da utilização desse padrão: a validação. A validação dos documentos é muito importante, pois isso é necessário para que os analisadores XML, que são módulos de *software* e geralmente são incorporados a outras aplicações, possibilitem o acesso a dados armazenados nos documentos XML.

Os processadores XML podem ser tanto com validação quanto sem validação. Caso seja assumida tal característica, é necessária a utilização de um novo componente que é o DTD (Document Type Definitions – definição de tipos de documento).

O DTD é um arquivo que trabalha em conjunto com os documentos XML e é uma estrutura utilizada para fornecer explicitamente um conjunto de regras para a construção dos XML, definindo quais elementos podem e devem ser utilizados. Assim é possível imaginar o DTD como um guia, no qual são definidas as diretrizes para a criação dos documentos XML (DEITEL; DEITEL; NIETO, 2003).

Um documento XML que esteja rigorosamente de acordo com as regras estabelecidas no DTD para o qual ele foi escrito, bem como obedece às regras da XML, é considerado um documento bem formado e válido. Assim como fizemos com os documentos XML, observemos nosso primeiro arquivo DTD que é apresentado no quadro 2.

Quadro 2 *Estrutura de um arquivo DTD.*

```
1 <!ELEMENT lista-veiculo (veiculo)*>
2 <!ELEMENT veiculo (nome, tipo, marca)>
3 <!ELEMENT nome (#PCDATA)>
4 <!ELEMENT tipo (#PCDATA)>
5 <!ELEMENT marca (#PCDATA)>
```

O arquivo apresentado deve ser obrigatoriamente salvo com o formato ".dtd", como, por exemplo, "lista.dtd". No quadro 2, é utilizado o tipo mais básico de declaração em um DTD que é a declaração <!ELEMENT>. O formato padrão de um elemento consiste na declaração que foi apresentada, no quadro 2, como, por exemplo, o que foi feito na declaração do elemento nome que utiliza a regra #PCDATA na linha 3.

As regras são as responsáveis por definir e especificar o que pode ser utilizado no conteúdo do elemento definido e que será utilizado no arquivo XML. Segundo Deitel, Deitel e Nieto (2003), você tem duas opções para a definição do conteúdo básico:

- **#PCDATA**: utilizado para declarar um elemento que aceita dados de caracteres analisados;
- **EMPTY**: utilizado para declarar um elemento que não conterá nenhum dado.

Além disso, para a definição de elementos em um DTD, segundo Deitel, Deitel e Nieto (2003), é necessário que sigamos algumas regras:

- os nomes não devem conter "<" ou ">";
- os nomes dos elementos devem começar obrigatoriamente com letra ou sublinhado;
- os nomes não podem começar com a *string* XML;
- não usar dois pontos, eles são proibidos.

Como foi apresentado no quadro 2, uma prática comum consiste em aninhar elementos filhos dentro de outros elementos, considerados pais, como apresentado na linha 1 e 2. Assim você pode especificar que tipo de elementos deve aparecer dentro de algum elemento, como, por exemplo, em no documento XML definido no quadro 1. É de extrema importância que sejam obedecidas a ordem na qual foram definidos os elementos nos arquivos DTD dentro dos documentos XML.

Ainda observando o quadro 2, repare que, na linha 1, existe a utilização do símbolo "*". Os símbolos associados à declaração dos elementos nos arquivos DTD são utilizados para descrever regras que tornem nossos arquivos XML mais flexíveis e condizentes com o que será produzido. Esses símbolos são conhecidos como operadores de ocorrência e descrevem quantas vezes um determinado elemento pode aparecer ou definir, entre dois elementos, qual será utilizado dentro do arquivo XML validado pelo DTD criado (CHAPPELL; JEWELL, 2002). Para essa definição, podem ser utilizados quatro tipos de operadores, como nos é apresentado no quadro 3.

Quadro 3 *Operadores de ocorrência.*

SÍMBOLO	DESCRIÇÃO
?	O elemento deve ocorrer zero ou uma vez dentro do arquivo XML validado pelo DTD.
+	O elemento deve ocorrer uma ou mais vezes dentro do arquivo XML validado pelo DTD.

SÍMBOLO	DESCRIÇÃO
*	O elemento pode ocorrer qualquer número de vezes ou nenhuma dentro do arquivo XML validado pelo DTD.
|	Especifica que é preciso fazer uma escolha entre dois elementos a serem utilizados dentro do XML validado pelo DTD.

Fonte: Chappell e Jewell (2002).

Podemos concluir que, por meio dos arquivos DTD, são produzidas regras que devem ser respeitadas por qualquer XML que se proponha a utilizá-lo. Ou seja, se o quadro 1, na qual está definido o documento XML, estiver obedecendo ao DTD criado, no quadro 2, poderíamos criar quantos nós fossem necessários para representar os dados de veículos, mas apenas um elemento "lista-veiculo". Para que um documento XML passe a obedecer, ou seja, se torne válido perante um DTD, é necessário que tenha definição do arquivo, conforme é apresentado no quadro 4.

Quadro 4 *Documento XML bem formado e validado por um arquivo DTD.*

```
1 <?xml version="1.0">
2 <!DOCTYPE lista-veiculo SYSTEM "lista.dtd">
3 <lista-veiculo>
4  <veiculo>
5  <nome>FUSCA</nome>
6  <tipo>PASSEIO</tipo>
7  <marca>VOLKSWAGEM</marca>
8  </veiculo>
9  <veiculo>
10 <nome>CORSA</nome>
11 <tipo>PASSEIO</tipo>
12 <marca>CHEVROLET</marca>
13 </veiculo>
14 </lista-veiculo>
```

Para que nosso documento XML seja corretamente apresentado em um *browser*, é necessário que o caminho do arquivo DTD esteja correto, como definido na linha 2. Em nosso caso, ele deve estar no mesmo diretório do documento XML, como é demonstrado na linha 2. Repare que todos os nós utilizados no documento XML foram definidos no DTD, caso contrário ocorreria um erro e o *browser* não apresentaria o XML. Aqui surgem novos conceitos que podem ser utilizados para representar dados nos documentos XML, no caso os **atributos**.

Os elementos ou nós em arquivos XML que nos foram apresentados anteriormente não são a única maneira de manter dados, outra opção consiste nos atributos. Eles são uma estrutura que pode ser utilizada para associar pares "nome-valor" nos elementos. Os atributos são simples fontes de informação adicional que estão obrigatoriamente associadas a um elemento.

Quadro 5 *Documento XML com elementos utilizando atributos.*

```
1 <?xml version="1.0">
2 <!DOCTYPE lista-veiculo SYSTEM "lista.dtd">
3 <lista-veiculo>
4  <veiculo concertado="nao">
5  <nome>FUSCA</nome>
6  <tipo>PASSEIO</tipo>
7  <marca>VOLKSWAGEM</marca>
8  </veiculo>
9  <veiculo concertado="sim">
10 <nome>CORSA</nome>
11 <tipo>PASSEIO</tipo>
12 <marca>CHEVROLET</marca>
13 </veiculo>
14 </lista-veiculo>
```

É possível observar a utilização dos atributos nas linhas 4 e 9. Repare que uma importante função dos atributos é permitir diferenciar os elementos filhos em um documento XML, como, por exemplo, considerando o quadro 5, seria extremamente simples identificar os veículos consertados e os não. Assim como feito com os elementos dos documentos XML, nos quais é possível defini-los em um arquivo DTD, o mesmo ocorre com os atributos de maneira a tornar o XML válido. Para isso, é necessário utilizar um novo formato na declaração de um atributo em um DTD. Então vamos a ela. Observe o quadro 6, no qual a sintaxe é apresentada.

Quadro 6 *Sintaxe para a declaração de um atributo em um DTD.*

```
<!ATTLIST elementoXmlAlvo nomeAtributo tipo valorPadrao>
```

Repare, no quadro 6, que é necessário que definamos qual o elemento do arquivo XML terá um atributo. Logo após a definição do nome do atributo, é definido o tipo de dados que podem ser trabalhados nesse atributo. Para finalizar a sintaxe da declaração dos atributos, deve ser definido o valor padrão que pode assumir três valores: #REQUIRED, #IMPLIED e #FIXED. Assim se aplicarmos a declaração de um atributo em nosso arquivo DTD, pode-se obter algo semelhante ao que é apresentado no quadro 7.

Quadro 7 *Arquivo DTD e a declaração de um atributo.*

```
1 <!ELEMENT lista-veiculo (veiculo)*>
2 <!ELEMENT veiculo (nome, tipo, marca)>
3 <!ATTLIST veiculo concertado CDATA #REQUIRED>
4 <!ELEMENT nome (#PCDATA)>
5 <!ELEMENT tipo (#PCDATA)>
6 <!ELEMENT marca (#PCDATA)>
```

Muitos autores citam a utilização dos atributos como sendo uma forma de manter dados e substituir os elementos tradicionais dos documentos XML, no qual temos uma marcação de abertura, os dados e outra para o fechamento. Porém essa não é a função principal dos atributos, e sim o auxílio e a definição de dados adicionais. Mas nada impede que você trabalhe dessa maneira.

Segundo Deitel, Deitel e Nieto (2003), existem nove tipos de atributos que podem ser utilizados em documentos XML. Eles estão classificados em três **categorias** que descrevem sua utilização, sendo:

- ***string***: consiste em qualquer *string* de caracteres válidos e é utilizado como o tipo CDATA;
- ***tokenized***: é utilizado para a identificação de forma única elementos dos documentos XML. São utilizados para isso os tipos ID e IDREF;
- **enumerado**: serve para listar possíveis valores que um atributo pode assumir. São utilizados para isso valores contidos dentro de "()" e separados pelo símbolo "|".

Aqui fica a dica para que você altere os atributos, utilizando as três categorias apresentadas. Tais estruturas são úteis para garantir que os documentos XML sejam formais, logo computacionalmente processáveis. Ainda temos outros elementos dos arquivos DTD e dos documentos XML que são importantes e devem ser considerados. Entre eles, temos as declarações de entidades.

Uma **declaração de entidade** consiste em uma marcação no arquivo DTD que possibilita a utilização de referências de caracteres. Ou seja, uma referência de entidade possibilita a substituição de valores dentro de um arquivo XML. Os valores reais estão definidos no arquivo DTD. Sua maior contribuição está no fato de possibilitar que sejam trabalhados caracteres que seriam inválidos em um documento XML. Para isso, é necessário que as manipulações sejam realizadas utilizando, sempre no início, os caracteres "&#". Vamos a um exemplo para que tal afirmação fique mais clara. Observe o quadro 8, na qual definimos novamente o arquivo DTD.

Quadro 8 *Declaração de entidades no arquivo DTD.*

```
1 <!ELEMENT lista-veiculo (veiculo)*>
2 <!ELEMENT veiculo (nome, tipo, marca)>
3 <!ATTLIST veiculo concertado CDATA #REQUIRED>
4 <!ELEMENT nome (#PCDATA)>
5 <!ELEMENT tipo (#PCDATA)>
6 <!ELEMENT marca (#PCDATA)>
7 <!ENTITY element "&#41;">
```

Em nosso XML, basta utilizar a referência da entidade e verificar no *browser* o resultado da inserção da referência de entidade. Então altere a linha apresentada no quadro 9 em seu documento XML na linha especificada do quadro anterior.

Quadro 9 *Utilização de entidades em documentos XML.*

```
...
6  <tipo>PASSEIO &element;</tipo>
...
```

Para finalizarmos o segundo capítulo, conheceremos dois importantes conceitos para os documentos XML: as seções CDATA e os comentários em XML.

As **seções CDATA** consistem em blocos de textos que podem conter dados que não são analisados no momento da apresentação do documento XML. Esse tipo de bloco de texto é muito comum para a inserção de *scripts* que podem ser

úteis no momento da manipulação dos XML. A sintaxe para as seções CDATA é apresentada no quadro 10.

Quadro 10 *Utilização da seção CDATA.*

```
1 <?xml version="1.0">
2 <!DOCTYPE lista-veiculo SYSTEM "lista.dtd">
3 <lista-veiculo>
4  <veiculo concertado="nao">
5  <nome>FUSCA</nome>
6  <tipo>PASSEIO</tipo>
7  <marca>VOLKSWAGEM</marca>
8  </veiculo>
9  <veiculo concertado="sim">
10 <nome>CORSA</nome>
11 <tipo>PASSEIO</tipo>
12 <marca>CHEVROLET</marca>
13 </veiculo>
14 <![CDATA[ isso <>%$#<> nao seria permitido em um
elemento XML ]]>
15 </lista-veiculo>
15
```

Os **comentários em XML** seguem o mesmo padrão utilizado nas páginas HTML. Os comentários são *tags* especiais e podem aparecer em qualquer parte do documento, e as informações contidas neles são desconsideradas pelos analisadores XML. Ou seja, para que você insira algum comentário, basta que você utilize a sintaxe apresentada no quadro 11.

Quadro 11 *Comentários em documentos XML.*

```
<!-- este texto sera considerado um comentario. -->
```

É importante mencionar que os documentos XML são uma poderosa arma para a interoperabilidade entre aplicações, e os conceitos apresentados neste capítulo

são a parte central das perspectivas de um documento XML. Existem diversos conceitos, que ainda podem ser relacionados à linguagem de marcação XML, que podem garantir e enriquecer suas características de flexibilidade e formalidade.

Saiba mais

Outra possibilidade de enriquecer os documentos XML é a utilização dos esquemas XML e das folhas de estilo. Os esquemas XML têm por objetivo algo semelhante ao dos arquivos DTD, ou seja, definir classes de documentos XML. Porém a principal diferença entre esses formatos está no fato de que os esquemas XML dividem os elementos entre complexos e simples, além de possibilitar um maior controle dos dados que serão inseridos nos elementos dos documentos XML. Já as folhas de estilo são extremamente comuns nas páginas HTML. Também conhecidas como arquivos XSL, são utilizadas para especificar a apresentação dos documentos XML que serão lidos. Você obterá mais informações sobre esses padrões ao acessar o endereço eletrônico <http://www.w3.org>.

Portanto, neste capítulo, estudamos os conceitos básicos sobre a criação de documentos XML, bem como os principais elementos que podem ser utilizados para sua criação. Vimos que a XML surgiu com o intuito de fornecer um padrão para codificar conteúdo e possibilitar a criação de estruturas de dados mais flexíveis. Conhecemos as principais regras para a definição dos documentos bem formados, assim como criação de elementos que compõem os documentos XML. Vimos que os arquivos DTD são estruturas utilizadas para fornecer explicitamente um conjunto de regras para a construção dos XML. Além dessas regras, estudamos elementos, atributos, operadores de ocorrência e outros tipos de marcações úteis na formação de documentos XML.

No próximo capítulo, apresentaremos os *parsers* XML para o manuseio de documentos XML por meio das APIs Java. Também mostraremos a estrutura e a arquitetura do SAX e DOM, que possibilitam a interoperabilidade entre as aplicações.

Referências

ABINADER, Jorge Abílio. **Web services em Java**. Rio de Janeiro: Brasport, 2006.

CHAPPEL, David A.; JEWELL, Tyler. **Java *Web Services***. Sebastopol, USA:O'REILLY, 2002.

DEITEL, Harvey M.; DEITEL, Paul J.; NIETO, Lin. **XML**: como programar. Porto Alegre: Bookman, 2003.

GALBRAITH, Bem et. al. **Profissional Java Web services**. Rio de Janeiro: Alta Books, 2002.

NEWCOMER, Eric. **Understanding Web services**: XML, WSDL, SOAP and UDDI. Indianapolis, USA: Addison-Wesley Professional, 2002.

Anotações

Parser XML: SAX e DOM

3

Introdução

Para que você tenha um melhor aproveitamento do que será abordado neste capítulo, é recomendável que os conceitos e os arquivos que foram apresentados no segundo capítulo tenham sido assimilados, uma vez que manipularemos documentos XML utilizando APIs Java que foram projetadas especificamente para a navegação em tais estruturas de dados. Além disso, é muito importante que você tenha conhecimento sobre a utilização da linguagem Java, já que muitos conceitos e definições das linguagens de programação serão consideradas durante a implementação dos programas que abrem espaço para a interoperabilidade e troca de mensagens com documentos XML.

Esperamos que, com a compreensão desse conteúdo, ao final deste capítulo, você seja capaz de entender o funcionamento da APIs SAX e DOM e utilizar métodos e classes das APIs SAX e DOM para a manipulação de documentos XML.

