9

Serviços Web

Um serviço Web (*Web service*) fornece uma interface de serviço que permite aos clientes interagirem com servidores de uma maneira mais geral do que acontece com os navegadores Web. Os clientes acessam operações de um serviço Web por meio de requisições e respostas formatadas em XML e, normalmente, transmitidas por HTTP. Os serviços Web podem ser acessados de uma maneira mais *ad hoc* do que os serviços baseados em CORBA, permitindo que eles sejam mais facilmente usados em aplicações de Internet.

Assim como no CORBA e em Java, as interfaces dos serviços Web podem ser descritas em uma IDL. Entretanto, para os serviços Web, informações adicionais precisam ser descritas, incluindo a codificação e os protocolos de comunicação em uso e o local do serviço.

Os usuários exigem uma maneira segura de criar, armazenar e modificar documentos e trocá-los pela Internet. Os canais seguros TLS (Transport Layer Security, descrito no Capítulo 9) não fornecem todos os requisitos necessários. A segurança XML se destina a suprir essa falta.

Os serviços Web são cada vez mais importantes nos sistemas distribuídos: eles suportam atividade conjunta na Internet global, incluindo a área fundamental da integração de empresa para empresa (*business-to-business*, B2B) e também a emergente cultura de "*mashup*", permitindo que desenvolvedores criem *software* inovador em cima da base de serviços já existente. Os serviços Web também fornecem o *middleware* subjacente para a computação de grade (*grid*) e em nuvem.

9.1 Introdução

O crescimento da Web nas últimas duas décadas (veja a Figura 1.6) prova a eficácia do uso de protocolos simples na Internet, como base para um grande número de serviços e aplicações remotos. Em particular, o protocolo de requisição-resposta HTTP (Seção 5.2) permite que clientes de propósito geral, chamados de navegadores, vejam páginas Web e outros recursos com referência aos seus URLs. Veja uma nota, no quadro a seguir, sobre URIs, URLs e URNs.

Entretanto, o uso de um navegador de propósito geral como cliente, mesmo com as melhorias fornecidas por *applets* específicos de uma aplicação, baixados por *download*, restringe a abrangência potencial dessas aplicações. No modelo cliente-servidor original, tanto o cliente como o servidor eram funcionalmente especializados. Os serviços Web (*Web services*) retornam a esse modelo, no qual um cliente específico da aplicação interage pela Internet com um serviço que possui uma interface funcionalmente especializada.

Assim, os serviços Web fornecem infraestrutura para manter uma forma mais rica e mais estruturada de interoperabilidade entre clientes e servidores. Eles fornecem uma base por meio da qual um programa cliente em uma organização pode interagir com um servidor em outra organização, sem supervisão humana. Em particular, os serviços Web permitem o desenvolvimento de aplicações complexas, fornecendo serviços que integram vários outros serviços. Devido à generalidade de suas interações, os serviços Web não podem ser acessados diretamente pelos navegadores.

O fornecimento de serviços Web como um acréscimo aos servidores Web é baseado na capacidade de usar uma requisição HTTP para provocar a execução de um programa. Lembre-se de que, em uma requisição HTTP, quando um URL se refere a um programa executável – por exemplo, uma busca – o resultado é produzido por esse programa e retornado. De maneira semelhante, os serviços Web são uma extensão da Web e podem ser fornecidos por servidores Web. Entretanto, seus servidores não precisam ser servidores Web. Os termos *servidor Web* e *serviços Web* não devem ser confundidos: um servidor Web fornece um serviço HTTP básico, enquanto um serviço Web fornece um serviço baseado nas operações definidas em sua interface.

A representação de dados externa e o empacotamento das mensagens trocadas entre clientes e serviços Web são feitos em XML, que foi descrita na Seção 4.3.3. Para recapitular, XML é uma representação textual que, embora mais volumosa do que as representações alternativas, foi adotada por sua legibilidade e pela consequente facilidade de depuração.

O protocolo SOAP (Seção 9.2.1) especifica as regras de uso da XML para empacotar mensagens, por exemplo, para suportar um protocolo de requisição-resposta. A Figura 9.1 resume os principais pontos a respeito da arquitetura de comunicação em que os serviços Web operam: um serviço Web é identificado por um URI e pode ser acessado pelos clientes usando mensagens formatadas em XML. O protocolo SOAP é usado para

URI, URL e URN • O URI (Uniform Resource Identifier) é um identificador de recurso geral, cujo valor pode ser um URL ou um URN. O URL, que inclui informações de localização do recurso, como o nome de domínio do servidor de um recurso que está sendo nomeado, é bem conhecido de todos os usuários da Web. Os URNs (Uniform Resource Names) são independentes da localização – eles contam com um serviço de pesquisa para fazer o mapeamento para os URLs dos recursos. Os URNs serão discutidos com mais detalhes na Seção 13.1.

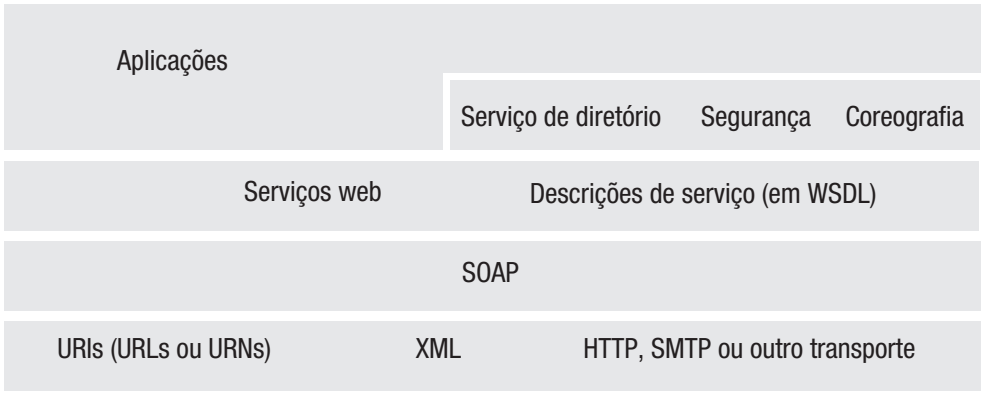


Figura 9.1 Infraestrutura e componentes dos serviços Web.

encapsular essas mensagens e transmiti-las por HTTP ou outro protocolo, por exemplo, TCP ou SMTP. Um serviço Web divulga para potenciais clientes a interface e outros aspectos dos serviços que implementa por meio das descrições de serviço.

A camada superior da Figura 9.1 ilustra o seguinte:

- Os serviços e aplicações Web podem ser construídos em cima de outros serviços Web.
- Alguns serviços Web fornecem a funcionalidade geral exigida para a operação de um grande número de outros serviços Web. Eles incluem os serviços de diretório, segurança e coreografia, todos os quais serão discutidos posteriormente neste capítulo.

Geralmente, um serviço Web fornece uma *descrição do serviço*, a qual inclui uma definição de interface e outras informações, como o URL do servidor. Isso é usado como base para um entendimento comum entre cliente e servidor quanto ao serviço oferecido. A Seção 9.3 apresentará a WSDL (Web Services Description Language).

Outra necessidade comum no *middleware* é um serviço de atribuição de nomes, ou de diretório, para permitir que os clientes descubram serviços. Os clientes de serviços Web têm necessidades semelhantes, mas frequentemente saem-se bem sem os serviços de diretório. Por exemplo, frequentemente, eles descobrem serviços a partir das informações presentes em uma página Web, por exemplo, como resultado de uma busca no Google. Entretanto, algum trabalho precisa ser feito para fornecer um serviço de diretório que seja conveniente para uso dentro das organizações. Isso será discutido na Seção 9.4.

A segurança em XML será apresentada na Seção 9.5. Nessa estratégia de segurança, documentos ou partes de documentos podem ser assinados ou cifrados. Um documento que possui elementos assinados ou cifrados pode, então, ser transmitido ou armazenado; posteriormente, podem ser feitas adições e estas também poderão ser assinadas ou cifradas.

Os serviços Web dão acesso a recursos de clientes remotos, mas não fornecem uma maneira de coordenar suas operações mútuas. A Seção 9.6 discutirá a coreografia dos serviços Web, a qual se destina a permitir que um serviço Web utilize padrões de acesso predefinidos no uso de um conjunto de outros serviços Web.

A última seção deste capítulo considera aplicações de serviços Web, incluindo o suporte para arquitetura orientada a serviços, a computação em grade e a computação em nuvem.

9.2 Serviços Web

Geralmente, uma interface de serviço Web consiste em um conjunto de operações que podem ser usadas por um cliente na Internet. As operações de um serviço Web podem ser fornecidas por uma variedade de recursos diferentes, por exemplo, programas, objetos ou bancos de dados. Um serviço Web pode ser gerenciado por um servidor Web, junto a páginas Web, ou pode ser um serviço totalmente separado.

A principal característica da maioria dos serviços Web é que eles podem processar mensagens SOAP formatadas em XML (veja a Seção 9.2.1). Uma alternativa é a estratégia REST, que está descrita em linhas gerais a seguir. Cada serviço Web usa sua própria descrição para tratar das características específicas das mensagens que recebe. Para uma boa narrativa de aspectos mais detalhados dos serviços Web, consulte Newcomer [2002] ou Alonso *et al.* [2004].