Agora que já conhecemos os documentos XML e arquivos correlatos que, como temos mencionado desde o início, são o segredo da interoperabilidade entre aplicações utilizando os *Web Services*, começaremos a manipular tais documentos. Então imagine, agora, quanta coisa não podemos fazer com os documentos XML. Uma estrutura XML, por exemplo, poderia ser utilizada para representar dados em uma mensagem, como é apresentado no quadro 1.

Quadro 1 *Documento XML descrevendo uma mensagem.*

```
<?xml version="1.0" ?>
<msg>
<para>Mario</para>
<de>Janaina</de>
<corpo>Mensagem entre dois pares! </corpo>
</msg>
```

A partir do documento XML apresentado no quadro 1, poderíamos imaginar uma aplicação que, com a manipulação dos dados que estão contidos na estru-

tura XML, pudesse produzir uma saída para o documento. Neste capítulo, estudaremos as ferramentas que possibilitam o acesso e a manipulação dos dados contidos nos documentos XML. Para tanto, vamos nos servir dos *parsers* XML, em específico com o DOM e a SAX.

3.1 Analisadores sintáticos

Você deve estar se perguntando o que são os *parseres* XML ou analisadores sintáticos, como são mais conhecidos pelos profissionais ligados à tecnologia. Um parser XML é o nome dado a *softwares* responsáveis por realizar uma análise sintática das estruturas XML. Ou seja, um *software* que está embutido em diversos sistemas, *web* ou não, que realiza a leitura de um documento escrito com a linguagem XML, ele verifica se sua sintaxe está correta, observando se não existem caracteres ilegais ou marcações incorretas (DEITEL; DEITEL; NIETO, 2003).

Assim, caso o documento esteja sintaticamente correto, é permitido o acesso do programa ao documento XML para a realização das tarefas a que ele se propõe. Tal situação já pode ter sido observada no capítulo anterior, caso você tenha tentado acessar os documentos XML criados por meio de um *browser*.

Como vimos até agora, as demais tecnologias ligadas à tecnologia XML são bastante limitadas, estão na maioria das vezes ligadas apenas à transformação e apresentação de conteúdo. Um *parser* XML é a mais básica e mais importante ferramenta XML, já que é ele é que faz a camada de intercâmbio entre o documento XML e as aplicações. Para tanto, segundo Deitel, Deitel e Nieto (2003), podemos considerar que tal arquitetura é estruturada em duas partes:

- o *parser* XML é o responsável por lidar com o documento XML;
- a aplicação consome o conteúdo do arquivo por meio do analisador.

Para isso, os analisadores sintáticos utilizam dois tipos de interfaces baseadas em objetos e eventos. Obviamente, cada uma tem suas vantagens e desvantagens. A principal diferença entre elas está na forma como as estruturas dos documentos XML são manipuladas e repassadas para as aplicações.

Os *parsers* XML que utilizam as interfaces baseadas em objetos, explicitamente, constroem uma estrutura em árvore contendo todos os elementos de um documento XML na memória. Essa é provavelmente a mais natural das interfaces para as aplicações, já que refletem naturalmente a estrutura utilizada computacionalmente, bem como o armazenamento das estruturas XML. Isso também é mais conveniente para aplicações que trabalham com documentos XML dos quais não se tem conhecimento sobre sua sintaxe e/ou estrutura (CHAPPELL; JEWELL, 2002).

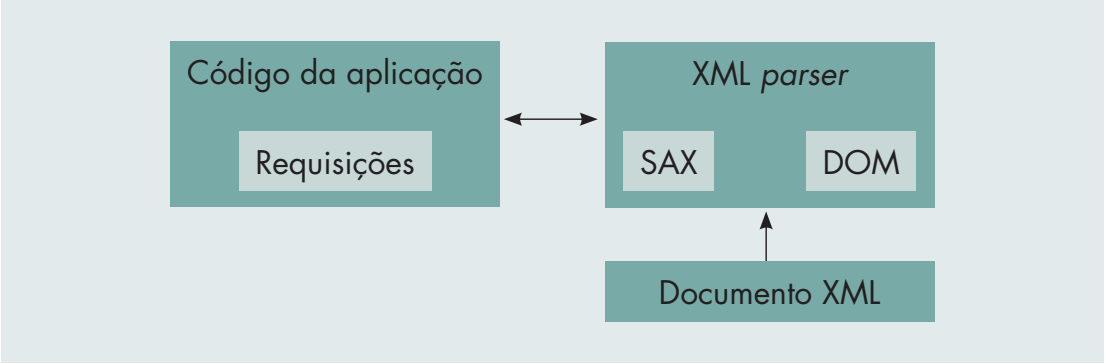
A segunda abordagem para as interfaces utilizadas pelos analisadores sintáticos é a baseada em eventos. Esse tipo de interface naturalmente é a forma mais complexa para a utilização em aplicações, porém bem mais atraentes computacionalmente. Isso se justifica pelo fato de ser uma arquitetura mais limpa, pois

elementos, atributos e demais estruturas de um documento XML são acessadas diretamente, ou seja, diferentemente das interfaces baseadas em objetos, o documento manipulado não constrói uma árvore que é colocada na memória. Isso torna o processo mais leve computacionalmente, porém exige que a aplicação conheça a sintaxe e a estrutura do documento XML que está sendo manipulado. Para isso, existem eventos que notificam a aplicação quando algum desses elementos da estrutura XML é encontrado (CHAPPELL; JEWELL, 2002).

Na prática, ambos os formatos têm sua importância dentro de um contexto em particular, tendo objetivos diferentes. Segundo Newcomer (2002), os analisadores sintáticos baseados em objetos são ideais para manipulação de documentos XML em situações como a apresentação em *browsers*, editores e processadores XSL.

Em contrapartida, os *parsers* XML baseados em eventos são recomendados para aplicações que manipulam dados ou estruturas proprietárias que sigam ou não o formato XML, como, por exemplo, em aplicações que importam documentos XML de banco de dados. Tais tipos de interfaces são mais eficientes, já que não consomem tanta memória. A figura 1 define a arquitetura de manipulação de dados que pode ser assumida para a utilização dos documentos XML.

Figura 1 *Arquitetura dos analisadores sintáticos.*



Conforme foi apresentado na figura 1, duas APIs são constantemente utilizadas pelos analisadores sintáticos: a DOM (*Document Object Model*) e a SAX (*Simple* API *for* XML). Essas APIs serão analisadas nas próximas seções.

3.1.1 DOM

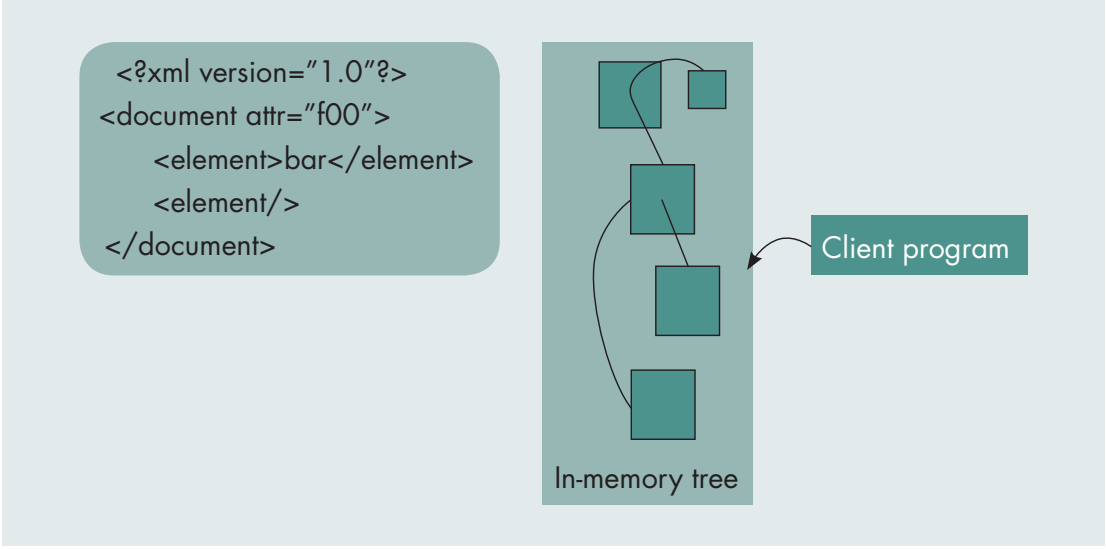
Deitel, Deitel e Nieto (2003) mencionam que originalmente a W3C desenvolveu o DOM para a manipulação de documentos em navegadores. Seu maior feito consistiu em unificar os modelos de objetos para os navegadores Netscape Internet Explorer. Criou uma maior possibilidade de compatibilidade na apresentação de conteúdo e é um marco na guerra travada entre os navegadores, bem como um avanço imensurável na internet, já que até então o que era visível em um navegador não tinha garantia de que seria visível no outro.

A recomendação DOM dá suporte tanto a documentos XML quanto a HTML, bem como a documentos validados por um arquivo DTD. A questão de o DOM

ser a recomendação padrão da W3C garante que tal API é suportada por diversas tecnologias, entre elas a javascript e, em nosso caso, a linguagem Java (DEITEL; DEITEL; NIETO, 2003).

Assim existem APIs que implementam o DOM definindo classes, métodos e interfaces para a manipulação de elementos dos documentos XML. Mas, conforme consideramos anteriormente, as características do DOM podem torná-lo inviável. Se o arquivo XML for muito grande, isso pode se tornar um impedimento, considerando que o DOM carrega toda a estrutura na forma de árvore para a memória, conforme pode ser observado na figura 2.

Figura 2 *Estrutura da API DOM e sua interação com aplicações.*



Fonte: Chappell e Jewell (2002).

Saiba mais

A API DOM é composta por diversas classes e interfaces que possibilitam uma maior interação com os documentos XML. Conforme mencionado, a W3C é o órgão responsável por definir os padrões a serem utilizados no DOM. Para tanto, ela disponibiliza diversas especificações e versões para a manipulação de estruturas XML. Assim uma importante tarefa a ser realizada seria a leitura e a observação de sua parte para a especificação que pode ser encontrada no endereço eletrônico <http://www.w3.org/DOM/>. Acesse-o!

Vamos considerar que nossas aplicações se propõem a lidar com pequenas estruturas XML. Para tanto, existem diversas APIs Java que se propõem a isso, como, por exemplo, a **JDOM** e a **Apache Xerces**. Em nosso caso, utilizaremos a

API Apache Xerces 1.4.4 que tem suporte para a manipulação de documentos XML 1.0 obedecendo às recomendações da W3C, além de conter funcionalidades avançadas para os analisadores, como o suporte ao DOM e à SAX.

A versão 1.4.4 é uma versão estável, o que mercadologicamente é algo mais seguro e menos suscetível a erros. A Apache Xerces (s/d) menciona que a API Xerces tem uma rica capacidade de geração e validação. Utiliza, para isso, classes, métodos e interfaces que possibilitam:

- manipulação de documentos XML para *Web Services*;
- utilização em aplicações em documentos XML de maneira simples e extensível;
- validação de documentos XML;
- integração das camadas de negócio e de dados de uma aplicação que utiliza estruturas XML;
- construção de aplicações que possibilitem a interoperabilidade por meio de documentos XML.

Como o foco deste capítulo é a manipulação dos documentos XML de maneira a tornar possível a interoperabilidade entre as aplicações, vamos ao primeiro exemplo que utiliza a API Apache Xerces para a manipulação de documentos XML usando o *parser* DOM. Crie um documento XML semelhante ao exemplificado no quadro 2.

Quadro 2 *Documento XML a ser manipulado.*

```
<?xml version="1.0"?>
<lista-livro>
      <livro>
            <titulo>Web Services</titulo>
            <isbn>12345678</isbn>
      </livro>
      <livro>
            <titulo>XML e DOM</titulo>
            <isbn>87654321</isbn>
      </livro>
</lista-livro>
```

Repare que, no quadro 2, não está sendo utilizada a validação de um documento DTD, mas nada impediria tal procedimento. Salve o documento apresentado com o nome "arquivo.xml" e garanta que ele esteja no mesmo local de sua

classe Java que manipulará o documento XML. Você ainda deve considerar que, para conseguir executar o código Java, apresentado no quadro 2, é necessário que proceda à instalação da API.

Para a utilização dos Xerces, assim como qualquer outra API, é necessário que você baixe e descompacte seu conteúdo, bem como acrescente a sua variável de ambiente CLASSPATH o caminho no qual estão disponibilizados os recursos da Apache Xerces, ou mesmo acrescente o conteúdo descompactado na pasta lib de seu diretório JAVA_HOME.

Saiba mais

Além da API Xerces, existem diversas outras que possibilitam a manipulação de documentos XML. Em sua maioria, tratam o processo de maneira mais complexa que a Xerces. Outra API que se destaca é a JDOM, que é uma API *open source* específica para a linguagem Java. Atualmente se encontra em sua versão 1.1.1, na qual foram incorporadas classes e interfaces que possibilitam a manipulação de documentos por meio do *parser* SAX. Acesse o endereço eletrônico <http://www.jdom.org/> e obtenha mais informações sobre essa API.

O código Java é disponibilizado no quadro 3 para a manipulação de documentos XML utilizando o *parser* DOM. Então vamos a ele.

Quadro 3 *Acesso a documentos XML utilizando DOM com a API Xerces.*

```
Import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import com.sun.org.apache.xerces.internal.parsers.DOMParser;

public class Main {
      public static void main(String[] args) {
            try {
            // nome do arquivo XML a ser manipulado
            String xmlFilename = "arquivo.xml";
            // Instancia parser DOM do xerces
            DOMParser parser = new DOMParser();
            // processa o arquivo XML
            parser.parse(xmlFilename);
            // recupera o documento em forma de objeto DOM
            Document doc = parser.getDocument();
            NodeList nodes = doc.getElementsByTagName("livro");
```

```
                System.out.println("Há " + nodes.getLength() +
                      " elemento(s).");
                // Para cada elemento
                for(int i=0; i< nodes.getLength(); i++)
                // imprime valor do primeiro nó que ele contém
                System.out.println( (i + 1) + " - " +
                nodes.item(i).getChildNodes().item(i).
                getNodeValue());
          } catch (Exception ex) {
                System.out.println(ex);
          }
     }
}
```

Caso tudo tenha ocorrido conforme o esperado, você deve ter conseguido verificar o conteúdo do documento XML apresentado. Observe que inicialmente importamos as classes disponibilizadas pelo Xerces e que serão utilizadas. Após isso, já no método principal, é indicado o nome do documento XML a ser lido, conforme pode ser observado na linha 9. Na linha 11, é definido o analisador DOM como o *parser* XML a ser utilizado.

Note que, na linha 13, o documento XML é incorporado à instância da classe DOMParser, que é o responsável por carregar a estrutura XML para a memória. Com isso, é possível ter acesso aos elementos existentes dentro do documento, o que ocorre na linha 16, que por meio do método getElementsBy TagName("nomeDoElemento"). O objeto *node* da classe NodeList possibilita a navegação nos elementos selecionados do documento por meio de um laço, que ocorre entre as linhas 20 a 23.

Podemos observar que as estruturas XML consistem em uma arquitetura que, além de neutra, ou seja, independentes de plataformas, são extremamente eficientes e podem ser utilizadas de maneira a tornar possível a troca de mensagens, logo propiciar a interoperabilidade entre aplicações. Porém, como mencionado, o DOM é um modelo baseado em objetos, o que muitas vezes não é uma estrutura atraente computacionalmente.

Conheceremos, na sequência, a estrutura e a utilização do *parser* SAX.