Muitos servidores Web comerciais conhecidos, incluindo Amazon, Yahoo, Google e eBay, oferecem interfaces de serviço que permitem aos clientes manipular seus recursos Web. Como exemplo, o serviço Web oferecido pela Amazon.com fornece operações que permitem aos clientes obter informações sobre produtos, adicionar um item a um carrinho de compras ou verificar o status de uma transação. Os serviços Web da Amazon [associates.amazon.com] podem ser acessado por SOAP ou por REST. Isso permite que aplicações de outros fornecedores construam serviços com valor agregado sobre aqueles fornecidos pela Amazon.com. Por exemplo, uma aplicação de controle de inventário e aquisição poderia pedir o fornecimento de mercadorias da Amazon.com, à medida que elas fossem necessárias, e controlar automaticamente a mudança de status de cada requisição. Mais de 50.000 desenvolvedores se registraram para uso desses serviços Web nos dois primeiros anos após eles terem sido introduzidos [Greenfield e Dornan 2004].

Outro exemplo interessante de aplicação que exige a presença de um serviço Web é a que implementa *sniping* em leilões da eBay. *Sniping* significa fazer um lance durante os últimos segundos antes que um leilão termine. Embora os seres humanos possam realizar as mesmas ações, por meio da interação direta com a página Web, eles não podem fazer isso com tanta rapidez.

Combinação de serviços Web • O fornecimento de uma interface de serviço permite que suas operações sejam combinadas com as de outros serviços para fornecer nova funcionalidade (veja também a Seção 9.7.1). A aplicação de aquisição mencionada anteriormente também poderia estar usando outros fornecedores. Como outro exemplo das vantagens de combinar vários serviços, considere o fato de que atualmente as pessoas utilizam seus navegadores para fazer reservas de passagens aéreas e de hotéis e para alugar carros com uma seleção de *sites* Web diferentes. Entretanto, se cada um desses *sites* Web fornecesse uma interface de serviço padrão, um serviço de agente de viagens poderia usar suas operações para fornecer ao viajante uma combinação desses serviços. Esse ponto está ilustrado na Figura 9.2.

Padrões de comunicação • O serviço de agente de viagens ilustra o possível uso dos dois padrões de comunicação alternativos disponíveis nos serviços Web:

- O processamento de uma reserva demora um tempo longo para terminar, e bem poderia ser suportado por uma troca assíncrona de documentos, começando com os detalhes das datas e destinos, seguida do retorno das informações de status de tempos em tempos e, finalmente, dos detalhes da conclusão. O desempenho não é problema neste caso.

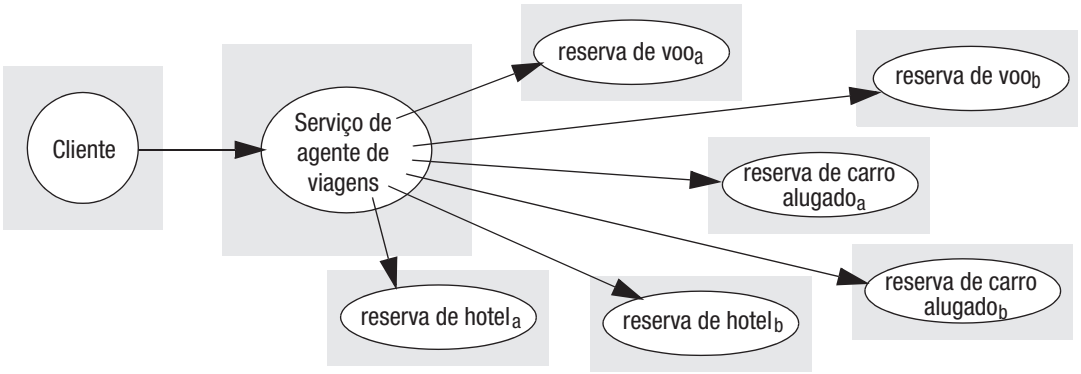


Figura 9.2 O serviço de agente de viagens combina vários serviços Web.

- A verificação dos detalhes do cartão de crédito e as interações com o cliente devem ser fornecidas por um protocolo de requisição-resposta.

Em geral, os serviços Web usam um padrão de comunicação de requisição-resposta síncrona com seus clientes, ou se comunicam por meio de mensagens assíncronas. Este último estilo de comunicação pode ser usado mesmo quando as requisições exigem respostas, no caso em que o cliente envia uma requisição e posteriormente recebe a resposta de forma assíncrona. Um padrão de estilo evento também pode ser usado: por exemplo, os clientes de um serviço de diretório podem se registrar nos eventos de interesse e serão notificados quando certos eventos ocorrerem. Por exemplo, um evento poderia ser a chegada ou a saída de um serviço.

Para possibilitar o uso de uma variedade de padrões de comunicação, o protocolo SOAP (Seção 9.2.1) é baseado no empacotamento de mensagens unidirecionais únicas. Ele suporta interações de requisição-resposta usando pares de mensagens únicas e especificando como irá representar as operações, seus argumentos e resultados.

Mais geralmente, os serviços Web são projetados para suportar computação distribuída na Internet, na qual coexistem diferentes linguagens de programação e paradigmas. Assim, eles são projetados para serem independentes de qualquer paradigma de programação em particular. Isso contrasta, por exemplo, com os objetos distribuídos, que defendem um paradigma de programação bastante específico para o desenvolvedores (uma discussão mais a fundo sobre as distinções entre os serviços Web e os objetos distribuídos pode ser encontrada na Seção 9.2.2).

Baixo acoplamento • Há um interesse considerável no *baixo acoplamento* em sistemas distribuídos, particularmente na comunidade de serviços Web. Contudo, é comum a terminologia ser mal definida e imprecisa. No contexto de serviços Web, baixo acoplamento se refere a minimizar as dependências entre os serviços para se ter uma arquitetura subjacente flexível (reduzindo o risco de uma alteração em um serviço causar uma reação em cadeia em outros serviços). Isso é parcialmente suportado pela independência pretendida pelos serviços Web, com a intenção subsequente de produzir combinações de serviços Web, conforme discutido anteriormente. Entretanto, o baixo acoplamento é melhorado por meio de vários recursos adicionais:

- A programação com interfaces (discutida no Capítulo 5) fornece um nível de baixo acoplamento, separando a interface de sua implementação (suportando também áreas de heterogeneidade importantes – por exemplo, na escolha da linguagem de

programação e da plataforma usadas). A programação com interfaces é adotada pela maioria dos paradigmas de sistemas distribuídos, incluindo objetos distribuídos e componentes (discutidos no Capítulo 8), assim como serviços Web.

- Há uma tendência em usar interfaces simples e genéricas em sistemas distribuídos, e isso é exemplificado pela interface mínima oferecida pela World Wide Web e pela estratégia REST nos serviços Web. Essa estratégia contribui para o baixo acoplamento por reduzir a dependência em relação a nomes de operação específicos (o estudo de caso do Google, no Capítulo 21, fornece um bom exemplo desse estilo de programação distribuída). Uma consequência disso é que os dados se tornam mais importantes do que a operação, com a semântica da operação conjunta frequentemente mantida nos dados (por exemplo, a definição de documento XML associada em serviços Web); essa visão *orientada a dados* é discutida mais a fundo no contexto dos sistemas móveis na Seção 19.3.1.
- Conforme mencionado anteriormente, os serviços Web podem ser usados com uma variedade de paradigmas de comunicação, incluindo comunicação por requisição-resposta, troca de mensagens assíncrona ou mesmo paradigmas de comunicação indireta (conforme apresentado no Capítulo 6). O nível de acoplamento é afetado diretamente por essa escolha. Por exemplo, na comunicação por requisição-resposta, os dois participantes são intrinsecamente acoplados; a troca de mensagens assíncrona oferece certo grau de desacoplamento (referido no Capítulo 6 como desacoplamento em relação ao sincronismo), enquanto a comunicação indireta também oferece desacoplamento em relação ao tempo e ao espaço.

Em conclusão, existem várias dimensões no baixo acoplamento, e é importante ter isso em mente ao se utilizar o termo. Os serviços Web suportam intrinsecamente um nível de baixo acoplamento, devido à filosofia de projeto adotada e à estratégia de programação com interfaces usada. Isso pode ser melhorado por escolhas de projeto adicionais, incluindo a adoção da estratégia REST e o uso de comunicação indireta.

Representação de mensagens • Tanto o protocolo SOAP quanto os dados que ele transporta são representados em XML – um formato textual auto-descritivo apresentado na Seção 4.3.3. As representações textuais ocupam mais espaço do que as binárias e exigem mais tempo para seu processamento. Nas interações de estilo documento, a velocidade não é problema, mas ela é importante nas interações tipo requisição-resposta. Entretanto, pode ser argumentado que há uma vantagem em ser um formato legível para seres humanos, que facilita a construção de mensagens simples e a depuração das mais complexas. Ele também permite que um usuário veja o texto de uma mensagem na sua frente. No entanto, existem situações em que ela é lenta demais.

REST (REpresentational State Transfer) • REST [Fielding 2000] é uma estratégia com um estilo de operação muito restrito, no qual os clientes usam URLs e as operações HTTP *GET*, *PUT*, *DELETE* e *POST* para manipular recursos representados em XML. A ênfase está na manipulação de recursos de dados, em vez de interfaces. Quando um novo recurso é criado, ele recebe um novo URL por meio do qual pode ser acessado ou atualizado. Os clientes recebem o estado inteiro de um recurso, em vez de chamar uma operação para fornecer alguma parte dele. Fielding acredita que, no contexto da Internet, a proliferação de diferentes interfaces de serviço não será tão útil quanto um conjunto mínimo de operações simples e uniformes. É interessante notar que, de acordo com Greenfield e Dornan [2004], 80% dos pedidos para os serviços Web na Amazon.com são feitos por intermédio da interface REST, com os 20% restantes usando SOAP.

Cada item em uma descrição em XML é anotado com seu tipo, e o significado de cada tipo é definido por um esquema referenciado dentro da descrição. Isso torna o formato extensível, permitindo que qualquer tipo de dados seja transportado. Não há limite para a riqueza e complexidade em potencial dos documentos formatados em XML, mas poderia haver um problema na interpretação daqueles que se tornassem excessivamente complexos.

Referências de serviço • Em geral, cada serviço Web tem um URI, o qual os clientes utilizam para se referirem a ele. O URL é a forma mais frequentemente usada de URI. Como um URL contém o nome de domínio de um computador, o serviço ao qual ele se refere sempre será acessado nesse computador. Entretanto, o ponto de acesso de um serviço Web com um URN pode depender do contexto e mudar de tempos em tempos – seu URL atual pode ser obtido a partir de um serviço de pesquisa de URN. Essa referência de serviço é conhecida nos serviços Web como *destino final* (*endpoint*).

Ativação de serviços • Um serviço Web será acessado por meio do computador cujo nome de domínio está incluído em seu URL corrente. Esse computador pode, ele próprio, executar o serviço Web ou executá-lo em outro computador servidor. Por exemplo, um provedor de serviço com dezenas de milhares de clientes precisará implantar centenas de computadores para fornecer esse serviço. Um serviço Web pode funcionar continuamente ou ser ativado sob demanda. O URL é uma referência persistente – significando que ele continuará a se referir ao serviço enquanto esse servidor existir.

Transparência • Uma tarefa importante de muitas plataformas de *middleware* é proteger o programador dos detalhes da representação e do empacotamento dos dados e, às vezes, fazer as invocações remotas parecerem ser locais. Nada disso é fornecido como parte de uma infraestrutura ou plataforma de *middleware* para serviços Web. No nível mais simples, clientes e servidores podem ler e gravar suas mensagens diretamente em SOAP, usando XML.

Porém, por conveniência, os detalhes do protocolo SOAP e da XML geralmente são ocultos por uma API local, em uma linguagem de programação como Java, Perl, Python ou C++. Nesse caso, a descrição do serviço pode ser usada como base para a geração automática dos procedimentos de empacotamento e desempacotamento necessários.

Proxies: uma opção para ocultar a diferença entre chamadas locais e remotas é fornecer um *proxy* cliente ou um conjunto de procedimentos *stub*. A Seção 9.2.3 explicará como isso é feito em Java. Os *proxies* clientes, ou *stubs*, fornecem uma forma estática de invocação na qual a estrutura de cada chamada e os procedimentos de empacotamento são gerados antes que quaisquer invocações sejam feitas.

Invocação dinâmica: uma alternativa para os *proxies* é fornecer aos clientes uma operação genérica para ser usada independentemente do procedimento remoto a ser chamado, semelhante ao procedimento *DoOperation* definido na Figura 5.3 (mas sem o primeiro argumento). Nesse caso, o cliente especifica o nome de uma operação e seus argumentos, e eles são convertidos dinamicamente para SOAP e XML. A comunicação assíncrona de mensagens únicas pode ser obtida de maneira semelhante, fornecendo-se aos clientes operações genéricas para enviar e receber mensagens.

9.2.1 SOAP

O protocolo SOAP (Simple Object Access Protocol) é projetado para permitir tanto interação cliente-servidor como assíncrona pela Internet. Ele define um esquema para uso da XML para representar o conteúdo de mensagens de requisição-resposta (veja a Figura 5.4), assim como um esquema para a comunicação de documentos. Originalmente, o

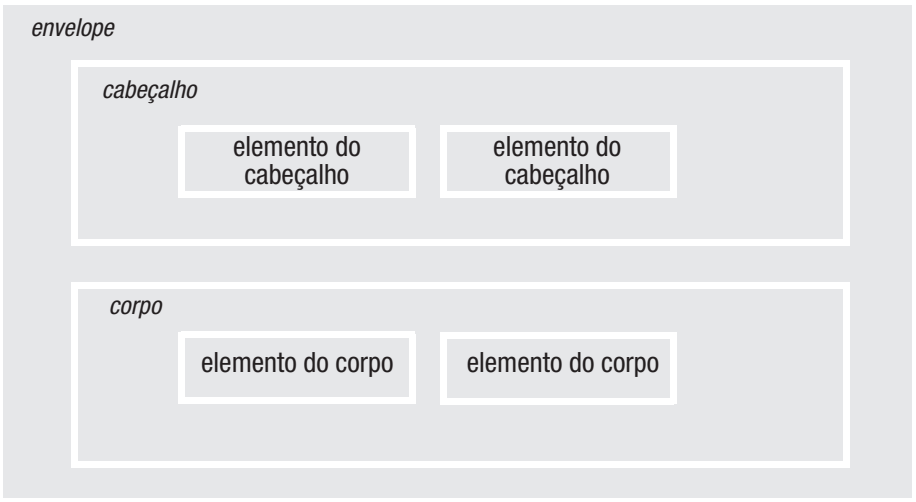


Figura 9.3 Mensagem SOAP em um envelope.

protocolo SOAP era baseado apenas em HTTP, mas a versão atual é projetada para usar uma variedade de protocolos de transporte*, incluindo SMTP, TCP ou UDP. A descrição desta seção é baseada no protocolo SOAP versão 1.2 [www.w3.org IX], que é uma recomendação do W3C (World Wide Web Consortium). O protocolo SOAP é uma extensão do XML-RPC da Userland [Winer 1999].

A especificação do protocolo SOAP declara:

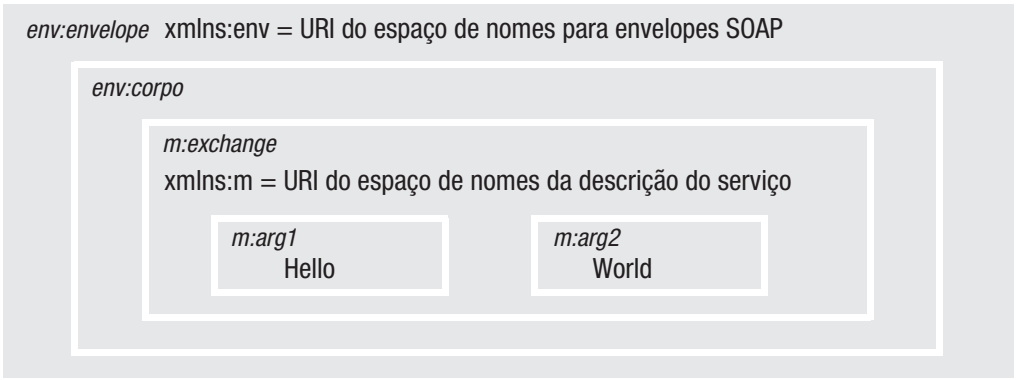
- como a XML deve ser usada para representar o conteúdo de mensagens individuais;
- como duas mensagens podem ser combinadas para produzir um padrão de requisição-resposta;
- as regras sobre como os destinatários das mensagens devem processar os elementos XML que elas contêm;
- como HTTP e SMTP devem ser usados para comunicar mensagens SOAP. É esperado que as versões futuras da especificação definam como usar outros protocolos de transporte, por exemplo, TCP.

Esta seção descreve como o protocolo SOAP usa XML para representar mensagens e HTTP para comunicação. Entretanto, normalmente o programador não precisa se preocupar com esses detalhes, pois APIs SOAP foram implementadas em muitas linguagens de programação, incluindo Java, Javascript, Perl, Python.NET, C, C++, C# e Visual Basic.

Para suportar comunicação cliente-servidor, o protocolo SOAP especifica como se faz para usar o método HTTP *POST* para a mensagem de requisição e para a mensagem de resposta. O uso combinado de XML e HTTP fornece um protocolo padrão para comunicação cliente-servidor pela Internet.

Pretende-se que uma mensagem SOAP possa ser passada por meio de intermediários no caminho para o computador que gerencia o recurso a ser acessado, e que serviços de *middleware* de nível mais alto, como transações ou segurança, possam usar esses intermediários para realizar o processamento.

* N. de R.T.: O leitor habituado com o modelo de referência OSI (Open System Interconnection), comumente usado na área de redes de computadores, pode estranhar a menção aos protocolos HTTP e SMTP como protocolos de transporte. Entretanto, em serviços Web, o termo *protocolo de transporte* é empregado para referenciar qualquer protocolo que sirva como "meio de transporte" para uma mensagem SOAP.