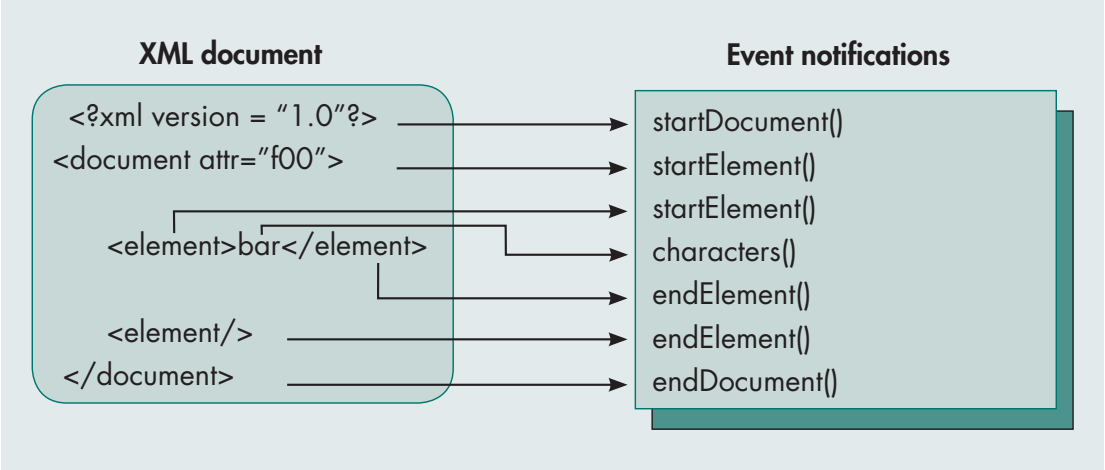
3.1.2 SAX

A Simple API for XML (SAX), assim como o DOM, permite que aplicações tenham acesso ao conteúdo de documentos XML utilizando programas escritos em diversas linguagens de programação como Java, Python, C ou mesmo javaScript. A SAX é um analisador sintático baseado no processamento por meio de eventos, ou seja, sempre são geradas notificações para a aplicação quando forem encontradas marcas, textos e comentários aos quais se necessita ter

acesso nos documentos XML. Apesar de não possuir um órgão oficial regulador de padrões, como no caso do DOM, que define padrões para *web*, acabou se tornando um padrão aberto e livre para criação dos mais diferentes tipos de aplicação. Sua primeira versão foi apresentada em 1998 (MENDES, 2004).

Tais eventos são os responsáveis por retornar os dados existentes na estrutura XML. Diferentemente da API DOM, o conteúdo do documento não é totalmente alocado na memória, o que computacionalmente consiste em uma grande vantagem, conforme pode ser observado na figura 3. Tais eventos acionados por meio de métodos são responsáveis por fazer acesso direto à estrutura do documento XML, tornando a SAX bem mais sofisticada que outros analisadores. Observemos, na figura 3, como trabalha essa API.

Figura 3 *Estrutura de eventos da API SAX.*



Fonte: Newcomer (2002).

Conforme notamos na figura 3, existem alguns métodos que podem ser considerados básicos. Esses métodos são utilizados para a manipulação dos documentos XML e estão baseados na classe DefaultHandler, que é disponibilizada por meio do pacote org.xml.sax. Tal pacote é o responsável por fornecer diversas classes e interfaces importantes para definição do *parser XML*. Segundo Chappell e Jewell (2002), os principais métodos disponibilizados por essa classe são:

- **startDocument()**: método responsável por acessar as *tags* iniciais do documento XML;
- **endDocument()**: método responsável por acessar as *tags* de finalização do documento XML;
- **startElement()**: método que realiza o acesso e a identificação a elementos do documento XML. O analisador sintático dispara o evento quando todos os elementos desejados são processados. Esse método inclui como parâmetro, além do nome da *tag* que será acessada, alguns atributos que também podem ser processados;

- **endElement()**: método responsável por identificar o final de um elemento no documento XML;
- **characters()**: método responsável por obter os dados contidos em um elemento existente dentro do documento XML.

O retorno gerado pelos eventos para a manipulação dos documentos XML para a aplicação é denominado *call-backs* (CHAPPELL; JEWELL, 2002). Assim diversos aspectos podem ser assumidos no momento da implementação de um *parser* XML utilizando a linguagem Java, como, por exemplo, o processo de herança sobre as principais classes existentes para a manipulação de documentos XML com a API SAX.

Transcreva o código apresentado no quadro 4 que utiliza a API SAX para a manipulação de documentos XML e verifique o seu funcionamento.

Quadro 4 *Exemplo de utilização da API SAX em código Java.*

```
import java.io.IOException;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
public class LeitorAPISAX extends DefaultHandler {
      // Inicialização os objetos do parse para ler o
arquivo livro.xml
      public void parse() throws ParserConfigurationException,
            SAXException, IOException {
      SAXParserFactory spf = SAXParserFactory.newInstance();
      SAXParser parser = spf.newSAXParser();
      parser.parse("livro.xml", this);
}
// Indica que o parser achou o início do documento XML.
      public void startDocument() {
            System.out.print("Iniciando a leitura do XML");
      }
// Indica que o parser achou e fim do documento XML.
      public void endDocument() {
            System.out.print("\nFinalizando a leitura do XML");
      }
// Indica que o parser achou o início de uma tag
      public void startElement(String uri, String localName,
            String tag, Attributes atributos){
            //imprime o nome do elemento
```

```
        System.out.print("\n" + tag + ": ");
        //se o elemento possui atributos, imprime
        for (int i=0; i< atributos.getLength(); i++){
        System.out.println(" " + atributos.getQName(i) + "="
        + atributos.getValue(i));
        }
    }
// Indica que o parser achou algum dado.
    public void characters(char[] ch, int start, int length) {
        System.out.print(String.copyValueOf(ch,start,
length).trim());
    }
// Indica que o parser achou o fim de um elemento.
    public void endElement(String uri, String localName,
        String tag){}
public static void main(String args[]){
    LeitorAPISAX reader = new LeitorAPISAX();
    try {
        reader.parse();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
       }
}
```

Conforme apresentado no quadro 4, foram criadas implementações para os diversos métodos definidos na classe DefaultHandler por meio do uso de conceito de polimorfismo, com o intuito de acessar e realizar a leitura dos valores do documentos XML. Repare ainda que o documento lido consiste no mesmo utilizado para o exemplo com a API DOM, ou seja, o arquivo "livro.xml". Inicialmente foi criado o método parser() responsável por definir o arquivo a ser lido por meio das SAXParserFactory e SAXParser. Tais classes trabalham em conjunto para criar a instância de um *parser* XML.

Depois disso tudo, são disparados eventos para cada elemento do documento XML que for encontrado. Para ter acesso a tais eventos, é necessário sobrescrever os métodos responsáveis por isso. Então, em nosso código, foram criadas implementações, como os métodos startDocument(), endDocument(), startElement() e characters().

O primeiro não passa nenhuma informação, apenas indica que o *parser* começará a ler o arquivo XML. O segundo indica que o parser XML achou o fim do documento, para tanto você deve fornecer o que deve ser realizado quando cada evento é encontrado. O método startElement() fornece o nome do elemento, o nome e valor de seus atributos, além de fornecer as informações sobre cada um dos elementos. A implementação do método characters() é o responsável por realizar a impressão dos valores contidos nos elementos.

Seguindo o modelo definido pela API SAX e como pode ser visto no quadro 4, outras classes e interfaces que também são importantes poderiam ser utilizadas possibilitando a manipulação de documentos XML. Segundo Abinader (2006), as principais são:

- **DocumentHandler**: responsável por definir métodos que disparam os principais eventos do *parser*;
- **DTDHandler**: responsável por definir métodos relacionados aos eventos ligados a um arquivo DTD;
- **EntityResolver**: responsável por definir métodos para gerar eventos para preencher entidades, possibilitando lidar com entidade especiais, como, por exemplo, em um banco de dados;
- **ErrorHandler**: responsável por definir métodos que realizam o tratamento dos eventos de erro em uma aplicação. Assim, como nas demais aplicações Java, ao lidarmos com documentos XML que são externos, é importante realizarmos o tratamento de exceção. Logo tais rotinas podem ser utilizadas para realizar um tratamento de erro específico.

Mas nem tudo são flores com a utilização da API SAX. Como você deve ter notado, ela, naturalmente, é mais complexa de se lidar. Além disso, outros problemas de operação podem ser citados. Entre eles, está o fato de a SAX fazer apenas uma leitura para realizar a análise sintática do documento, no caso, é necessário que seja realizado o armazenamento do que será necessário para o restante do processo da aplicação.

É necessário também que consideremos os aspectos positivos da SAX, que é recomendada quando existe a necessidade de processar documentos de maneira sequencial, bem como aplicações do lado servidor, no qual as diversas requisições tornam o processo de leitura de documentos algo a ser considerado, já que existem diversas requisições o que impossibilitaria o uso da DOM, uma vez que seria computacionalmente caro lidar com o problema de memória.

Chegamos ao final do terceiro capítulo. Nele, analisamos conceitos para utilização dos analisadores sintáticos de documentos XML validados ou não, definidos como *parsers*, que são pequenos *softwares*. Além disso, estudamos os dois tipos de interfaces que se destacam, sendo as baseadas em objetos e as baseadas em eventos. Assim, para cada uma dessas abordagens, examinamos implementações de *parsers* que utilizam as APIs DOM e a SAX. Por fim, vimos

implementações em Java para cada uma destas APIs fazendo acesso a documentos XML, bem como as vantagens e as desvantagens de cada uma.

No próximo capítulo, conheceremos os principais conceitos referentes ao protocolo SOAP, que é um padrão para a comunicação entre *Web Services* e abre caminho para a interoperabilidade entre aplicações.

Referências

ABINADER, Jorge Abílio. ***Web Services* em Java**. Rio de Janeiro: Brasport, 2006.

APACHE XERCES. **Projeto Apache Xerces**. Disponível em: <http://xerces.apache.org/>. Acesso em: 21 jul. 2009.

CHAPPEL, David A.; JEWELL, Tyler. **Java *Web Services***. Sebastopol, USA: O'REILLY, 2002.

DEITEL, Harvey M.; DEITEL, Paul J.; NIETO, Lin. **XML**: como programar. Porto Alegre: Bookman, 2003.

MENDES, Antônio. **Programando com XML**. Rio de Janeiro: Campus, 2004.

NEWCOMER, Eric. **Understanding Web services**: XML, WSDL, SOAP and UDDI. Indianapolis, USA: Addison-Wesley Professional, 2002.

Anotações

SOAP – *Simple Object Access Protocol*

4

Introdução

É de extrema importância que os conceitos considerados nos capítulos anteriores tenham sido bem assimilados por você, pois agora iniciaremos a apresentação de um dos principais componentes dos *Web Services*: o protocolo para troca de mensagens denominado SOAP (*Simple Object Access Protocol*). Além disso, realizaremos manipulação de documentos no padrão XML para a criação de mensagens, bem como lidaremos com os analisadores sintáticos, o que abre espaço para a interoperabilidade entre as aplicações, em nosso caso, sistemas do tipo cliente e servidor.

Esperamos que, ao final deste capítulo, você seja capaz de conhecer os principais conceitos relacionados ao protocolo SOAP e compreender a utilização e construção de mensagens utilizando o protocolo SOAP.

Como citado no primeiro capítulo, os *Web Services* oferecem um novo paradigma para a construção de aplicações com sistemas distribuídos, leia-se interoperabilidade entre aplicações. Tal tarefa já era desempenhada por diversas linguagens e API como RMI e CORBA. Com o avanço dos *Web Services* e o passar do tempo, período no qual a evolução da internet foi um marco de transformação mercadológica, abriu-se espaço para um novo modelo de comunicação baseado na troca de mensagens em um formato neutro: o padrão XML que conhecemos nos capítulos anteriores.

A partir desse padrão, diversos protocolos surgiram com o intuito de tornar o processo mais homogêneo e claro. Nenhum protocolo é mais utilizado que o SOAP (*Simple Object Access Protocol*) para a implementação de serviços disponibilizados com *Web Services*. Considerando tamanha a importância de tal protocolo, neste capítulo, conheceremos como se comporta o SOAP, além de verificarmos como é realizada a construção de mensagens que podem ser utilizadas na interoperabilidade entre aplicações.

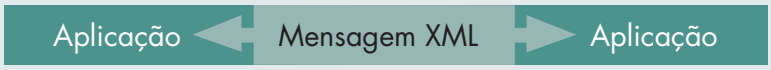
4.1 Mensagens SOAP

No primeiro capítulo, aprendemos que os *Web Services* operam sobre uma arquitetura que era dividida em uma pilha de protocolos divididas em serviços de transporte (serviços fornecidos por protocolos como HTTP ou SMTP), mensagens XML (utilizaremos o SOAP), descrição de serviços (WSDL) e descoberta de serviços (UDDI).

Segundo Deitel, Deitel e Nieto (2003), um *Web Service* consiste em qualquer serviço que possa ser disponibilizado na internet e que, para isso, utiliza um sistema de comunicação de troca de mensagens baseado em XML. Entre os mais diversos sistemas baseados em *Web Services*, surgiram serviços que muitas vezes passam despercebidos em nosso dia a dia, como, por exemplo, a verificação de cartões de crédito, consulta a CEP nos correios, ou mesmo tradutores de línguas estrangeiras, além da atualização de qualquer repositório de dados.

É aí que entra o protocolo mais utilizado para a troca de mensagens em XML: o SOAP. A ideia central desse protocolo é possibilitar a construção e o tráfego de informações estruturadas e extensível em ambientes distribuídos e descentralizados por meio de uso de recursos da linguagem XML, o que possibilita a utilização de outros protocolos também oriundos da padronização XML (DEITEL; DEITEL; NIETO, 2003), conforme é apresentado na figura 1.

Figura 1 *Utilização do padrão XML para comunicações entre aplicações.*



Muitos autores destacam que uma das principais vantagens da utilização do SOAP é sua independência de modelos e linguagens de programação, tornando-o mais maleável e abrangente sem especificidade, como, por exemplo, semânticas proprietárias (DEITEL; DEITEL; NIETO, 2003). Entre os principais objetivos propostos pelo protocolo SOAP, estão características como a simplicidade de desenvolvimento e sua extensibilidade.

O protocolo SOAP é uma das mais recentes maneiras de se trabalhar com a computação distribuída considerando uma especificação na qual todos os elementos da mensagem são trabalhados da mesma maneira, por meio de marcações. Ou seja, as mensagens criadas com a utilização do SOAP, segundo Deitel, Deitel e Nieto (2003), são compostas de um número distinto de partes, que expomos a seguir.

- **Envelope**: é utilizado para descrever o conteúdo da mensagem e alguns detalhes de processamento.
- **Regras**: são utilizadas para definir tipos e codificações utilizadas pela mensagem. É a parte mais crítica já que deve se ter muito cuidado para não comprometer os conceitos de extensibilidade dos *Web Services*.
- **Aplicação**: descreve a aplicação dos envelopes e das regras para chamadas remotas e respostas entre os *Web Services*, geralmente realizados por meio do protocolo HTTP.

Algo semelhante foi visto quando trabalhamos com os arquivos XML puro, no qual existiam as marcações para descrever os dados em um documento XML, bem como os arquivos DTD para descrever as regras para uma estrutura

XML que poderia ser acessada por meio de um analisador sintático (CHAPPELL; JEWELL, 2002). Como vimos, todas as mensagens SOAP empacotadas em um documento XML são chamadas de envelopes, no qual estão contidos elementos que descrevem o conteúdo da mensagem, bem como sua origem e destino.

Saiba mais

Apesar de a maioria dos *Web Services* estar baseada no SOAP, existem outros protocolos que são utilizados para a construção das importantes ferramentas para a interoperabilidade, no caso os *Web Services*. Entre os outros formatos que estão disponíveis no mercado, destacamos o REST (Representational State Transfer). Essa abordagem vem, com o passar dos anos, ganhando muito espaço no desenvolvimento de *Web Services*. Muitos desenvolvedores abrem mão do SOAP por considerar o REST mais simples, já que tudo é visto como um recurso do tipo URI (Universal Resource Identifier), ou seja, como um endereço eletrônico comum na internet, no qual são utilizados métodos padrão para manipulação de dados por meio de uma interface uniforme. Mais informações sobre o protocolo e como proceder para criar *Web Services* com o REST podem ser obtidas no sítio da SUN em <http://java.sun.com/javaone/2009/articles/gen_restful.jsp>. Acesse-o!

Segundo Chappell e Jewell (2002), o conteúdo de um envelope SOAP, que obviamente obedece à sua especificação, permite que sejam enviadas mensagens de intercâmbio entre dois pontos de maneira neutra, independentemente de as aplicações no cliente, por exemplo, estarem escritas em Java ou mesmo C#. A maioria das implementações e *framewroks* disponibilizados geralmente faz uso dos protocolos HTTP e SMTP (CHAPPELL; JEWELL, 2002). Para que possamos trabalhar com os envelopes SOAP, é importante que as requisições para a manipulação e o intercâmbio de dados em nossas aplicações sejam realizadas por meio do método de envio POST do protocolo HTTP.

Diversas características que até então eram extremamente importantes para os sistemas distribuídos, como confiabilidade e segurança nos *Web Services* que utilizam o SOAP, geralmente são omitidas, pois tais detalhes são definidos já na especificação do protocolo, sem que a aplicação tenha de se preocupar com o intercâmbio entre os diversos padrões que eram utilizados para a troca de mensagens nos sistemas distribuídos (DAUM; MERTEN, 2002).

Assim a estrutura de comunicação a ser utilizada pelas aplicações que se propõem a utilizar o SOAP deve obviamente, segundo Chappell e Jewell (2002), considerar:

- **modelo de processamento**: o protocolo SOAP disponibiliza um modelo de processamento distribuído e definido, que possibilita que suas mensagens

sejam enviadas e, no caso, possam realizar saltos entre diversas aplicações intermediárias no caminho até a aplicação destino, tudo com o intuito de garantir que o serviço seja realmente disponibilizado e processado;

- **modelo de extensibilidade**: como uma característica herdada da XML, o SOAP dá suporte a qualquer extensão necessária ao ambiente de comunicação, o que abre espaço para a criação de modelos de comunicação mais realistas e abrangentes, que atendam às necessidades;
- **estrutura de ligação do protocolo**: no SOAP, considerando que ele descende da linguagem neutra XML, é possível utilizar diferentes protocolos subjacentes por meio do conceito de Binding, que são regras formais que definem como o intercâmbio será realizado;
- **construção de mensagem**: consiste no mais importante passo para o processo de comunicação por meio do SOAP, utilizando a especificação XML para compor documentos de dados nos quais deve estar definida, por exemplo, a forma como a mensagem SOAP será enviada por meio de um documento XML.

Para esse processo de comunicação, podem ser aplicadas duas formas de SOAP: o RPC (*Remote Procedure Call*) e o EDI (Electronic Document Interchange ou também encontrado como Electronic Data Interchange). O **RPC** é a base para a computação distribuída, é um método definido para que aplicações possam realizar chamadas a procedimentos ou funcionalidades sobre outra aplicação, passando argumentos, se necessário, e recebendo valores trabalhados como retorno do processo (MENDES, 2004). Já o **EDI** é uma estrutura de transmissão e troca de dados entre aplicações utilizadas para transações automatizadas e padronizadas de negócios entre parceiros, é um padrão para a formatação e interpretação de mensagens e documentos financeiros e comerciais.

Saiba mais

Outra possibilidade para o processo de comunicação em *Web Services* implementados com a linguagem Java é a utilização do JMS (Java Message Service). A API JMS é um padrão aberto baseado nos componentes disponibilizados pela plataforma Java 2 Enterprise Edition (J2EE), que permite criar, enviar, receber e ler mensagens em uma aplicação, como acontece com as necessidades apresentadas pelos *Web Services*, permitindo a comunicação em sistemas computacionais de maneira confiável e assíncrona. Você pode obter mais informações sobre essa API acessando o endereço eletrônico disponibilizado pela SUN Microsystem <http://java.sun.com/products/jms/>. Acesse-o e adquira mais informações sobre a API JMS.

A manipulação de documentos EDI com SOAP é considerada como a forma mais simples de interoperabilidade de dados entre duas aplicações, já que consiste na simples troca de arquivos padronizados. Para tanto, já temos o padrão que é definido na estrutura dos arquivos XML que pode ser ainda validado por meio dos arquivos DTD (CHAPPELL; JEWELL, 2002).

O RPC, como vimos, consiste em outro estilo para a criação do processo de interoperabilidade entre aplicações cliente/servidor que trabalha com a ideia de um *proxy*, no qual as mensagens construídas com o padrão SOAP partem da estação cliente fazendo chamadas ou requisições a métodos de objetos remotos disponibilizados no servidor. Assim toda chamada realizada por um cliente exige que seja gerada uma resposta. Embora as implementações com o protocolo SOAP não exijam a utilização do estilo RPC, a maioria das ferramentas e *frameworks* SOAP mais modernos volta-se para utilização de chamadas remotas (CHAPPELL; JEWELL, 2002).

Mas como são realizadas tais chamadas remotas aos servidores com o RPC? Como mencionamos, é aí que surge a importância dos metadados ou estruturas XML. A ideia geral consiste no encapsulamento das chamadas RPC segundo o padrão do protocolo SOAP antes que elas sejam enviadas para a aplicação servidora por meio de uma rede. Após isso, o servidor remoto realiza o processo contrário, fazendo o desencapsulamento da chamada por meio de um *parser* que extrai as chamadas dos métodos dos objetos remotos. Com isso, é realizado o processamento da requisição que envia uma mensagem de resposta ao cliente que, ao ser recebida, disponibiliza a aplicação cliente.

Segundo a W3C (s/d), em sua especificação para o protocolo SOAP, algumas informações importantes devem ser definidas, pois são necessárias para as requisições do tipo RPC entre aplicações. São elas:

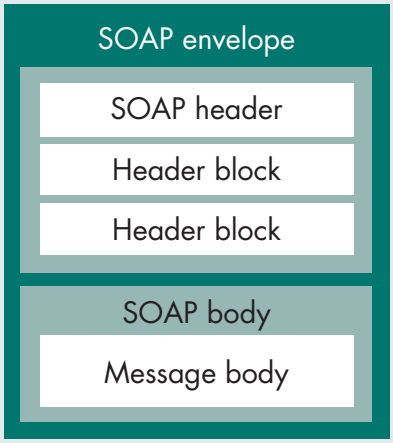
- a URI, no caso o endereço eletrônico do objeto alvo disponibilizado no servidor remoto;
- o nome do método a ser acessado no servidor;
- os parâmetros necessários para o processamento do método;
- uma assinatura opcional para o método a ser requisitado;
- um cabeçalho opcional para a requisição definida na mensagem SOAP.

Você já deve estar curioso para conhecer como é a estrutura de uma mensagem ou envelope SOAP, não é mesmo? Agora trabalharemos com a construção de mensagens por meio do protocolo SOAP.

Inicialmente uma mensagem SOAP consiste em um envelope composto por dois itens: *header* (cabeçalho) e *body* (corpo). O bloco de *header* é opcional. Ele contém informações relevantes para identificar como a mensagem será processada, incluindo rotas e configurações para envio e recepção, autenticação e

contextos de transações (CHAPPELL; JEWELL, 2002). O segundo item de nosso envelope, no caso o *body*, empacota a mensagem a ser entregue e processada. Qualquer coisa que respeite a sintaxe dos documentos XML pode fazer parte do corpo da mensagem. Para que você tenha um entendimento mais claro dos itens que podem compor um envelope SOAP, observe a figura 2.

Figura 2 *Estrutura de um envelope SOAP.*



Fonte: Chappell e Jewell (2002).

Toda mensagem SOAP deve ter pelo menos um elemento do tipo *body* em seu contexto. Mas como fica isso em nossos documentos XML? Vamos a um exemplo da criação de mensagens de requisição e de reposta. Observe o quadro 1 que contém tais estruturas.

Quadro 1 *Estrutura de uma mensagem de requisição SOAP.*

```
<?xml version="1.0"?>
<soap:Envelope  xmlns:soap="http://www.w3.org/2001/12/
soap-envelope"
      soap:encodingStyle="http://www.w3.org/2001/12/
soap-encoding"
      xmlns:tiposns="http://www.w3.org/2001/XMLSchema">
      <soap:Header>
            <m:atenticacao xmlns:m="http://www.unitins.
br/ws ">123</m:atenticacao>
</soap:Header>
<soap:Body>
      <m:retornaNome xmlns:m="http://www.unitins.br/ws ">
            <numdre type="tiposns:int">123456</drenum>
```

```
        </m:retornaNome>
        </soap:Body>
    </soap:Envelope>
```

Como podemos observar no quadro 1, a mensagem SOAP deve obrigatoriamente ser identificada como uma mensagem do tipo XML, por ser naturalmente um documento XML. Deve obedecer às regras de sintaxe desse padrão, o que é feito na linha 1 de nossa mensagem. Logo após, temos a marcação que define o documento XML como sendo um envelope SOAP. A *tag* "soap:Envelope", apresentada na linha 2, é o elemento raiz da nossa mensagem. Só pode existir um único elemento desse tipo em um documento que se propõe a enviar uma mensagem SOAP. Essa marcação é composta por diversos atributos opcionais, mas não menos importantes para a definição da mensagem.

O primeiro atributo consiste no *namespace* xmlns:soap=http://www.w3.org/2001/12/soap-envelope. Esse atributo é obrigatório, pois é ele que identifica que o documento XML consiste em um envelope SOAP. Assim, sem o atributo "xmlns", nosso documento não será interpretado como uma mensagem válida do tipo SOAP entre *Web Services*. O valor desse atributo consiste em um endereço na W3C, que pode variar segundo a versão do SOAP a ser utilizada.

Outro atributo utilizado consiste no "soap:encodingStyle", que, apesar de opcional, pode ser utilizado para definir tipos de dados (int, char etc) para os elementos filhos. A W3C, assim como no caso do atributo xmlns, também define um documento padrão que é disponibilizado em conjunto com a versão SOAP, como, por exemplo, http://www.w3.org/2001/12/soap-encoding. O último atributo consiste na *namespace* do esquema de validação a ser seguido pelo documento conforme pode ser observado na linha 4 do quadro 1.

Repare que, em nosso exemplo, utilizamos os dois blocos possíveis em um envelope SOAP: *header* e *body*. Conforme citado, o cabeçalho *header* do SOAP é um elemento opcional que contém informações específicas para uma aplicação como metainformações, como, por exemplo, itens importantes para a autenticação em um *Web Service*. Caso seja definido que uma mensagem SOAP terá um cabeçalho, esse elemento deve ser o primeiro do envelope, ou seja, a primeira marcação filha da *tag* que define o envelope, como foi especificado no exemplo do quadro 1.

Depois de finalizado o bloco de *header* de um envelope, é necessário que seja realizada a definição do corpo da mensagem. Logo nossa mensagem deve, diferentemente do cabeçalho, criar um elemento filho do tipo *body*. Esse elemento deve ser o responsável por apresentar a informação da mensagem, no caso, é o responsável por realizar a chamada ao método do objeto remoto em um cliente ou servidor, ou mesmo o resultado do processamento.

Em nosso exemplo, como consta no quadro 1, na mensagem de requisição, o elemento *body* tem uma marcação filho, que é a responsável por realizar a chamada a um método. Ou seja, esse elemento define a operação requisitada na mensagem. Para ficar mais clara tal afirmação, consideremos mais um exemplo, só que agora definiremos uma mensagem de resposta para nossa requisição obedecendo aos critérios estabelecidos pela SOAP.

Quadro 2 *Estrutura de uma mensagem de resposta SOAP.*

```
<?xml version="1.0"?>
<soap:Envelope      xmlns:soap="http://www.w3.org/2001/12/
soap- envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-
encoding"
  xmlns:tiposns="http://www.w3.org/2001/XMLSchema">
  <soap:Header>
    <m:autenticacao   xmlns:m="http://www.unitins.br/ws
">123456</m:autenticacao>
  </soap:Header>
    <soap:Body>
    <m:retornaNomeResponse  xmlns:m="http://www.unitins.
br/ws">
      <nome type="tiposns:string">Manoel Santos</nome>
    </m:retornaNomeResponse>
      </soap:Body>
</soap:Envelope>
```

Repare que a definição e o cabeçalho da mensagem SOAP são semelhantes ao envelope de requisição, bem como a definição do elemento de corpo da mensagem. No corpo, são acessados diversos serviços específicos na forma de atributos ou mesmo nameSpace definidos pela especificação SOAP. Assim, no quadro 2, algumas marcações definem os serviços ou funcionalidade a serem efetuadas, por exemplo, as marcações iniciadas com "<m:...", que definem chamadas a métodos em objetos remotos. No quadro 2, dois métodos são claramente invocados: o autenticar e retornarNomeReponse.

As principais vantagens da utilização do protocolo SOAP são sua neutralidade, graças ao fato de ele ser baseado em documentos XML, bem minimização dos problemas de rede, já que o SOAP é considerado um protocolo de comunicação assim como o HTTP. Claro que itens como a complexidade do processo podem comprometer a utilização desse protocolo para a criação de *Web Service*, mas nada que os benefícios trazidos por esse modelo não supram.

Neste capítulo, vimos que o protocolo mais utilizado para a troca de mensagens em XML é o SOAP, que possibilita a construção e o tráfego de informações

estruturadas e extensíveis a ambientes distribuídos e descentralizados. Para isso, utiliza recursos da linguagem XML, o que possibilita a uso de outros protocolos também oriundos da padronização XML. Ou seja, as mensagens criadas com a utilização do SOAP são compostas de um número distinto de partes. A mais importante é o envelope. Vimos também que, para a comunicação nos *Web Services*, existem duas formas de aplicarmos o SOAP: o RPC o EDI.

Saiba mais

Como vimos até aqui, o protocolo SOAP é uma ferramenta padrão para os *Web Services* e diversas possibilidades de se trabalhar com o protocolo estão disponíveis no mercado. Entre elas, podemos citar algumas referências para a construção de *Web Services*, como o *framework* Apache AXIS, que pode ser encontrado no endereço eletrônico <http://ws.apache.org/axis/>, ou o projeto XFire, disponível em <http://xfire.codehaus.org/>, que é mais uma vertente do JAX-WS, que é *o framework* disponibilizado pela própria Sun Microsystem para a criação e manipulação de *Web Services*, localizado no endereço eletrônico <https://jax-ws.dev.java.net/>. Acesse esses endereços para ampliar seu conhecimento sobre o protocolo SOAP.

No próximo capítulo, conheceremos outro arquivo de extrema importância para que você possa compreender o funcionamento dos *Web Services*, o WSDL, arquivo que possibilita o acesso aos serviços disponibilizados.

Referências

CHAPPEL, David A.; JEWELL, Tyler. **Java *Web Services***. Sebastopol, USA: O'REILLY, 2002.

DAUM, Berthold; MERTEN, Udo. **Arquitetura de sistemas com XML**. Rio de Janeiro: Campus, 2002.

DEITEL, Harvey M.; DEITEL, Paul J.; NIETO, Lin. **XML**: como programar. Porto Alegre: Bookman, 2003.

MENDES, Antônio. **Programando com XML**. Rio de Janeiro: Campus, 2004.

W3C CONSORTUIM. **World Wide Web Consortium**. Disponível em: <http://www.w3.org/>. Acesso em: 20 ago. 2009.

Anotações

WSDL – *Web Service Definition Language*

5

Introdução

Para que você compreenda melhor este capítulo, é interessante que tenha assimilado os conceitos, os arquivos e os demais elementos que compõem um *Web Service*, temas vistos nos capítulos anteriores. Agora trabalharemos com a descrição de aplicações e serviços disponibilizados pelos *Web Services* em servidores *web*. Mas com certeza, o item mais importante para que você tire maior proveito deste capítulo é dominar a criação de documentos do tipo XML, uma vez que veremos, no decorrer deste capítulo, um arquivo WSDL, que nada mais é que uma estrutura XML.

Nos capítulos anteriores, falamos exaustivamente sobre a estrutura e os protocolos envolvidos na criação dos *Web Services*. Entre todos, vimos a importância dos documentos XML, que são a base para a comunicação entre um *Web Service* e uma aplicação cliente, que é a base para a interoperabilidade entre sistemas.

Conversamos sobre o mais importante desses protocolos, no caso o SOAP (*Simple Object Access Protocol*), que, por meio de uma estrutura formal e flexível, possibilita que ambas as aplicações troquem mensagens, descrevendo o que deve ser feito além dos dados necessários para isso. Mas somente isso não basta para que os *Web Services* funcionem. Para tanto, é necessário que tenhamos um arquivo que seja capaz de descrever os serviços que um *Web Service* está disponibilizando. É aí que entram os arquivos WSDL.

Neste capítulo, conheceremos a importância, a formatação e as regras que devem ser obedecidas para a criação desse arquivo que é de extrema importância para os *Web Services*, já que possibilita que as aplicações cliente possam acessar e manipular os serviços públicos disponibilizados em um servidor.

5.1 Estruturas WSDL

Conforme mencionamos, o SOAP é uma estrutura XML para a comunicação entre *Web Services* e aplicações clientes. Porém esse protocolo não supre a necessidade de descrever as informações dos serviços e como invocá-los. Para suprir tal deficiência, surgiram os arquivos WSDL.

Então era necessário um protocolo de comunicação e formatação de mensagem padronizado baseado na *web*. Isso passou a ser de extrema

importância para os *Web Services*, pois era necessário divulgar e descrever a estrutura de comunicação a ser utilizada pelos serviços fornecidos, bem como a estrutura a ser seguida nas trocas de mensagens. É aí que entra o WSDL, que provia uma documentação para detalhar a comunicação em sistemas distribuídos (CHAPPELL; JEWELL, 2002).