Nesta figura e na próxima, cada elemento XML é representado por uma caixa com seu nome em itálico, seguido dos atributos e de seu conteúdo.

Figura 9.4 Exemplo de uma requisição simples sem cabeçalhos.

Mensagens SOAP • Uma mensagem SOAP é transportada em um *envelope*. Dentro do envelope existe um cabeçalho opcional e um corpo, como mostrado na Figura 9.3. Os cabeçalhos das mensagens podem ser usados para estabelecer o contexto necessário para um serviço ou para manter um *log* ou uma auditoria das operações. Um intermediário pode interpretar e atuar sobre as informações presentes nos cabeçalhos das mensagens, por exemplo, adicionando, alterando ou removendo informações. O corpo da mensagem transporta um documento XML para um serviço Web em particular.

Os elementos XML *envelope*, *cabeçalho* e *corpo*, juntos a outros atributos e elementos de mensagens SOAP, são definidos como um esquema no espaço de nomes XML do protocolo SOAP. A definição desse esquema pode ser encontrada no *site* Web do W3C [www.w3.org IX]. Como utilizam uma codificação textual, os esquemas XML podem ser vistos com a opção "exibir código-fonte" de um navegador. Tanto o cabeçalho como o corpo contêm elementos internos.

A seção anterior explicou que a descrição de serviço contém informações que devem ser compartilhadas por clientes e servidores. Os remetentes de mensagens usam essas descrições para gerar o corpo e para garantir que ele possua o conteúdo correto, e os destinatários das mensagens as utilizam para analisar e verificar a validade do conteúdo.

Uma mensagem SOAP pode ser usada para transmitir um documento ou para suportar comunicação cliente-servidor:

- Um documento a ser comunicado é colocado diretamente dentro do elemento *corpo*, junto a uma referência a um esquema XML contendo a descrição do serviço – a qual define os nomes e tipos usados no documento. Esse tipo de mensagem SOAP pode ser enviado de forma síncrona ou de forma assíncrona.
- Para comunicação cliente-servidor, o elemento *corpo* contém uma requisição (*Request*) ou uma resposta (*Reply*). Esses dois casos estão ilustrados nas Figuras 9.4 e 9.5.

A Figura 9.4 mostra um exemplo de mensagem de requisição simples, sem cabeçalho. O *corpo* engloba um elemento com o nome do procedimento a ser chamado e o URI do espaço de nomes (o arquivo que contém o esquema XML) para a descrição do serviço relevante, que é denotada por *m*. Os elementos internos de uma mensagem de requisição contêm os argumentos do procedimento. Essa mensagem de requisição fornece dois *strings* a serem retornados na ordem oposta pelo procedimento que está no servidor. O espaço de nomes XML, denotado por *env*, contém as definições SOAP de um

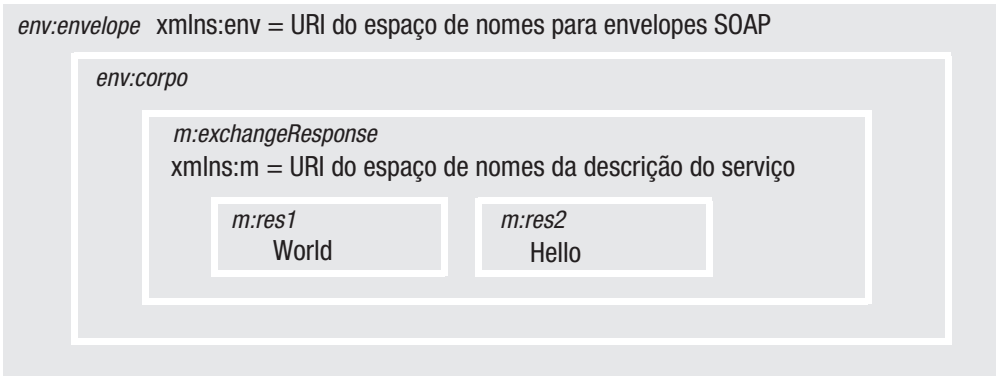


Figura 9.5 Exemplo de resposta correspondente à requisição da Figura 9.4.

envelope. A Figura 9.5 mostra a mensagem de resposta bem-sucedida correspondente, a qual contém os dois argumentos de saída. Note que o nome do procedimento tem *Response* anexado a ele. Se um procedimento tem um valor de retorno, então ele pode ser denotado como um elemento chamado *rpc: result*. Note que a mensagem de resposta usa os mesmos dois esquemas XML da mensagem de requisição, o primeiro definindo o envelope SOAP e o segundo definindo os nomes do procedimento e do argumento específicos da aplicação.

Erros SOAP: Se uma requisição falha de alguma maneira, as descrições do erro são transmitidas no corpo de uma mensagem de resposta em um elemento *falha*. Esse elemento contém informações sobre o erro, incluindo um código e um *string* associado, junto a detalhes específicos da aplicação.

Cabeçalhos SOAP • Os cabeçalhos das mensagens se destinam ao uso pelos intermediários para adicionar um serviço responsável por tratar a mensagem transportada no corpo. Entretanto, dois aspectos dessa utilização não ficam claros na especificação do SOAP:

1. A forma como os cabeçalhos serão usados por qualquer serviço de *middleware* de mais alto nível, em particular. Por exemplo, um cabeçalho poderia conter:
   - um identificador de transação para uso em um serviço de transação;
   - um identificador de mensagem para relacionar uma mensagem com outra, por exemplo, para implementar distribuição confiável;
   - um nome de usuário, uma assinatura digital ou uma chave pública.
2. A maneira como as mensagens serão direcionadas por meio de um conjunto de intermediários para o destinatário final. Por exemplo, uma mensagem transportada por HTTP poderia ser direcionada por meio de um encadeamento de servidores *proxies*, alguns dos quais poderiam assumir o papel do SOAP.

Entretanto, a especificação define as funções e tarefas dos intermediários. Um atributo chamado *papel* (*role*) pode especificar se todo intermediário, nenhum deles ou apenas o destinatário final deve processar o elemento (consulte [www.w3.org IX]). As ações em particular a serem executadas são definidas pelas aplicações; por exemplo, uma ação poderia ser o registro do conteúdo de um elemento.

Transporte de mensagens SOAP • Um protocolo de transporte é exigido para enviar uma mensagem SOAP para seu destino. As mensagens SOAP são independentes do tipo de

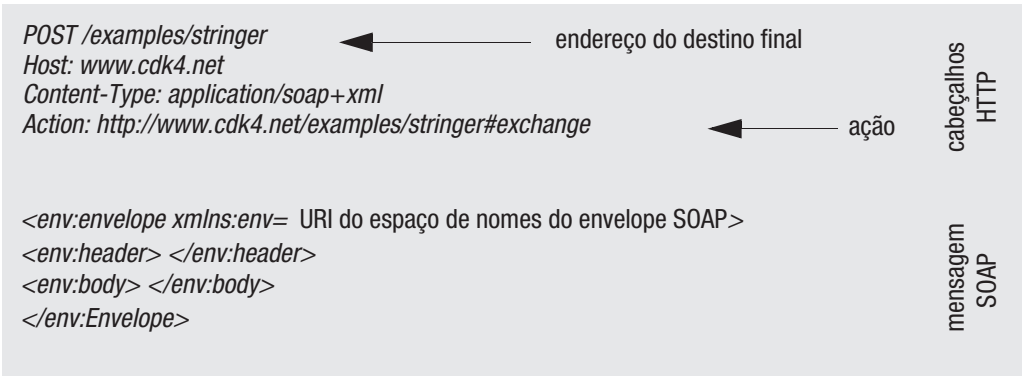


Figura 9.6 Uso de requisição HTTP *POST* na comunicação cliente-servidor do protocolo SOAP.

transporte usado – seus envelopes não contêm nenhuma referência ao endereço de destino. Fica para o protocolo HTTP (ou qualquer protocolo usado para transporte de uma mensagem SOAP) especificar o endereço do destino.

A Figura 9.6 ilustra como o método HTTP *POST* é usado para transmitir uma mensagem SOAP. Os cabeçalhos e o corpo HTTP são usados como segue:

- os cabeçalhos HTTP especificam o endereço do destino final (o URI do receptor final) e a ação a ser executada. O parâmetro *Action* se destina a otimizar o envio – revelando o nome da operação, sem a necessidade de analisar a mensagem SOAP presente no corpo da mensagem HTTP;
- o corpo HTTP transporta a mensagem SOAP.

Como o protocolo HTTP é síncrono, ele é usado para retornar uma resposta contendo a resposta SOAP; por exemplo, aquela mostrada na Figura 9.5. A Seção 5.2 pormenorizou os códigos de *status* e os motivos retornados pelo protocolo HTTP para requisições bem-sucedidas e malsucedidas.

Se uma requisição SOAP é apenas uma solicitação para um retorno de informações, não tendo argumentos e sem alterar dados no servidor, então o método HTTP *GET* pode ser usado para transportá-la.