As estruturas WSDL são nada mais nada menos que documentos XML para descrever como uma coleção de serviços e seus pontos de acesso podem ser capazes de responder a mensagens de interoperabilidade entre duas ou mais aplicações e um servidor *Web Service*. Ou seja, um arquivo WSDL é um recipiente utilizado para realizar a automatização dos meios de comunicação, no qual estão descritos detalhes de como deve ser realizada a troca de mensagens para se obter o resultado esperado (DEITEL; DEITEL; NIETO, 2003).

Os arquivos WSDL devem informar o que cada um dos serviços faz, como invocá-los e, o mais importante, como fazer para achá-los na *web* de maneira que os outros possam usufruir de suas funcionalidades. Os arquivos do tipo WSDL definem um formato, com mensagens de entrada e saída, baseadas no protocolo SOAP, que é uma maneira formal de garantir que todos os itens obrigatórios sejam cumpridos antes que os serviços sejam disparados.

Saiba mais

Um dos mais importantes fatores que tornaram os arquivos WSDL um padrão de mercado para a divulgação de operações, ou serviços para os *Web Services*, está no fato de esses arquivos serem uma recomendação da W3C. Assim, como a maioria dos protocolos e padrões da W3C, o WSDL 1.1 segue os conceitos e as regras da especificação dos documentos XML, os quais têm suporte na maioria dos navegadores disponíveis no mercado, o que possibilita a visualização desse arquivo, já que geralmente eles são disponibilizados pelos *Web Services*, como, por exemplo, no Axis e no JAX-WS. Mais informações sobre a especificação dos arquivos WSDL podem ser encontradas no endereço eletrônico <http://www.w3.org/TR/wsdl>.

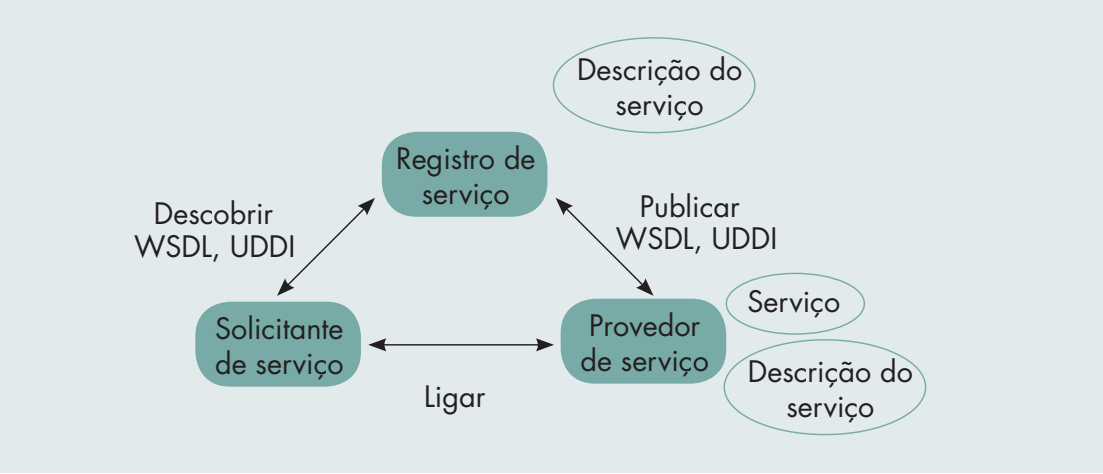
Conforme mencionamos nos capítulos anteriores, tal processo ou ideia não é algo novo, já que outras tecnologias para a criação de sistemas distribuídos como o CORBA e Java RMI se utilizam de estruturas semelhantes para definir interfaces ou assinaturas padrão para acesso à camada lógica em um par de aplicações que se comunicam. Porém a WSDL consiste em uma nova forma de realizar tal serviço já que oferece um grau de extensibilidade não encontrado nas demais tecnologias citadas.

Segundo Deitel, Deitel e Nieto (2003), a extensibilidade é capaz de permitir que a WSDL possa ser utilizada para:

- descrever parâmetros e suas mensagens, independentemente do formato ou mesmo de protocolos utilizados nas redes de computadores para o intercâmbio das informações;
- realizar o tratamento das mensagens descrevendo que tipos de dados serão trocados entre as aplicações;
- efetuar o tratamento e divulgação das portas em um servidor que podem ser utilizadas para acesso aos serviços *web*, mapeando a porta, tipo de protocolo e formatos dos dados necessários para a realização das operações.

Com isso, acreditamos que você já deva ter percebido a importância desse arquivo para os *Web Services*. Sem eles, não é possível que aplicações clientes acessem serviços, pois é a estrutura WSDL a responsável por publicar os serviços, bem como informar as regras que devem ser seguidas. Quando uma aplicação cliente acessar um *Web Service*, o primeiro elemento a ser trabalhado é o WSDL. Aqui é importante relembrarmos a abstração de um *Web Service*, conforme apresentado no primeiro capítulo e observado na figura 1.

Figura 1 *Abstração de um Web Service entre cliente e funcionalidades.*



Fonte: Chappell e Jewell (2002).

Conforme apresentado na figura 1, o WSDL é a base para que o registro UDDI possa informar aos clientes qual serviço está disponível e como ele deve proceder para fazer uso desse serviço. Para que isso seja possível, existe uma estrutura bem formal nesse tipo de documento.

Um arquivo do tipo WSDL que apresenta uma coleção de serviços necessita seguir uma padronização de elementos definidos. Esses elementos ou marcações nada mais são que uma estrutura XML, que se inicia pelo elemento raiz e é denominado <definitions>. O elemento "definitions" pode conter diversos atributos de nameSpace, que geralmente são utilizados para garantir a validade do arquivo com referências a arquivos do tipo XSD (XML Schema Definition). Geralmente o endereço eletrônico desse documento é utilizado para sua definição (CHAPPELL; JEWELL, 2002).

Você provavelmente deve estar familiarizado com tais elementos, já que tal situação foi observada e é semelhante à vista na definição das mensagens e dos envelopes SOAP, trabalhadas no capítulo anterior. O WSDL se utiliza das nameSpaces para maximizar o processo de reutilização e explorar, assim, a característica de extensibilidade dos documentos XML, bem como dos componentes em um documento WSDL, utilizando-se de itens como os atributos para realizar no conteúdo do arquivo referências a outros elementos, seja dentro ou fora do documento.

Dentro de nosso elemento raiz do tipo "definition", podem ser acrescentados outros tipos de marcações úteis para descrever tanto os serviços quantos aspectos ligados a regras que devem ser obedecidas, conhecidas com *binding*, e os parâmetros de uma rede, que geralmente são as portas disponibilizadas para a conexão entre as aplicações. Essas portas são as ligações que serão utilizadas e devem estar associadas às mensagens que são enviadas tanto ao *Web Service* quanto ao cliente (ABINADER, 2006). A especificação WSDL disponibilizada pela W3C define além do elemento "definition" outros elementos que conheceremos nas próximas seções. Então vamos a eles.

5.1.1 Elemento <import>

O elemento import tem por finalidade principal realizar a inserção de trechos de outros elementos XML. Tal elemento é semelhante à diretiva #include existente nas linguagens C e C++. O uso de tal elemento proporciona aos documentos WSDL a capacidade de modularização, criando um ambiente propício ao reuso. Dessa forma, as estruturas dos arquivos WSDL podem ser facilmente manipuladas sem que isso tenha interferência em seus serviços, claro que considerando os aspectos dos documentos XML bem formados (CHAPPELL; JEWELL, 2002). Para que você possa explorar mais tal elemento, é importante mencionar que você pode utilizar quantos elementos forem necessários.

O elemento import é o responsável por possibilitar a separação, em diversos arquivos, as várias partes de uma definição WSDL. A notação para a utilização do elemento import em arquivos WSDL é apresentada no quadro a seguir.

Quadro 1 *Utilização da marcação import.*

```
<import namespasce="http://www.unitins.com/definitions/
service"
location="http://localhost/ws/service.wsdl/" />
```

Como você deve ter observado no elemento import apresentado, são definidos dois atributos importantes para sua definição. O primeiro atributo denomi-

nado nameSpace define como o arquivo será tratado e acessado dentro de nosso documento, e o segundo define a localização do documento a ser importado para o arquivo WSDL. Outros elementos podem ser citados para a definição de nossos serviços em um *Web Service*. Conheceremos, na sequência, o elemento *types*.

5.1.2 Elemento <types>

Esse tipo de elemento, que pode estar presente em um arquivo WSDL, trata-se de um *container* de dados para a definição de tipos que estarão presentes na mensagem. Eles podem utilizar referências a elementos externos como um esquema de validação definidos em um arquivo XSD (CHAPPELL; JEWELL, 2002). Um exemplo da utilização desse tipo de marcação é apresentado no quadro 2.

Quadro 2 *Utilização da marcação types.*

```
<wsdl:part name="aluno" type="xsd:string" />
<wsdl:part name="disciplina" type="xsd:int" />
```

Conforme pode ser observado no quadro anterior, o elemento *type* é definido como sendo do tipo *part*, ou seja, parte da definição dos dados que serão utilizados nas operações disponibilizadas pelo arquivo WSDL. Mas dentro da *tag*, repare que temos a definição do tipo de dado a ser utilizado por meio do esquema XSD e os tipos de dados.

Vamos ao próximo elemento de um arquivo WSDL.

5.1.3 Elemento <message>

Trata-se de um dos mais importantes arquivos WSDL. Ele consiste na abstração da definição dos tipos de dados a serem trocados na comunicação, ou seja, os dados que serão transmitidos. Esses elementos podem conter um ou mais elementos do tipo <type>, que formam as partes que definem os dados da mensagem. As marcações <type> dentro do elemento <message> definem quais parâmetros serão passados pela aplicação cliente para o processamento das operações, bem como a resposta enviada por um servidor como retorno do processamento de um serviço do *Web Service* (CHAPPELL; JEWELL, 2002). Vamos a um exemplo para uma melhor fixação desse elemento.

Quadro 3 *Utilização da marcação message.*

```
<wsdl:message name="getAluno">
```

```
        <wsdl:part name="aluno" type="xsd:string"/>
    </wsdl:message>
    <wsdl:message name="getDisciplina">
        <wsdl:part name="disciplina" type="xsd:string" />
    </wsdl:message>
```

Repare que cada um dos elementos *message* contém um nome que o define de maneira individual para que possa ser acessado por outros elementos de um *Web Service* para a manipulação de seu conteúdo. Além disso, conforme mencionado, esses nomes geralmente estão descritos de maneira que possamos identificar se seu conteúdo é uma requisição ou respostas do sistema. Geralmente esses elementos são acessados por elementos do tipo *operation*. Logo nada mais interessante a fazer a não ser conhecer como são definidas essas marcações.

5.1.4 Elementos <operation>, <portType> e <binding>

O **elemento *operation*** é uma marcação XML muito importante para os *Web Services*, pois ele é o responsável pela descrição das operações, em nosso caso, já que trabalhamos com Java, os métodos que fornecem os serviços suportados pelo *Web Service*. Tais elementos geralmente estão concentrados em outros dois tipos de marcações: o *portType* e *binding* (CHAPPELL; JEWELL, 2002). Esses dois elementos têm um conjunto de *operations*, que definem um atributo com seu nome além de um atributo para especificação dos parâmetros que serão utilizados nas operações.

O **elemento *portType*** refere-se a aspectos de configuração das operações disponibilizadas definindo um ou mais parâmetros necessários para a realização do serviço. Já o **elemento *binding*** é o responsável por definir a regra para um protocolo e uma especificação dos formatos dos dados para comunicação, servindo-se de um tipo de porta em particular usada para execução de uma operação, ou seja, um serviço em específico (CHAPPELL; JEWELL, 2002). Para uma melhor visualização desses processos, analise o quadro 4.

Quadro 4 *Marcação binding.*

```
<wsdl:binding name="ServiceSoapBinding" type="impl:Servico">
<wsdlsoap:binding style="rpc" transport=http://schemas.
xmlsoap.org/soap/http />
```

Repare que, no quadro 4, o elemento *operation*, dentro dos elementos *portType* e *binding*, utiliza-se das marcações *input* e *output* para acessar as mensagens definidas. Dessa forma, as mensagens que citamos na seção anterior podem ser acessadas de quatro maneiras. Segundo W3C (s/d), as mensagens podem ser utilizadas pelos elementos *operation*, como:

- **unidirecional**: mensagem que é enviada de um cliente para um serviço no qual não é gerada resposta alguma;
- **solicitação-resposta**: é a mensagem mais utilizada nesse tipo de sistema, já que um cliente envia uma mensagem de requisição a um servidor e aguardar o recebimento de uma resposta como resultado de sua solicitação. Tal formato é o que conhecemos no capítulo anterior como o RPC (*Remote Procedure Call*);
- **pedido-resposta**: é o caminho contrário ao utilizado pelas mensagens do tipo solicitação-reposta, já que o pedido sai do servidor que fica aguardando uma mensagem de resposta a ser enviada pelo cliente do sistema, o que não é algo tão natural, pois nem sempre é possível saber ou definir um cliente em específico para se solicitar algo sem que ele tenha se identificado antes;
- **notificação**: trata-se de mensagens que são enviadas a todos os clientes do *Web Service* independente de eles estarem nesse momento consumindo ou não algum serviço.

Outros elementos que têm destaque para que os *Web Services* possam disponibilizar seus serviços são os elementos *service* e *port*, que consideraremos a seguir.

5.1.5 Elemento <*service*> e <*port*>

O elemento *service* é uma coleção de diversos parâmetros que são importantes para definir o acesso aos *Web Services*. Entre esses diversos parâmetros, podemos citar o elemento *port*, que está diretamente associado ao processo de conexão e troca de mensagens, já que é nele geralmente estão definidos itens como a localização dos *Web Services*, bem como a troca de mensagens (CHAPPELL; JEWELL, 2002). O quadro 5 demonstra a criação de um trecho do arquivo WSDL, no qual é definida a utilização do elemento *service* e *port*.

Quadro 5 *Marcações service e port.*

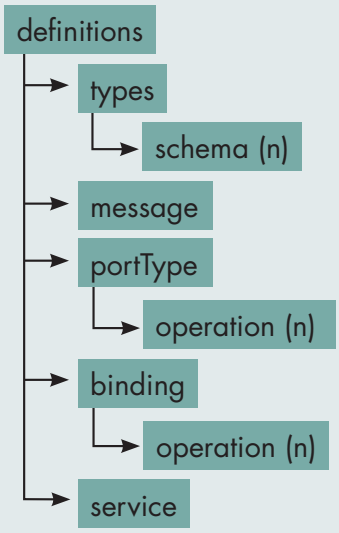
```
<service name="ServicoService">
<port binding="impl:ServicoSoapBinding" name= "Servico">
```

```
<address location="http://localhost:8080/axis/Servico.jws"/>
</port>
</service>
```

Apresentamos os principais elementos que podem estar contidos em um arquivo do tipo WSDL. Nosso próximo passo é entender como esses elementos estão estruturados dentro do documento XML. Sua arquitetura básica é apresentada de duas formas, de maneira a tornar o processo mais didático para seu aprendizado sobre essa importante ferramenta utilizada pelos *Web Services*. Então observe tais definições na figura 2.

Figura 2 *Estrutura básica de um arquivo WSDL.*

```
<definitions>
    <import>
    <types>
        <schema></schema>
    </types>
    <message>
        <part></part>*
    </message>
    <portType>*
        <operation>*
            <input></input>
            <output></output>
            <fault></fault>*
        </operation>
    </portType>
    <binding>*
        <operation>*
        <input></input>
        <output></output>
        </operation>
    </binding>
<service>*
<port></port>*
</service>
</definitions>
```



Fonte: Chappell e Jewell (2002).

O arquivo WSDL pode conter a definição de diversas marcações de alguns elementos. Como você deve ter observado, alguns elementos têm um "*", que denota que tais elementos podem aparecer diversas vezes dentro de seu arquivo WSDL (MENDES, 2004). Vamos a um exemplo de arquivo WSDL.

Os *Web Services* geralmente consistem em classes nas quais são acessados tanto os métodos quanto os serviços. Dessa forma, criaremos um exemplo básico de uma classe na qual apenas retornamos um nome passado como parâmetro. Isso pode ser observado no quadro 6.

Quadro 6 *Classe Java para a definição de um WSDL.*

```
public class Servico{
      public String dados(String nome){
            return nome;
      }
}
```

Note que não há tratamento especial, diferente do que você já conhece sobre a criação de classes para aplicações em console, visual ou mesmo *web*. Para a classe Java, um arquivo WSDL válido e bem formado poderia ser algo semelhante ao que é apresentado no quadro 7.

Quadro 7 *Arquivo WSDL criado baseado na classe Java.*

```
<definitions targetNamespace="http://localhost:8080/axis/
Servico.jws">
      <message name="dadosRequest">
            <part name="nome" type="xsd:string"/>
      </message>
      <message name="dadosResponse">
            <part name="dadosReturn" type="xsd:string"/>
      </message>
      <portType name="Servico">
            <operation name="dados" parameterOrder="nome">
      <input message="impl:dadosRequest" name="dadosRequest"/>
      <output message="impl:dadosResponse" name="dadosResponse"/>
            </operation>
</portType>
```

```
<binding name="ServicoSoapBinding" type="impl:Servico">
<binding style="rpc"
  transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="dados">
    <operation soapAction=""/>
    <input name="dadosRequest">
      <body
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="http://DefaultNamespace" use="encoded"/>
    </input>
    <output name="dadosResponse">
      <body
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="http://localhost:8080/axis/Servico.jws"
      use="encoded"/>
    </output>
  </operation>
</binding>
<service name="ServicoService">
<port binding="impl:ServicoSoapBinding" name="Servico">
<address  location="http://localhost:8080/axis/Servico.
jws"/>
</port>
</service>
</definitions>
```

A construção de um arquivo WSDL pode requerer, conforme observamos, certo trabalho, caso fosse necessária sua construção de maneira manual. Porém existem diversas especificações para *Web Services* que realizam o processo de geração e controle do tráfego de mensagens, baseadas no protocolo SOAP, que serão trocadas durante a comunicação entre o servidor e o cliente.

Saiba mais

Além das diversas APIs, como Axis, JWS ou mesmo a WSDL2Java, que possibilitam a criação automática dos arquivos WSDL, outras ferramentas como o *netbeans* e o eclipse fornecem suporte à edição de documentos WSDL. Além disso, algumas vezes, torna-se necessário que sejam realizadas algumas configurações que tornem o processo desempenhado pelos

arquivos WSDL mais customizados, possibilitando que alguns serviços não estejam disponíveis ou mesmo sejam divulgados. Mais informações sobre tais processos podem ser obtidos no endereço eletrônico <http://www.netbeans.org/kb/60/websvc/wsdl-guide_pt_BR.html>.

No quinto capítulo, vimos que as estruturas WSDL são nada mais nada menos que documentos XML, que servem para descrever como uma coleção de serviços e seus pontos de acesso podem responder a mensagens de interoperabilidade entre duas ou mais aplicações e um servidor *Web Service*. Um arquivo do tipo WSDL que apresenta uma coleção de serviços necessita seguir uma padronização de elementos definidos. Esses elementos ou marcações nada mais são que uma estrutura XML que se inicia pelo elemento raiz, que é denominado <definitions>. Vimos outros elementos como: *import*, *types*, *message*, *operation* e service e finalizamos o capítulo com análise de um exemplo de um documento WSDL.

No próximo capítulo, veremos a criação de *Web Service* na prática, utilizaremos para isso a API Axis e o servidor Tomcat, possibilitando a criação de serviços e arquivos vistos no decorrer deste caderno.

Referências

ABINADER, Jorge Abílio. ***Web Services* em Java**. Rio de Janeiro: Brasport, 2006.

CHAPPEL, David A.; JEWELL, Tyler. **Java *Web Services***. Sebastopol: O'REILLY, 2002.

DEITEL, Harvey M.; DEITEL, Paul J.; NIETO, Lin. **XML**: como programar. Porto Alegre: Bookman, 2003.

MENDES, Antônio. **Programando com XML**. Rio de Janeiro: Campus, 2004.

W3C Consortuim. **World Wide Web Consortium**. Disponível em: <http://www.w3.org/>. Acesso em: 20 ago. 2009.

Anotações

Desenvolvendo *Web Services*

6

Introdução

Para que você possa tirar o máximo proveito deste capítulo, é recomendável que os conceitos dos demais capítulos trabalhados até aqui tenham sidos assimilados, pois lidaremos, mesmo que indiretamente, com documentos e arquivos como XML, SOAP, WSDL, bem como classes Java e APIS externas que conhecemos anterioremente. Além disso, é importante que você também utilize conceitos trabalhados na disciplina de Programação do quarto período, na qual estudou diversos métodos para lidar com aplicações *web*, visto que, para a conclusão deste capítulo, é essencial a manipulação dos servidores *web*. A partir desse conteúdo, você estará apto para conhecer os principais componentes disponíveis no mercado para a criação de *Web Services* Java e criar *Web Services* utilizando a linguagem de programação Java.

Lidar com os conceitos de sistemas distribuídos dominou o mercado até poucos anos atrás. A internet ganhou força considerável e tal modelo de implementação e interoperabilidade entre aplicações passou a ser revisto, considerando a abrangência e mesmo as limitações impostas por tal tecnologia. Como vimos nos capítulos anteriores, devido a tais deficiências, os *Web Services* surgiram como uma tendência e hoje já são uma realidade. Tal perspectiva atualmente só tem crescido, como pode ser comprovado com o trabalho de grandes empresas, como Google, prestadoras de cartões de créditos e até mesmo os correios do Brasil, que já utilizam tal modelo. Até aqui foram apresentados a você os principais componentes que, em conjunto, formam o todo para a criação dos *Web Services*.

Neste capítulo, colocaremos a mão na massa e criaremos um *Web Service*. Para isso, utilizaremos APIs baseadas na linguagem de programação Java.

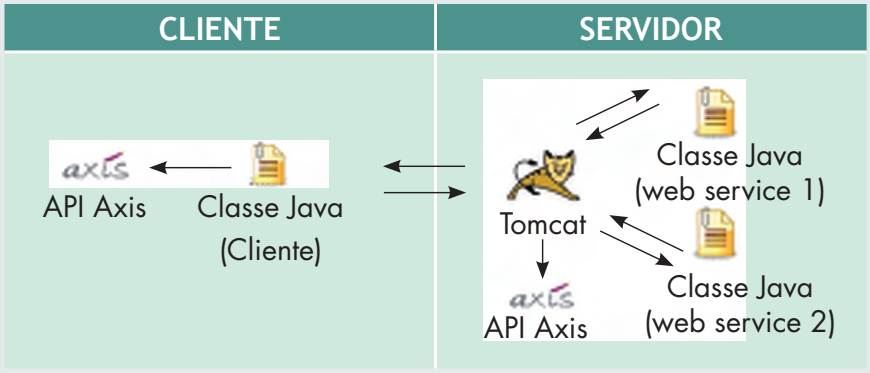
6.1 Criando um *Web Services*

Vimos que a tecnologia dos *Web Services* é uma salada de frutas tecnológica, ou seja, a interoperabilidade entre diversas APIs e aplicações. Para criação do primeiro *Web Service*, lidaremos com algumas tecnologias *open source*, que atualmente são a referência para o desenvolvimento desse tipo de serviço no mercado utilizando a linguagem Java.

Os *Web Services* são aplicações que podem ser acessadas por meio do protocolo HTTP ou mesmo SMTP na *web* e, por isso, necessitam estar localizados

em um servidor *web* que responde às solicitações. Além disso, precisamos lidar com outras tecnologias que devem ser utilizadas para que as classes sejam consideradas e tratadas como um *Web Service* com serviços acessíveis. O processo de interação dos itens que compõem o *Web Service* são apresentados na figura 1.

Figura 1 *Itens e interações para a criação de um Web Service.*

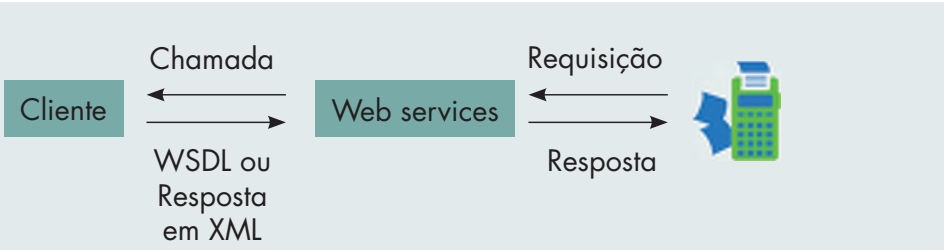


Conforme você pode observar na figura 1, os *Web Services* e clientes são classes Java seguindo os conceitos vistos por você em na disciplina de Programação do segundo período. Lembre-se de que qualquer aplicação que utiliza a linguagem Java é uma classe. Precisamos de APIs que auxiliem no entendimento do suporte para que as classes sejam interpretadas como *Web Services*, tanto por parte do servidor *web* quanto pela máquina virtual instalada na máquina do cliente. Como apresentado na figura 1, tal função é desempenhada pela API Apache Axis. Agora que você já conhece o esquema a ser utilizado pelas tecnologias que auxiliam no processo de interoperabilidade e disponibilização de serviços, veremos como trabalhará o *Web Service*.

No estudo de caso, criaremos uma aplicação cliente desenvolvida para funcionar no modo *console*, que realizará uma chamada a um serviço disponibilizado em um *Web Service*. Se você se lembra, essa chamada pode ser realizada de duas maneiras: por meio do EDI (*Electronic Document Interchange* ou do RPC (*Remote Procedure Call*)). No primeiro caso, trabalharemos com o RPC. Para tanto, a API Axis fornece classes específicas para lidar com tais chamadas a métodos de objetos remotos tratados pelo *Web Service* por meio do servidor *web*, no qual estão disponíveis nossos serviços, que nada mais são que métodos de uma classe. Com isso, tais classes podem prover serviços como a conexão a um banco de dados, ou mesmo o acesso a outra máquina e assim por diante. As possibilidades desse tipo de aplicação são inesgotáveis. Nosso *Web Service* disponibilizará serviços simples de uma calculadora, ou seja, as quatros operações matemáticas básicas, soma, subtração multiplicação e divisão. A resposta à nossa chamada consiste em um arquivo WSDL, conforme visto nos capítulos anteriores. A figura 2 demonstra o cenário para

interoperabilidade com *Web Service* entre nossas aplicações servidoras disponibilizadas na *web* e cliente operando no modo *console*.

Figura 2 *Cenário utilizado para o estudo de caso.*



O primeiro item que necessitamos trabalhar consiste no servidor *web*, no qual estão disponibilizados os serviços. Para isso, você deve instalar o servidor *web* Apache Tomcat. Se você se lembra da disciplina de Programação do quarto período, o Tomcat consiste no servidor, também conhecido como *container*, mais popular para execução e interpretação de serviços *web* que utilizam a linguagem Java como base. No caso, o Tomcat possibilita a interpretação de páginas JSP ou mesmo Servlets, ou ainda como em nosso caso, *Web Services*.

Assim como o servidor Apache Tomcat, outros poderiam ser utilizados para a disponibilização de serviços como o Oracle Application Server ou JBoss. A diferença mais marcante entre esses servidores não consiste no preço, mas no suporte dado aos diversos serviços existentes, já que em sua maioria esses servidores utilizam rotinas do próprio Tomcat. Então, para iniciar, instale o Tomcat.

Saiba mais

O servidor Apache Tomcat pode ser descarregado da internet de maneira gratuita no endereço eletrônico <http://tomcat.apache.org/>. O processo de instalação é simples e intuitivo. Existem diversos tutoriais disponíveis na *web* orientando o processo. Outra importante informação é que o Apache Tomcat, mais que um servidor *web*, é parte de um projeto definido pela organização Apache denominado Jakarta. Nesse projeto, além do Tomcat, são definidas outras ferramentas que possibilitam a criação e a definição de diversos serviços em sua maioria para *web*, como, por exemplo, o Struct que você trabalhou no quarto período na disciplina de Programação. Para mais informações, acesse a página oficial do projeto que está no endereço eletrônico <http://jakarta.apache.org/>.

Após instalado o servidor *web*, é necessário que utilizemos uma API que possibilite que o Tomcat interprete as classes como sendo um *Web Service* e seus

métodos como serviços. E aí que entra em ação a API Apache Axis. Apache Axis é uma implementação para a troca de mensagens que utiliza o protocolo SOAP regido pela especificação definida pela W3C, que vimos no capítulo anterior.

Assim como o SOAP, a API Axis utiliza um *parser* interno para a tradução e o acesso aos documentos XML que trafegam, ou seja, todo o processo será realizado automaticamente como veremos mais adiante.

Além disso, a API Axis é um padrão de mercado estável e é a base para a implementação das maiorias de *Web Services* Java. Para garantir sua continuidade, existe uma comunidade de usuários e várias empresas de grande porte que dão suporte para o projeto. Sua implementação tem por base a implementação da especificação JAX-RPC 1.1, o que facilita todo o processo de troca de mensagens com a camada remota de métodos.

Saiba mais

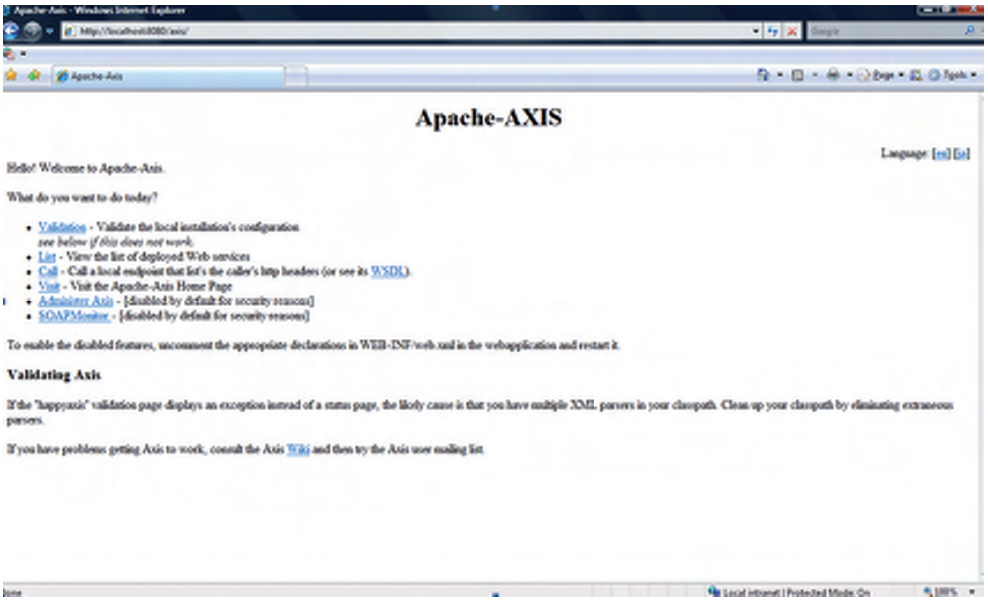
Assim como o projeto Apache Tomcat, a API Axis é um projeto da organização Apache Software Foundation. A diferença é que a Axis é um seguimento de projetos voltados para a criação de *Web Services*, do qual fazem parte outros padrões de mercado como JUDDI e XML-RPC. Toda a documentação, bem como a implementação da API Axis pode ser acessada e descarregada no endereço eletrônico <http://ws.apache.org/axis/>.

De posse da implementação da API Axis, você deve realizar o processo de instalação da API que deve obedecer, segundo Chappell e Jewell (2002), aos seguintes passos:

- realize a descompactação do arquivo descarregado da API Axis;
- copie a pasta *axis* que está dentro da pasta *webapps* da API Axis extraída;
- cole a pasta *axis* na pasta *webapps* do servidor do servidor Tomcat;
- inicie o serviço do servidor Tomcat e faça um teste utilizando, por exemplo, a URL <http://localhost:8080/axis>, caso seu servidor tenha sido instalado na porta padrão 8080.

Pronto! Se tudo correu conforme o esperado, você deve ter visualizado a página informando que o processo de instalação foi concluído com sucesso, conforme é apresentado na figura 3. Na página inicial da API Axis, é possível realizar a validação da instalação por meio de um *link* disponibilizado na parte superior.

Figura 3 *Tela de confirmação da instalação da API Axis no servidor Tomcat.*



Assim estamos prontos para criar o primeiro *Web Service*. Agora vamos ao que interessa! Vamos criar a primeira classe que responderá como um *Web Service* em nosso servidor *web* disponibilizando métodos de uma calculadora. Para isso transcreva a classe Java apresentada no quadro 1.