A questão acima sobre o cabeçalho *Action* e sobre o envio se aplica a qualquer serviço que execute uma variedade de ações diferentes para os clientes, mesmo que ele não ofereça essas operações. Por exemplo, um serviço Web pode ser capaz de lidar com diferentes tipos de documentos, como requisições de compra e de consultas, os quais são tratados por diferentes módulos de *software*. O cabeçalho *Action* permite que o módulo correto seja escolhido sem inspecionar a mensagem SOAP. Esse cabeçalho pode ser usado, caso o tipo de conteúdo HTTP seja especificado como *application/soap+xml*.

A separação da definição do envelope SOAP das informações sobre como e para onde elas devem ser enviadas torna possível usar uma variedade de protocolos subjacentes diferentes. A especificação SOAP diz como o protocolo SMTP pode ser usado como uma maneira alternativa de transmitir documentos codificados como mensagens SOAP.

No entanto, essa vantagem também é uma desvantagem. Ela implica que o desenvolvedor deve estar envolvido nos detalhes do protocolo de transporte específico escolhido. Além disso, torna difícil usar diferentes protocolos para diferentes partes da rota seguida por uma mensagem em particular.

WS-Addressing: avanços no endereçamento e no direcionamento do protocolo SOAP • Dois problemas foram mencionados anteriormente:

- como tornar o protocolo SOAP independente do transporte subjacente usado;
- e como especificar uma rota a ser seguida por uma mensagem SOAP por meio de um conjunto de intermediários.

Um trabalho anterior nessa área, realizado por Nielsen e Thatte [2001], sugere que o endereço do destino final e a informação de envio devem ser especificados nos cabeçalhos SOAP. Isso separa efetivamente o destino da mensagem do protocolo subjacente. Eles sugeriram especificar o caminho a ser seguido fornecendo o endereço do destino final e o próximo passo (ou etapa). Cada um dos intermediários atualizaria a informação do próximo passo.

O trabalho de Box e Curbera [2004] sugere que fazer os intermediários alterarem os cabeçalhos poderia levar a brechas de segurança. Eles propuseram o *WS-Addressing*, que permite ao cabeçalhos SOAP especificar dados de roteamento de mensagem, com uma infraestrutura SOAP subjacente fornecendo a informação do próximo passo.

As recomendações do W3C para o WS-Addressing estão definidas no endereço [www.w3.org XXIII]. Essa forma de endereçamento usa uma *referência de destino final* (*Endpoint Reference*) – uma estrutura XML contendo o endereço de destino, informações de roteamento e, possivelmente, outras informações sobre o serviço. Para suportar interações assíncronas de longa duração, os cabeçalhos SOAP podem fornecer um endereço de retorno e identificadores de mensagem próprios e de mensagens relacionadas.

WS-ReliableMessaging: comunicação confiável • O protocolo normal do SOAP, HTTP, é executado sobre TCP, cujo modelo de falha foi discutido na Seção 4.2.4. Em resumo: o protocolo TCP não garante o envio de mensagens em face de todas as dificuldades – e quando atinge um tempo limite na espera por confirmações, ele declara que a conexão está desfeita, no ponto em que os processos que estão se comunicando ficam sem saber se as mensagens que enviaram recentemente foram recebidas ou não.

O trabalho anterior visando ao fornecimento de comunicação confiável de mensagens SOAP com distribuição garantida, sem duplicação e com a ordem das mensagens assegurada, levou a duas especificações concorrentes de Ferris e Langworthy [2004] e Evans *et al*. [2003].

Mais recentemente, o Oasis (um consórcio global que trabalha no desenvolvimento, acordo e adoção de padrões para comércio eletrônico e serviços Web) elaborou uma recomendação chamada *WS-ReliableMessaging* [www.oasis.org]. Isso permite que uma mensagem SOAP seja enviada *pelo menos uma vez, no máximo uma vez* ou *exatamente uma vez*, com a seguinte semântica:

> *Pelo menos uma vez:* a mensagem é entregue pelo menos uma vez, mas se não puder ser entregue, um erro será relatado.
>
> *No máximo uma vez:* a mensagem é entregue no máximo uma vez, mas se não puder ser entregue, nenhum erro é relatado.
>
> *Exatamente uma vez:* a mensagem é entregue exatamente uma vez, mas se não puder ser entregue, um erro será relatado.

A ordem das mensagens também é fornecida, em combinação com qualquer um dos modo anteriores:

Em ordem: as mensagens serão entregues para o destino na ordem em que foram enviadas por um remetente em particular.

Note que a recomendação *WS-ReliableMessaging* se preocupa com o envio de mensagens únicas e não deve ser confundida com a semântica de chamada RPC, descrita na Seção 5.3.1, que se refere ao número de vezes que o servidor executa o procedimento remoto. No Exercício 9.16, o leitor verá uma consideração melhor sobre a comparação.

Passando por firewalls • Os serviços Web se destinam a serem usados por clientes de uma organização que acessam servidores de outra pela Internet. A maioria das organizações utiliza um *firewall* para proteger os recursos presentes em suas próprias redes, e os protocolos de transporte, como os usados pela RMI Java ou CORBA, normalmente não seriam capazes de atravessar um *firewall*. Entretanto, os *firewalls* normalmente permitem que mensagens HTTP e SMTP passem por eles. Portanto, é conveniente usar um desses protocolos para transportar mensagens SOAP.

9.2.2 Uma comparação de serviços Web com o modelo de objeto distribuído

Um serviço Web tem uma interface que pode fornecer operações para acessar e atualizar os recursos de dados que gerencia. De forma superficial, a interação entre cliente e servidor é muito parecida com RMI, em que um cliente usa uma referência de objeto remoto para invocar uma operação em um objeto remoto. Para um serviço Web, o cliente usa um URI para invocar uma operação no recurso nomeado por esse URI. Para argumentos sobre as semelhanças e diferenças entre serviços Web e objetos distribuídos, consulte Birman [2004], Vinoski [2002] e Vogels [2003].

Tentaremos mostrar que existem limites para a analogia acima, fazendo uso do exemplo do quadro compartilhado utilizado na Seção 5.5 para Java RMI e na Seção 8.3 para CORBA.

Referências de objeto remoto versus URIs • O URI de um serviço Web pode ser comparado com a referência de objeto remoto de um único objeto. Entretanto, no modelo de objeto distribuído, os objetos podem criar objetos remotos dinamicamente e retornar referências remotas para eles. O destinatário dessas referências remotas pode usá-las para invocar operações nos objetos a que se referem. No exemplo do quadro compartilhado, uma invocação do método de fábrica *newShape* faz uma nova instância de *Shape* ser criada e uma referência remota ser retornada para ela. Nada parecido com isso pode ser feito com serviços Web, que não podem criar instâncias de objetos remotos; com efeito, um serviço Web consiste em um único objeto remoto e, assim, tanto a coleta de lixo como a referência a objeto remoto são irrelevantes.

Modelo de serviços Web • Os usuários dos *toolkits* de serviços Web Java (JAX-RPC) [java.sun.com VII] devem modelar seus programas que usam serviços Web considerando o fato de que não estão usando invocação remota transparente de Java para Java, mas sim usando o modelo de serviços Web, no qual objetos remotos não podem ser instanciados. Isso é levado em conta pelo JAX-RPC, que não permite que referências de objeto remoto sejam passadas como argumentos, nem retornadas como resultados.

A Figura 9.7 mostra uma versão da interface apresentada na Figura 5.16, que foi modificada, como segue, para se tornar uma interface de serviços Web:

- Na versão original (objeto distribuído) do programa, instâncias de *Shape* são criadas no servidor e referências remotas são retornadas para eles por *newShape*, cuja versão (serviço Web) modificada é mostrada na linha 1. Para evitar a instanciação

```
import java.rmi.*;
public interface ShapeList extends Remote {
      int newShape(GraphicalObject g) throws RemoteException;                1
      int numberOfShapes( )throws RemoteException;
      int getVersion( ) throws RemoteException;
      int getGOVersion(int i) throws RemoteException;
      GraphicalObject getAllState(int i) throws RemoteException;
}
```

Figura 9.7 Interface de serviço Web Java *ShapeList.*

de objetos remotos e o consequente uso de referências de objeto remoto, a interface *Shape* foi removida e suas operações (*getAllState* e *getGOVersion* – originalmente *getVersion*) foram adicionadas na interface *ShapeList*.

- Na versão original (objeto distribuído) do programa, o servidor armazenava um vetor de *Shape*. Isso foi mudado para um vetor de *GraphicalObject*. A nova versão (serviço Web) do método *newShape* retorna um valor inteiro que fornece o deslocamento do objeto *GraphicalObject* nesse vetor.

Essa alteração no método *newShape* significa que ele não é mais um método de fábrica – isto é, ele não cria instâncias de objetos remotos.

Serventes • No modelo de objeto distribuído, o programa servidor geralmente é modelado como um conjunto de serventes (potencialmente, objetos remotos). Por exemplo, a aplicação de quadro compartilhado usava um servente para a lista de figuras e um servente para cada objeto gráfico criado. Esses serventes eram criados como instâncias das classes serventes *ShapeList* e *Shape*, respectivamente. Quando o servidor iniciava, sua função *main* criava a instância de *ShapeList,* e, sempre que o cliente chamava o método *newShape*, o servidor criava uma instância de *Shape*.

Em contraste, os serviços Web não suportam serventes. Portanto, as aplicações de serviços Web não podem criar serventes como e quando eles são necessários para manipular diferentes recursos do servidor. Para impor essa situação, as implementações de interfaces de serviço Web não devem ter construtores nem métodos principais.

9.2.3 O uso de SOAP com Java

A API Java para desenvolvimento de serviços Web e clientes em SOAP é chamada de JAX-RPC. Ela está descrita no exercício dirigido sobre serviços Web Java [java.sun.com VII]. Essa API oculta todos os detalhes do protocolo SOAP dos programadores tanto de clientes como de serviços.

A JAX-RPC faz o mapeamento de alguns dos tipos da linguagem Java para definições em XML usadas tanto em mensagens SOAP como em descrições de serviço. Os tipos permitidos incluem *Integer*, *String*, *Date* e *Calendar*, assim como *java.net.uri*, que permite que URIs sejam passados como argumentos ou retornados como resultados. Ela suporta alguns dos tipos de coleção (incluindo *Vector*), assim como os tipos primitivos da linguagem e *vetores*.

Além disso, instâncias de algumas classes podem ser passadas como argumentos e resultados de chamadas remotas, desde que:

- Cada uma de suas variáveis de instância seja de um dos tipos permitidos.
- Elas tenham um construtor público padrão.

```
import java.util.Vector;
public class ShapeListImpl implements ShapeList{
      private Vector theList = new Vector( );
      private int version = 0;
      private Vector theVersions = new Vector( );

      public int newShape(GraphicalObject g) throws RemoteException{
            version++;
            theList.addElement(g);
            theVersions.addElement(new Integer(version));
            return theList.size( );
      }
      public int numberOfShapes( ){ }
      public int getVersion( ) { }
      public int getGOVersion(int i){ }
      public GraphicalObject getAllState(int i) { }
}
```

Figura 9.8 Implementação em Java do servidor *ShapeList*.

- Elas não implementem a interface *Remote*.

Em geral, conforme mencionado na seção anterior, os valores de tipos que são referências remotas (isto é, que implementam a interface *Remote*) não podem ser passados como argumentos, nem retornados como resultados de chamadas remotas.

A interface de serviço • A interface Java de um serviço Web deve obedecer às regras a seguir, algumas das quais estão ilustradas na Figura 9.7:

- Ela deve estender a interface *Remote*.
- Ela não deve ter declarações constantes, como *public final static*.
- Os métodos devem disparar a exceção *java.rmi.RemoteException* ou uma de suas subclasses.
- Os parâmetros e tipos de retorno do método devem ser tipos permitidos na JAX-RPC.

O programa servidor • A classe que implementa a interface *ShapeList* aparece na Figura 9.8. Conforme explicado anteriormente, não há nenhum método *main*, e a implementação da interface *ShapeList* não tem construtor. Na verdade, um serviço Web é um objeto único que oferece um conjunto de procedimentos. O código-fonte dos programas mostrados nas Figuras 9.7, 9.8 e 9.9 está disponível na página deste livro, em www.grupoa.com.br ou em www.cdk5.net/Web.

A interface de serviço e sua implementação são compiladas normalmente. Duas ferramentas, chamadas *wscompile* e *wsdeploy*, podem ser usadas para gerar a classe esqueleto e a descrição do serviço (em WSDL, conforme descrito na Seção 9.3), usando informações relativas ao URL do serviço, seu nome e a descrição de um arquivo de configuração escrito em XML. O nome do serviço (neste caso, *MyShapeListService*) é usado para gerar o nome da classe utilizada no programa cliente para acessá-lo. Isto é, *MyShapeListService_Impl*.

Contêiner de servlet • A implementação do serviço é executada como um *servlet* dentro de um *contêiner de servlet*, cuja função é carregar, inicializar e executar *servlets*. O contêiner de *servlet* inclui um despachante (*dispatcher*) e esqueletos (veja a Seção 5.4.2). Quando chega uma requisição, o despachante faz seu mapeamento para um esqueleto

em particular, o qual a transforma em código Java e passa para o método apropriado no *servlet*, o qual executa a requisição e produz uma resposta, que o esqueleto transforma de volta em uma resposta SOAP. O URL de um serviço consiste em uma concatenação do URL do contêiner de *servlet* e da categoria e do nome do serviço, por exemplo, *http:// localhost:8080/ShapeList-jaxrpc/ShapeList*.

O Tomcat [jakarta.apache.org] é um contêiner de *servlet* comumente usado. Quando o Tomcat está em execução, sua interface de gerenciamento está disponível em um URL para visualização com um navegador. Essa interface mostra os nomes dos *servlets* que estão correntemente distribuídos e fornece operações para gerenciá-los e para acessar informações sobre cada um, incluindo sua descrição de serviço. Uma vez que um *servlet* seja distribuído no Tomcat, os clientes podem acessá-lo, e os efeitos combinados de suas operações serão armazenados em suas variáveis de instância. Em nosso exemplo, uma lista de objetos *GraphicalObjects* será construída, à medida que cada um deles for adicionado, como resultado de uma requisição cliente para a operação *newShape*. Se um *servlet* for interrompido pela interface de gerenciamento do Tomcat, os valores das variáveis de instância serão reconfigurados quando ele for reiniciado.

O Tomcat também dá acesso a uma descrição de cada um dos serviços que contém, para permitir que os programadores projetem programas clientes e para facilitar a compilação automática dos *proxies* exigidos pelo código do cliente. A descrição do serviço é legível para seres humanos, pois é expressa em notação XML (mais especificamente, em WSDL, conforme apresentado na Seção 9.3)

Note que é possível desenvolver serviços Web sem usar contêineres de *servlet*; por exemplo, o Apache Axis oculta do programador esse nível de detalhe.

O programa cliente • O programa cliente pode usar *proxies* estáticos, *proxies* dinâmicos ou uma interface de invocação dinâmica. Em todos os casos, a descrição do serviço relevante pode ser usada para obter as informações exigidas pelo código do cliente. Em nosso exemplo, a descrição do serviço pode ser obtida do Tomcat.

Proxies estáticos: a Figura 9.9 mostra o cliente *ShapeList* fazendo uma chamada por meio de um *proxy* – um objeto local que passa mensagens para o serviço remoto. O código do *proxy* é gerado por *wscompile* a partir da descrição do serviço. O nome da classe do *proxy* é formado pela adição de *_Impl* ao nome do serviço – neste caso, a classe do *proxy* é chamada de *MyShapeListService_Impl*. Esse nome é específico da implementação, pois a especificação do SOAP não fornece uma regra para atribuição de nomes de classes de *proxy*.

Na linha 1, o método *createProxy* é chamado. Esse método aparece na linha 5, onde vai para a linha 6 para criar um *proxy*, usando a classe *MyShapeListService_Impl*; em seguida, ele retorna o *proxy* (note que às vezes os *proxies* são chamados de *stubs*, daí o nome da classe *Stub*). Na linha 2, o URL do serviço é fornecido para o *proxy* por intermédio do argumento dado na linha de comando. Na linha 3, o tipo do *proxy* é ajustado para estar de acordo com o tipo da interface – *ShapeList*. A linha 4 faz uma chamada para o procedimento remoto *getAllState*, solicitando ao serviço para que retorne o objeto que está no elemento zero no vetor de *GraphicalObjects*.

Como o *proxy* é criado no momento da compilação, ele é chamado de *proxy* estático. A descrição do serviço a partir da qual ele foi gerado não terá sido necessariamente gerada a partir de uma interface Java, mas pode ter sido feita por qualquer uma das várias

```
package staticstub;
import javax.xml.rpc.Stub;
public class ShapeListClient {
    public static void main(String[ ] args) { /* passa URL de serviço */
        try {
            Stub proxy = createProxy( );                                    1
            proxy._setProperty                                              2
                (javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY, args[0]);
            ShapeList aShapeList = (ShapeList)proxy;                        3
            GraphicalObject g = aShapeList.getAllState(0);                  4
        } catch (Exception ex)    { ex.printStackTrace( ); }
    }

    private static Stub createProxy( ) {                                    5
        return
            (Stub) (new MyShapeListService_Impl( ).getShapeListPort( ))     6
  }
}
```

Figura 9.9 Implementação em Java do cliente *ShapeList*.

ferramentas associadas a uma variedade de linguagem diferentes. Ela pode até ter sido escrita diretamente em XML.

Proxies dinâmicos: em vez de usar um *proxy* estático previamente compilado, o cliente usa um *proxy* dinâmico cuja classe é criada em tempo de execução a partir das informações presentes na descrição do serviço e na interface do serviço. Esse método evita a necessidade de usar, na implementação, um nome específico para a classe *proxy*.

Interface de invocação dinâmica: isso permite que um cliente chame um procedimento remoto mesmo que sua assinatura ou o nome do serviço seja desconhecido até o momento da execução. Em contraste com as alternativas anteriores, o cliente não exige um *proxy*. Em vez disso, ele precisa usar uma série de operações para configurar o nome da operação do servidor, o valor de retorno e cada um dos parâmetros antes de fazer a chamada de procedimento.