Quadro 1 *Classe Calculadora.java.*

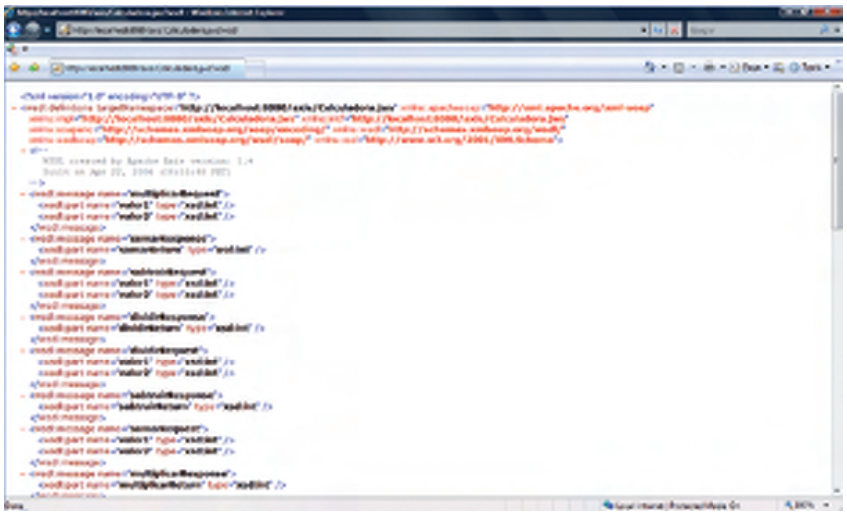
```
public class Calculadora{
      public int somar(int valor1, int valor2){
            return valor1+valor2;
      }
      public int subtrair(int valor1, int valor2){
            return valor1-valor2;
      }
      public int multiplicar(int valor1, int valor2){
            return valor1*valor2;
      }
      public int dividir(int valor1, int valor2){
            if(valor2 != 0)
            return valor1 / valor2;
      return 0;
      }
}
```

Salve o arquivo que você transcreveu como Calculadora.java. Conforme você pode observar, a classe calculadora fornece métodos para realizar as operações matemática básicas. Essa classe será nosso *Web Service*, ou seja, os métodos criados para somar, subtrair, multiplicar e dividir serão serviços que poderão ser acessados por outras aplicações. Para que nossa classe esteja disponível, é necessário que realizemos alguns passos. Segundo Chappell e Jewell (2002), devemos:

- copiar a classe Calculadora.java e a pasta *axis* dentro da pasta *webapps* do servidor Tomcat;
- alterar a extensão de sua classe de ".java" para ".jws".

Pronto! Agora basta você verificar se tudo está funcionando. Para isso, acesse a URL <http://localhost:8080/axis/Calculadora.jws> por meio de seu navegador. Caso tudo esteja realmente funcionando, você verá um *link* para o arquivo WSDL (*Web Services* Description Language), que, conforme vimos no capítulo anterior, é o contrato a ser seguido pelas aplicações que utilizarão o *Web Service*, como é apresentado na figura 4.

Figura 4 *Arquivo WSDL descrevendo o Web Service Calculadora.*

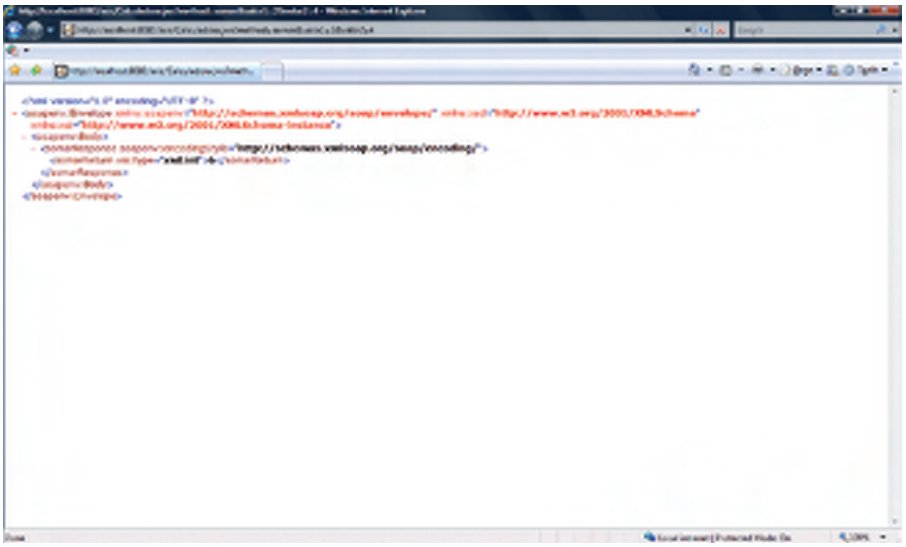


Conforme você deve ter percebido, nosso WSDL é um arquivo XML no qual está descrito o nome do *Web Service*, além da forma de comunicação utilizada, bem como as mensagens SOAP a serem trocadas e os serviços disponibilizados. Note que tudo foi feito de maneira automática, o que para você, desenvolvedor, será algo comercialmente interessante. Caso ocorra algum erro, certifique-se de que todas APIs Java estão disponíveis para o servidor *web*. Um erro muito comum, por exemplo, consiste no fato de que no momento da visualização do arquivo WSDL o servidor não localiza o pacote tools.jar. Para solucionar esse problema, configure as variáveis de ambiente JAVA_HOME e CLASSPATH, conforme apresentado na disciplina de programação do segundo período.

Uma forma de verificar se tudo está funcionando é acessar e passar os parâmetros diretamente no URI do serviço. Em nosso exemplo, poderíamos testar

nosso *Web Service* com o seguinte endereço <http://localhost:8080/axis/Calculadora.jws?method=somar&valor1=2&valor2=4>. Repare que o resultado desse processo consiste em uma mensagem ou, como você conheceu em capítulos anteriores, trata-se do envelope SOAP, como pode ser observado na figura 5.

Figura 5 *Envelope SOAP de resultado do acesso direto ao serviço.*



Agora que já temos nosso *Web Service* funcionando e gerando o arquivo WSDL, necessário para responder a nossa aplicação cliente, bem como definindo as regras a serem seguidas, devemos desenvolver a aplicação que acessará os serviços disponibilizados no servidor *web*. Para isso, temos de seguir alguns passos, assim como fizemos na criação e na configuração de nosso *Web Service*. Veja a seguir os passos a serem dados indicados por Chappell e Jewell (2002).

- Copie os arquivos axis.jar, jaxrpc.jar, commons-logging.jar, commons-discovery.jar, saaj.jar e wsdl4j.jar disponíveis na pasta lib da API Axis que já foi descompactada por você.
- Cole os pacotes copiados no primeiro passo no *path* da biblioteca de sua aplicação para que ela tenha acesso a classes, interfaces e métodos.

Feito isso, estamos prontos para criar a aplicação cliente que interagirá com os serviços disponíveis no *Web Service*. Assim como fizemos na classe Calculadora.java, transcreva a aplicação cliente que é apresentada no quadro 2.

Quadro 2 *Classe ClienteWS.java.*

```
//Classe ClienteWS.java
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
```

```
public class ClienteWS{
  public static void main(String[] args){
    try{
    String url = "http://localhost:8080/axis/Calculadora.jws";
    Object[] params = {new Integer(3), new Integer(3)};
    Service servico = new Service();
    Call chamada = (Call) servico.createCall();
    chamada.setTargetEndpointAddress(url);
    chamada.setOperationName("somar");
    Integer ret = (Integer) chamada.invoke(params);
    System.out.println("Resultado da soma: " + ret);
}
catch(Exception ex){
      ex.printStackTrace();
      }
   }
}
```

Assim como no código do quadro anterior, salve o seu arquivo como ClienteWS.java. Repare que, no código anterior, foi criado um objeto denominado "service", que é o responsável por criar a instância do objeto Call para o objetos "chamada" que utilizarão o RPC para realizar o acesso aos métodos disponibilizados pelo *Web Service*. Note que é necessário que você indique o caminho do serviço na *web* para a instância do objeto Call. Além disso, é necessário que seja informado o método a ser trabalhado por meio do método setOperationName(). Finalizando por meio do método invoke() do objeto chamada são passados os parâmetros e é então retornado o resultado da operação.

Caso você queira lidar com tipos mais complexos que não sejam apenas tipos primitivos, *wrappers* e *string*, é necessário que sejam trabalhados os arquivos WSDD (*Web Service* Deployment Descriptor), que deve conter todas as informações dos serviços que você deseja publicar em seu *Web Service* (ABINADER, 2006). Para tanto, fica a sugestão para que você se aprofunde nesse tipo de configuração que pode ser observada na página oficial da API Axis.

Saiba mais

Nesse momento, a utilização dos *Web Services* pode gerar um problema para a implementação de aplicações clientes. Se você estiver utilizando uma IDE, você provavelmente não gostará de verificar no WSDL a todo

momento quais são os serviços disponíveis em um *Web Service*. Para isso, existe a API WSDL2Java, que auxilia no processo de identificação das operações em um *Web Service* dentro das próprias IDEs. Para mais informações e *download* da API, acesse o sítio da *sourceforge* no endereço <http://sourceforge.net/projects/wsdl2javawizard/>, bem como sua referência em <http://wsdl2javawizard.sourceforge.net/>.

Mas e a interoperabilidade entre aplicações? Calma, trataremos disso agora. O mesmo *Web Service* e seus serviços podem ser acessados por outros tipos de aplicação em outras plataformas e arquiteturas como em uma aplicação em dispositivos móveis que você está estudando na disciplina de Programação para Dispositivos Móveis. Para tanto, nossa aplicação seria algo conforme é apresentado no quadro 3.

Quadro 3 *Classe ClienteMovelWS.java.*

```
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.TextBox;
import org.ksoap.SoapObject;
import org.ksoap.transport.HttpTransport;
public class ClienteMovelWS extends javax.microedition.
midlet.MIDlet {
   private Display display;
   private  String  url  =  "http://localhost:8080/axis/
Calculadora.jws";
   TextBox textbox = null;
public void startApp() {
   display = Display.getDisplay(this);
   try {
      operacaoWebService();
   } catch (Exception ex) {
      System.out.println(ex);
   }
}
public void pauseApp() {}
public void destroyApp(boolean unconditional) {}
public void operacaoWebService() throws Exception {
   StringBuffer stringBuffer = new StringBuffer() ;
   TextBox texto = null;
   SoapObject cliente = new SoapObject(url,"somar");
```

```
        cliente.addProperty("valor1",new Integer(2));
        cliente.addProperty("valor2",new Integer(4));
        HttpTransport ht = new HttpTransport(url,"somar");
        stringBuffer.append("Resultado: " + ht.call(cliente));
        texto = new TextBox("Soma:", stringBuffer.toString(), 1024, 0);
        display.setCurrent(texto);
            }
    }
```

Para que você possa trabalhar com as mensagens SOAP em dispositivos móveis, é necessário que utilize a API KSOAP. Ela disponibiliza classes, interfaces e métodos para acesso, manipulação e interpretação das mensagens SOAP e WSDL em um *Web Service*. Conforme você pôde observar no quadro anterior, a instância cliente da classe SoapObject é a responsável por fazer o acesso a nossa operação de soma no *Web Service*.

Saiba mais

Outra possibilidade de criar *Web Service* com a tecnologia Java seria o uso da API JAX-WS. Esta API é uma das soluções de *software* disponíveis na plataforma J2EE, que utiliza a JAX-RPC. A Sun buscou simplificar ao máximo o desenvolvimento de *Web Services*, dando suporte ao protocolo SOAP, além de fornecer suporte a protocolos adicionais para a *web*. Mais informações sobre como proceder para criar serviços por meio dessa API, visite a página oficial no endereço eletrônico <https://jax-ws.dev.java.net/>.

No sexto capítulo, aprendemos como criar, disponibilizar e acessar serviços em um *Web Service*, bem como vimos na prática o processo de interoperabilidade entre aplicações.

Neste capítulo, vimos conceitos básicos para a criação de *Web Services* com o uso da API Apache Axis, que trabalha com o RPC (*Remote Procedure Call*). Além disso, utilizamos o servidor Web Apache Tomcat para disponibilizar a implementação das funcionalidades de uma calculadora criada com o uso da linguagem de programação Java. Consideramos, também, a utilização dos protocolos SOAP e WSDL nos Web Serivces. Por fim, vimos o processo de interoperabilidade entre duas aplicações que utilizam o *Web Service* com um programa *console* e outro para dispositivos móveis.

No próximo capítulo, veremos os arquivos UDDI, que são de extrema importância, pois é um serviço que possibilita facilidade e rapidez para se encontrar *Web Services* segundo seu interesse.

Referências

ABINADER, Jorge Abílio. ***Web Services* em Java**. Rio de Janeiro: Brasport, 2006.

CHAPPEL, David A.; JEWELL, Tyler. **Java *Web Services***. Sebastopol, USA: O'REILLY, 2002.

Anotações

UDDI – *Universal Description, Discovery and Integration*

7

Introdução

Chegamos ao último capítulo dominando os conceitos básicos para a construção de *Web Services*. A utilização dos *Web Services* pode garantir a interoperabilidade entre aplicações e é extremamente comum dentro de empresas e organizações. Até aqui aprendemos que o XML é o padrão para a definição de protocolos e documentos, o SOAP é o protocolo responsável por transportar e definir nossas mensagens, e que o WSDL é o documento responsável por descrever os *Web Services* e seus respectivos serviços. Você também pode se deparar com situações nas quais seja necessário utilizar serviços e especificações de outras pessoas e empresas. O que fazer? Como proceder para que tais *Web Services* possam ser acessados e integrados à nossa aplicação? Calma, é aí que entra a especificação UDDI. No último capítulo, aprenderemos como criar documentos UDDI, implementar utilizando tais documentos e fazer com que nossos *Web Services* possam ser descritos, descobertos e integrados a sistemas que tenham acesso a tais documentos.

Para uma melhor compreensão desse conteúdo, é interessante que você tenha assimilado os conceitos vistos nos demais capítulos desta disciplina, que domine os conceitos vistos para a construção de documentos XML, garantindo que os *Web Services* sejam localizados e acessados por meio de documentos bem formados e válidos. Além disso, você deve dominar as estruturas e os processo de comunicação existentes entre *Web Services* e aplicações clientes, já que isso é importante para a especificação UDDI.

7.1 Especificação UDDI

Como vimos até aqui, um *Web Service* é um coquetel de tecnologias que possibilitam que possamos trabalhar com a interoperabilidade entre sistemas distribuídos. Um desses diversos produtos que compõem a estrutura do mundo definido para os *Web Services* é a UDDI. Essa especificação é a responsável por registrar sistemas distribuídos pela *web*, ou seja, *Web Service*, está acessível a aplicações de clientes, registra informações sobre como os serviços de um *Web Service* podem ser acessados e o que eles podem fazer. Segundo Chappell e Jewell (2002), os registros UDDI nada mais são que métodos para a padronização da promoção e a descoberta de *Web Services* e de seus serviços disponibilizados na *web*.

A ideia por trás dos registros UDDI está em criar uma plataforma independente, na qual possam ser descritos todos os serviços, descobrir como funcionam,

além do processo a ser utilizado para a integração com aplicações clientes. Assim um registro UDDI serve como um elemento que realiza a mediação entre clientes e servidores e faz com que clientes possam encontrar um fornecedor do serviço que julga apropriado para sua aplicação. A especificação dos registros UDDI foi lançada em 2000 e é mantida pelo consórcio UDDI.org (NEWCOMER, 2002).

Vimos que *Web Services* e seus serviços são a base para o processo de interoperabilidade, que está baseado na arquitetura SOA (*Service-Oriented Architectura*). Tais elementos também são extremamente importantes, pois são utilizados para possibilitar o registro de serviços.