Implementação de SOAP Java • A maneira como a API Java é implementada pode ser explicada com referência à Figura 5.15. Os parágrafos a seguir explicam as funções dos vários componentes presentes em um ambiente Java/SOAP – as interações entre os componentes são iguais às de antes. Não existe nenhum módulo de referência remota.

Módulos de comunicação: as tarefas desses módulos são executadas por dois módulos HTTP. O módulo HTTP no servidor seleciona o despachante de acordo com o URL dado no cabeçalho *Action* da requisição POST.

Proxy cliente: um método de *proxy* (ou *stub*) conhece o URL do serviço e empacota seu próprio nome de método e seus argumentos, junto a uma referência para o esquema XML do serviço, em um envelope de requisição SOAP. Desempacotar a resposta consiste em analisar um envelope SOAP para extrair os resultados, o valor de retorno ou o relatório de falha. A chamada de método de requisição do cliente é enviada para o serviço como uma requisição HTTP.

Despachante e esqueleto: conforme mencionado anteriormente, o despachante e os esqueletos ficam no contêiner de *servlet*. O despachante extrai o nome da operação do cabeçalho *Action* na requisição HTTP e invoca o método correspondente no esqueleto apropriado, passando a ele o envelope SOAP. Um método de esqueleto executa as seguintes tarefas: ele analisa o envelope SOAP presente na mensagem de requisição e extrai seus argumentos, chama o método correspondente e monta um envelope de resposta SOAP contendo os resultados.

Erros, falhas e correção no SOAP/XML: falhas podem ser relatadas pelo módulo HTTP, pelo despachante, pelo esqueleto ou pelo próprio serviço. O serviço pode relatar seus erros por meio de um valor de retorno ou de parâmetros de falha especificados na descrição do serviço. O esqueleto é responsável por verificar se o envelope SOAP contém uma requisição e se o código XML no qual ele está escrito é bem formado. Tendo estabelecido que o código XML é bem formado, o esqueleto usará o espaço de nomes XML presente no envelope para verificar se a requisição corresponde ao serviço oferecido e se a operação e seus argumentos são apropriados. Se a validação da requisição falhar em um desses níveis, um erro será retornado para o cliente. Verificações semelhantes são feitas pelo *proxy*, quando ele recebe o envelope SOAP contendo o resultado.

9.2.4 Comparação entre serviços Web e CORBA

A principal diferença entre serviços Web e o CORBA, ou outro *middleware* semelhante, é o contexto no qual eles se destinam a serem usados. O CORBA foi projetado para uso dentro de uma única organização, ou entre um pequeno número de organizações colaboradoras. Isso resultou em certos aspectos do projeto sendo centralizados demais para uso cooperativo por parte de organizações independentes ou para uso *ad hoc* sem arranjos anteriores, conforme será explicado a partir de agora.

Problemas de atribuição de nomes • No CORBA, cada objeto remoto é referenciado por meio de um nome, que é gerenciado por uma instância do serviço de atribuição de nomes CORBA (Seção 8.3.5). Esse serviço, assim como o DNS, fornece um mapeamento de um nome para um valor, a ser usado como endereço (um IOR, no CORBA). No entanto, ao contrário do DNS, o serviço de atribuição de nomes CORBA é projetado para uso dentro de uma organização, em vez de ser usado por toda a Internet.

No serviço de atribuição de nomes CORBA, cada servidor gerencia um grafo de nomes com um contexto de atribuição de nomes inicial e é inicialmente independente de quaisquer outros servidores. Embora organizações separadas possam confederar (unir) seus serviços de atribuição de nomes, isso não é automático. Antes que um servidor possa se unir com outro, ele precisa conhecer o contexto de atribuição de nomes inicial deste. Assim, o projeto do serviço de atribuição de nomes CORBA restringe efetivamente o compartilhamento de objetos CORBA dentro de um pequeno conjunto de organizações que tenham confederado seus serviços de atribuição de nomes.

Problemas de referência • Agora, consideraremos se uma referência de objeto remoto CORBA, que é chamada de IOR (Seção 8.3.3), poderia ser usada como uma referência de objeto em nível de Internet, da mesma maneira que um URL. Cada IOR contém um campo que especifica o identificador de tipo da interface do objeto que referencia. Entretanto, esse identificador de tipo é entendido apenas pelo repositório de interfaces que armazena a definição do tipo correspondente. Isso tem a implicação de que o cliente e o servidor precisam usar o mesmo repositório de interfaces, o que na realidade não é prático em escala global.

Em contraste, no modelo de serviços Web, um serviço é identificado por meio de um URL, permitindo que um cliente em qualquer parte da Internet faça uma requisição para um serviço que pode pertencer a qualquer organização de qualquer lugar. Isto é, um serviço Web pode ser compartilhado por clientes de toda a Internet. O DNS é o único serviço exigido para acesso de URL – e é projetado para funcionar eficientemente no âmbito da Internet.

Separação de ativação e localização • As tarefas de localizar e ativar serviços Web são bem separadas. Em contraste, uma referência persistente do CORBA se refere a um componente da plataforma (o repositório de implementação) que ativa o objeto correspondente sob demanda em um computador determinado e também é responsável por localizar o objeto quando ele tiver sido ativado.

Facilidade de uso • A infraestrutura HTTP e XML de serviços Web é bem entendida e conveniente para uso e já está instalada em todos os sistemas operacionais comumente usados, embora o usuário exija uma API de linguagem de programação com suporte para SOAP. Em contraste, a plataforma CORBA é um *software* grande e complexo, exigindo instalação e suporte.

Eficiência • O CORBA foi projetado para ser eficiente: o CDR CORBA (Seção 4.3.1) é binário, enquanto a XML é textual. Um estudo realizado por Olson e Ogbuji [2002] compara o desempenho do CORBA com o do SOAP e da XML-RPC. Eles descobriram que as mensagens de requisição SOAP são 14 vezes maiores do que as equivalentes em CORBA, e que uma requisição SOAP demora 882 vezes mais que uma invocação CORBA equivalente. Embora o desempenho relativo dependa da linguagem utilizada e da implementação do *middleware* específica empregada, esse exemplo fornece uma indicação da sobrecarga em potencial das estratégias baseadas em XML. Para muitas aplicações, contudo, a sobrecarga de mensagem e o pior desempenho do SOAP não são notados e seus efeitos se tornam menos óbvios pela disponibilidade de largura de banda, processadores, memória e espaço em disco mais baratos.

O W3C e outros investigaram a possibilidade de permitir a inclusão de dados binários em elementos da XML para aumentar a eficiência. Discussões sobre esse assunto podem ser encontradas no endereço [www.w3.org XXI] e [www.w3.org XXII]. Note que a XML já fornece representações em hexadecimal e base64 de dados binários. A representação em base64 é usada em conjunto com a criptografia na XML (veja a Seção 9.5). Existe uma sobrecarga de tempo e espaço considerável quando dados binários são convertidos para base64 ou hexadecimal. Portanto, o que é realmente necessário é poder incluir uma representação em binário de uma sequência de itens de dados previamente analisada, como, por exemplo, a produzida pelo CDR CORBA ou por *gzip*. Outra estratégia, que também está sendo investigada, é pegar uma mensagem SOAP, junto a anexos, alguns dos quais podem ser binários, e usar um documento MIME de várias partes para transportá-la.

As vantagens do CORBA • A disponibilidade de serviços CORBA para transações, controle de concorrência, segurança e controle de acesso, eventos e objetos persistentes o torna uma escolha desejável em muitas aplicações destinadas para uso interno de uma organização ou em um grupo de organizações relacionadas. Em geral, essa é uma boa escolha para as aplicações que exigem interações muito complexas. Além disso, o modelo de objeto distribuído é atraente para o projeto de aplicações complexas e vale o esforço de aprendizado extra, necessário para entender os detalhes do relacionamento entre o objeto modelo CORBA (Seção 8.3) e a linguagem de programação em particular que esteja em uso.

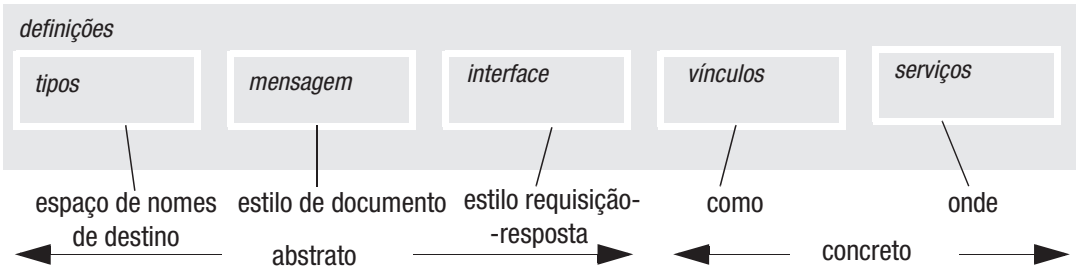


Figura 9.10 Os principais elementos em uma descrição WSDL.