Saiba mais

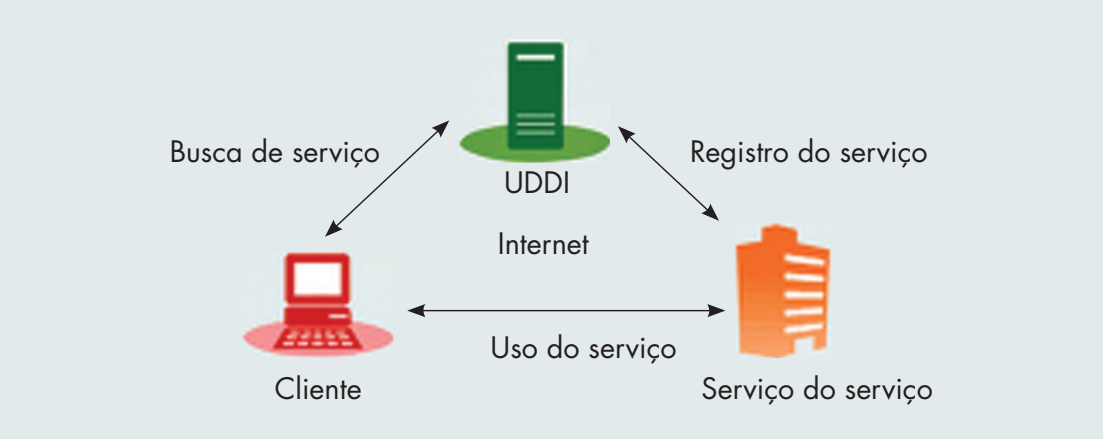
O projeto UDDI é uma iniciativa de diversas empresas que compõem a comunidade UDDI Organization. Esse consórcio é composto por empresas como IBM e Microsoft e se encontra em sua versão 3.0. Esse grupo de trabalho é composto por membros ligados às maiores empresas de *software* do mundo que trabalham no desenvolvimento da especificação dos registros UDDI que, depois de sua oficialização em 2005 como um padrão, não sofreu grandes alterações, está muito bem consolidado no mercado. Mais informações sobre o UDDI podem ser obtidas no endereço eletrônico <http:www.uddi.org>.

Poderíamos citar um serviço de registros UDDI como sendo um tipo de *Web Service* especial, no qual são gerenciadas as informações sobre os provedores, implementações e definições de parâmetros e metadados que são necessários para utilizar os serviços. Segundo Newcomer (2002), a especificação UDDI define alguns elementos importantes para seu funcionamento:

- assim como nos *Web Services*, a comunicação entre os provedores UDDI e as aplicações utiliza o protocolo SOAP para publicar e obter informações de seus registros;
- os registros seguem o esquema XML para definir e manter os dados para especificar todos os serviços de um *Web Service*, como, por exemplo, seu ID e sua descrição;
- são utilizadas as definições dos arquivos WSDL para especificar os serviços que estão disponíveis no registro UDDI.

Entre esses conceitos, não podemos esquecer os protocolos e as ferramentas vistas para desenvolvimento dos *Web Services*, como o SOAP e o WSDL. Entender como estão disponibilizados e como atuam os registros UDDI é extremamente importante para uma melhor compreensão do processo. Na figura 1, é apresentada a estrutura para a divulgação e o consumo dos registros UDDI.

Figura 1 *Utilização de registros UDDI.*



Fonte: Newcomer (2002).

Como pode ser observado na figura 1, de um lado temos um provedor de serviços e de outro um cliente, ambos fazem acesso a um diretório, ou mesmo a um servidor de registros, que possibilita tanto a consulta quanto o registro de novos serviços. Conceitualmente, os serviços de um Web Serviçe têm suas informações registradas utilizando métodos e padrões dos registros UDDI.

Poderíamos fazer uma analogia dos registros UDDI e provedores UDDI com uma página de pesquisa, na qual um determinado usuário realiza a busca e localiza a página que melhor lhe convém. Assim a especificação UDDI descreve a oferta dos serviços de uma maneira bem familiar, como no caso de uma lista telefônica. Segundo Newcomer (2002), tal sumário provê uma classificação para os registros.

- **Páginas brancas**: basicamente contêm o contato e a identificação sobre a empresa que fornece algum serviço. Entre os dados disponíveis, podem ser encontrados nome, endereço e telefone, permitindo que outras pessoas possam descobrir seu *Web Service* e assim manter contato com você.
- **Páginas amarelas**: são informações que descrevem um *Web Service* e utilizam a categorização comercial ou demograficamente, possibilitando que, com essas informações, seja possível localizar seu *Web Service*.
- **Páginas verdes**: são informações técnicas que descrevem o comportamento e as funções suportadas em um *Web Service* e seus serviços, organizados por tipo de negócio. Tais informações são normalmente a localização do servidor *web*, bem como dados de como proceder para que seja possível a comunicação e interação entre cliente e servidor.

Mas o ponto central do processo está no fato de que os registros UDDI nada mais são que descrições implementadas que utilizam o formato XML.

Se organizássemos tudo o que vimos até aqui nesta disciplina, no caso, todos os processos e os protocolos, teríamos uma pilha de elementos e serviços

que possibilitam a utilização dos *Web Services* e, com isso, a interoperabilidade entre aplicações. Observemos esses elementos dispostos no quadro 1.

Quadro 1 *Pilha de execução de um Web Service.*

PILHA DE INTEROPERABILIDADE	Interoperabilidade entre serviços
	Universal Description, Discovery Integration (UDDI)
	Simple Object Access Protocol
	Extensible Markup Language (XML)
	Protocolos comuns da internet (HTTP, TCP/IP)

Fonte: Chappell e Jewell (2002).

Observando o que foi apresentado no quadro 1, você deve ter compreendido que os registros UDDI são a última camada na pilha de execução de um *Web Service*. Conforme pode ser observado ainda dentro da pilha de execução, eles estão baseados em tecnologias padrões, como TCP/IP, HTTP, XML, SOAP e WSDL para a criação de serviços uniformes descritos formalmente.

Mas outros detalhes foram sendo considerados com o passar dos anos, pelas organizações responsáveis e, com isso, passou a se observar que em sua maioria os *Web Services* eram utilizados de maneira interna dentro de empresas e organizações. Percebeu-se uma notória necessidade de definir e controlar a visibilidade dos *Web Services*, algo natural para algumas linguagens de programação como Java. No *release* 3.0 da especificação UDDI, foram especificados alguns padrões que permitem a definição de acessos a implementações com determinadas restrições a serviços (ABINADER, 2006). O quadro 2 apresenta os tipos de visibilidade previstos para os registros UDDI.

Quadro 2 *Tipos de visibilidade para a especificação UDDI 3.0.*

TIPO DE REGISTRO	DESCRIÇÃO	ABRANGÊNCIA
Privado	Consiste em um registro que geralmente é protegido por um *firewall*, sendo acessado apenas internamente, estando isolado de uma rede de maior alcance como a internet. Isso, com certeza, possibilita acesso a serviços com maior segurança já que os dados não estão compartilhados.	Intranet
Semiprivado	São os registros disponibilizados em um ambiente do qual se tem controle. Os serviços estão disponíveis apenas para entidades externas confiáveis das quais é possível obter controle das ações e dos passos. Esse tipo de registro pode ser acessado em redes que já tenham alguma interação com o mundo externo com a internet.	Extranet

TIPO DE REGISTRO	DESCRIÇÃO	ABRANGÊNCIA
Público	Neste tipo de registro, todos os serviços estão disponibilizados. Assim um registro público está disponível para acesso na internet para utilização livre. Não que isso caracterize algo descontrolado, muito pelo contrário, é aí que geralmente existe um maior controle e segurança sobre os itens que estão sendo disponibilizados, principalmente no que tange as funções administrativas.	Internet

Fonte: Newcomer (2002).

A partir do que foi apresentado no quadro, é possível definir uma estrutura de serviços de um *Web Service*. Nessa estrutura, são controlados os itens que podem ou não estar disponíveis para os clientes. Temos repositórios de serviços mais conhecidos como UDDI Business Registry (UBR), no qual é possível se definir um diretório de serviços disponibilizados nos registros e que podem ser acessados para se publicar ou mesmo realizar a solicitação de informações de um *Web Service* (NEWCOMER, 2002).

Como já citamos, o UDDI está baseado na tecnologia XML, o que fornece uma estrutura neutra no qual seus dados podem ser especificados para descrever todas as relações existentes na definição dos serviços e que devem, de alguma maneira, compor a estrutura de uma publicação UDDI.

Saiba mais

A OASIS (Organization for the Advancement of Structured Information Standards), ou ainda, em português, Organização para o Avanço de Padrões em Informação Estruturada, é um consórcio global que conduz o desenvolvimento, a convergência e a adoção de padrões para *e-business* e *Web Services*. Entre os membros da organização, estão grandes nomes da indústria de tecnologia da informação, como IBM, SAP AG e Sun Microsystems. A OASIS (s/d) desenvolve especificações e padrões relacionados com áreas como: *supply chain*, SOA, *Web Services*, processamento de XML e segurança da informação. Desenvolve também manuais, guias de melhores práticas e ferramentas para facilitar a adoção de padrões para informação estruturada. Além disso, estabelece fóruns de discussão para incentivar a adoção de padrões e o desenvolvimento relacionado à troca de informação eletrônica em governos e empresas. Você pode encontrar mais informações sobre essa organização no sítio <http://www.oasis-open.org/>.

Assim a estrutura dos registros UDDI é composta por alguns tipos de elementos: entidade de negócio definido por meio das marcações <business-

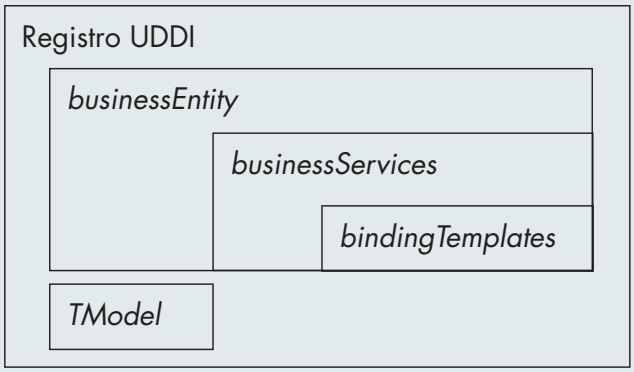
Service>, entidade serviço negócio que são descritos no elemento <businessEntity>, *template* de ligação definido utilizando a *tag* <bindingTemplate> e finalizando o tModel. Além disso, com novos *releases*, dois novos elementos surgiram para melhorar o processo de interligação: publisherAssertion e subscription (NEWCOMER, 2002). Os registros de um serviço disponibilizado por um *Web Service* podem conter uma ou mais especificações de ligação, ou seja, diversos *templates* de ligação para o processo de comunicação entre as aplicações. A partir de Chappell e Jewell (2002), vejamos para que servem cada um desses elementos.

- A marcação *businessService* é o elemento responsável por descrever a função de um serviço que tem alguns atributos e parâmetros que indicam a categoria da qual o *Web Service* faz parte como o businessKey e name.
- A marcação *businessEntity* é um elemento no qual é representado o provedor de um *Web Service*, são apresentados descritores de informações importantes como dados de contato, categoria, serviços oferecidos, identificadores de negócio de uma empresa.
- A marcação *bindingTemplate* é o elemento responsável por fazer referência a detalhes técnicos do serviço como endereço. A função chave do *template* de ligação é oferecer e expor os tipos e serviços de transporte disponíveis, considerando os protocolos HTTP, HTTPS e SMTP, entre outros.
- A marcação *TModel* trabalha com qualquer tipo de dado que pode ser registrado, independente de sua taxonomia, transporte, conexões ou mesmo protocolos. Assim geralmente é comum que registros UDDI tenham informações sobre as especificações com a utilização de metadados projetada para suportar características das mais variadas, como, por exemplo, agregar o arquivo WSDL ao registro sendo extremamente flexível para descrever quase todos os tipos de dados.
- A marcação *publisherAssertion* surgiu a partir do *release* 2.0 da especificação UDDI com o intuito de criar e possibilitar o relacionamento entre várias entidades no registro.

A marcação subscription, assim como o elemento publisherAssertion, surgiu na versão 2.0 da UDDI e é responsável por localizar e identificar algumas mudanças nos registros de um *Web Service*.

O resultado de tais elementos consiste em um registro UDDI baseado no padrão XML, no qual cada uma dessas marcações define uma parte do processo, desde a comunicação até as informações básicas como a de quem desenvolveu o serviço (DAUM; MERTEN, 2002). A figura 2 apresenta a estrutura de um registro UDDI.

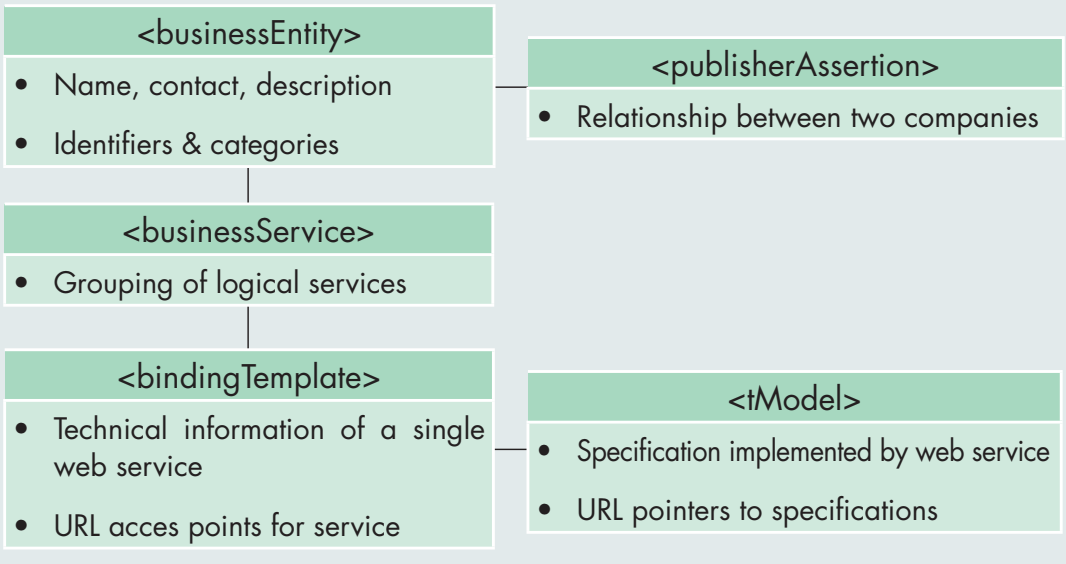
Figura 2 *Estrutura de um registro UDDI.*



Fonte: Chappell e Jewell (2002).

Como vimos na figura 2, cada um desses elementos tem um relacionamento intrínseco, bem como dados importantes para o processo. O modelo de dados UDDI é definido por meio da interação de todas as marcações existentes em um registro no qual são expostos os serviços e os dados (DAUM; MERTEN, 2002). A figura 3 é a responsável por apresentar graficamente o modelo de dados dos registros UDDI.

Figura 3 *Modelo de dados UDDI.*



Fonte: Chappell e Jewell (2002).

Conhecemos a estrutura básica dos registros UDDI. Você deve ter reparado que essa estrutura não é algo tão natural de ser desenvolvido. Então nada mais interessante que a existência de ferramentas que automatizem todo o processo

de criação dessa especificação. Para tanto, podem ser citadas algumas APIS que facilitam o processo de registro e divulgação de *Web Services* e seus serviços, como no caso da plataforma .Net da Microsoft que disponibiliza uma interface para manipulação UDDI (CHAPPELL; JEWELL, 2002).

Outro exemplo é a API JUDDI, que comercialmente é definida como um padrão para registro de *Web Services* criados em Java. A JUDDI é uma implementação *open source* Java para a criação de registros que obedecem à especificação UDDI independente de plataforma ou mesmo servidor *web*, que é necessário para a manipulação de um sistema de armazenamento de dados, bem como o suporte à tecnologia Java.

Disponibilizado pela Apache, ele consiste em uma plataforma para a divulgação de registros UDDI e atualmente está em sua versão 3.0. A facilidade de integração aos provedores é uma de suas principais características. Mais informações sobre a plataforma e seu processo de instalação podem ser obtidas no seguinte endereço eletrônico <http://ws.apache.org/juddi/>.

No último capítulo, foram considerados os conceitos básicos dos registros UDDI. Vimos que a especificação UDDI disponibiliza métodos para a padronização da promoção e da descoberta dos *Web Services*. Vimos também que os registros se utilizam da notação análoga a uma lista telefônica para descrever seus registros, bem como para tratar sua visibilidade. Além disso, conhecemos os principais elementos que compõem a estrutura de um registro UDDI, verificando que o mesmo nada mais é que um documento que segue o padrão XML. Por fim, comentamos algumas ferramentas que facilitam a criação e a divulgação dos registros como a API JUDDI.

Referências

ABINADER, Jorge Abílio. ***Web Services* em Java**. Rio de Janeiro: Brasport, 2006.

CHAPPEL, David A.; JEWELL, Tyler. **Java *Web Services***. Sebastopol, USA: O'REILLY, 2002.

DAUM, Berthold; MERTEN, Udo. **Arquitetura de sistemas com XML**. Rio de Janeiro: Campus, 2002.

MENDES, Antônio. **Programando com XML**. Rio de Janeiro: Campus, 2004.

NEWCOMER, Eric. **Understanding Web services**: XML, WSDL, SOAP and UDDI. Indianapolis, USA: Addison-Wesley Professional, 2002.

Anotações