9.3 Descrições de serviço e IDL para serviços Web

As definições de interface são necessárias para permitir que os clientes se comuniquem com os serviços. Para serviços Web, as definições de interface são fornecidas como parte de uma descrição de serviço mais geral, que especifica duas outras características adicionais – como as mensagens devem ser comunicadas (por exemplo, por SOAP com HTTP) e o URI do serviço. Para fornecer serviço em um ambiente com múltiplas linguagens, as descrições de serviço são escritas em XML.

Uma descrição de serviço forma a base de um acordo entre um cliente e um servidor quanto ao serviço oferecido. Ela reúne todos os pontos pertinentes ao serviço que são relevantes para seus clientes. As descrições de serviço geralmente são usadas para gerar *stubs* de cliente que implementam automaticamente o comportamento correto para o cliente.

O componente do tipo IDL de uma descrição de serviço é mais flexível do que as outras IDLs, pois um serviço pode ser especificado em termos dos tipos de mensagens que enviará e receberá, ou em termos das operações que suporta, para permitir a troca de documentos e interações estilo requisição-resposta.

Uma variedade de métodos de comunicação diferentes pode ser usada pelos serviços Web e seus clientes. Portanto, o método de comunicação fica para ser decidido pelo provedor do serviço e é especificado na descrição do serviço, em vez de ser incorporado ao sistema, como no CORBA, por exemplo.

A capacidade de especificar o URI de um serviço como parte de sua descrição evita a necessidade do serviço vinculador ou de separado atribuição de nomes, usado pela maioria dos outros *middlewares*. Isso implica que o URI não pode ser alterado, uma vez que a descrição do serviço tenha se tornado disponível para clientes em potencial. No entanto, o esquema URN aceita a troca de local, permitindo um procedimento indireto no nível da referência.

Em contraste, na estratégia do vinculador, o cliente usa um nome para pesquisar a referência do serviço em tempo de execução, possibilitando que as referências de serviço mudem com o passar do tempo. Porém, essa estratégia exige um procedimento indireto de um nome para uma referência de serviço para todos os serviços, mesmo que muitos deles possam sempre permanecer no mesmo local.

No contexto dos serviços Web, a WSDL (ou Web Services Description Language) é comumente usada para descrições de serviço. A versão atual, a WSDL 2.0 [www.w3.org XI], tornou-se a recomendação do W3C em 2007. Ela define um esquema XML para representar os componentes de uma descrição de serviço, os quais incluem, por exemplo, os nomes de elemento *definições*, *tipos*, *mensagens*, *interface*, *vínculos* e *serviços*.

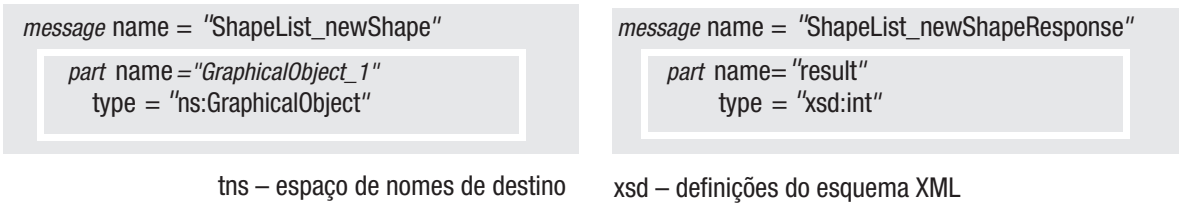


Figura 9.11 Mensagens de requisição-resposta WSDL para a operação *newShape*.

A WSDL separa a parte abstrata de uma descrição de serviço da parte concreta, como se vê na Figura 9.10.

A parte abstrata da descrição inclui um conjunto de definições (*definitions*) dos tipos usados pelo serviço, em particular os tipos dos valores trocados nas mensagens. O exemplo em Java da Seção 9.2.3, cuja interface Java aparece na Figura 9.7, usa os tipos Java *int* e *GraphicalObject*. O primeiro (assim como qualquer tipo básico) pode ser transformado diretamente no equivalente da XML. No entanto, *GraphicalObject* é definido na linguagem Java em termos dos tipos *int*, *String* e *boolean*. *GraphicalObject* é representado na XML, para uso comum por clientes heterogêneos, como *complexType*, consistindo em uma sequência de tipos XML nomeados, incluindo, por exemplo:

```
<element name="isFilled" type="boolean"/>
<element name="originx" type="int"/>
```

O conjunto de nomes definido dentro da seção tipos (*types*) de uma definição WSDL é chamado de *espaço de nomes de destino*. A seção mensagem (*message*) da parte abstrata contém uma descrição do conjunto de mensagens trocadas. Para o estilo documento de interação, essas mensagens serão usadas diretamente. Para o estilo de interação requisição-resposta, existem duas mensagens para cada operação, as quais são usadas para descrever as operações na seção *interface*. A parte concreta especifica como e onde o serviço pode ser contatado.

A modularidade inerente de uma definição WSDL permite que seus componentes sejam combinados de diferentes maneiras; por exemplo, a mesma interface pode ser usada com diferentes vínculos ou localizações. Os tipos podem ser definidos dentro do elemento *types,* ou em um documento separado referenciado por um URI do elemento *types*. Neste último caso, as definições de tipo podem ser referenciadas a partir de vários documentos WSDL diferentes.

Mensagens ou operações • Nos serviços Web, tudo que o cliente e o servidor precisam é ter uma ideia comum sobre a mensagem a ser trocada. Para um serviço baseado na troca de um pequeno número de tipos diferentes de documento, a WSDL descreve apenas os tipos das diferentes mensagens a serem trocadas. Quando um cliente envia uma dessas mensagens para um serviço Web, este decide qual operação vai efetuar e qual tipo de mensagem vai enviar de volta para o cliente, de acordo com o tipo da mensagem que recebeu. Em nosso exemplo em Java, duas mensagens serão definidas para cada uma das operações na interface – uma para a requisição e uma para a resposta. Por exemplo, a Figura 9.11 mostra as mensagens de requisição-resposta da operação de *newShape*, que tem um único argumento de entrada de tipo *GraphicalObject* e um único argumento de saída de tipo *int*.

Contudo, para serviços que suportam várias operações diferentes, é mais eficiente especificar as mensagens trocadas como requisições de operações com argumentos e suas respostas correspondentes, permitindo que o serviço envie cada requisição para a operação apropriada. Entretanto, na WSDL uma operação é uma construção de mensa-

Nome	*Mensagens enviadas por*			
	Cliente	*Servidor*	*Distribuição*	*Mensagem de falha*
In-Out	*Requisição*	*Resposta*		Pode substituir *resposta*
In-Only	*Requisição*			Nenhuma
Robust In-Only	*Requisição*		Garantida	Pode ser enviada
Out-In	*Resposta*	*Requisição*		Pode substituir *resposta*
Out-Only		*Requisição*		Nenhuma
Robust Out-Only		*Requisição*	Garantida	Pode ser enviada

Figura 9.12 Padrões de troca de mensagem para operações WSDL.

gens de requisição-resposta relacionadas, em vez da definição de uma operação em uma interface de serviço.

Interface • O conjunto de operações pertencentes a um serviço Web é agrupado em um elemento da XML chamado *interface* (também chamado de *portType*). Cada operação deve especificar o padrão de troca de mensagens entre cliente e servidor. As opções disponíveis incluem as que aparecem na Figura 9.12. A primeira delas, *In-Out*, é a forma requisição-resposta de comunicação cliente-servidor normalmente usada. Nesse padrão, a mensagem de resposta pode ser substituída por uma mensagem de falha. *In-Only* serve para mensagens unilaterais com semântica *maybe* e *Out-Only* serve para mensagens unilaterais do servidor para o cliente; mensagens de falha não podem ser enviadas com nenhuma das duas. *Robust In-Only* e *Robust Out-Only* são as mensagens correspondentes com distribuição garantida; e mensagens de falha podem ser trocadas. *Out-In* é uma interação requisição-resposta iniciada pelo servidor. A WSDL 2.0 também é extensível, pois as organizações podem introduzir seus próprios padrões de troca de mensagem, caso os predefinidos se mostrem inadequados.

Voltando ao nosso exemplo em Java, cada uma das operações é definida de modo a ter um padrão *In-Out*. A operação *newShape* aparece na Figura 9.13, usando as mensagens definidas na Figura 9.11. Essa definição, junto às definições das quatro outras operações, é incluída em um elemento *interface* da XML. Uma operação também pode especificar as mensagens de falha que podem ser enviadas.

Contudo, se, por exemplo, uma operação tem dois argumentos, digamos, um inteiro e um *string*, então não há necessidade de definir um novo tipo de dados, pois esses tipos são definidos para esquemas XML. Entretanto, será necessário definir uma mensagem

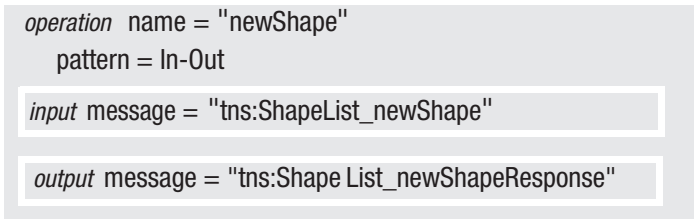


Figura 9.13 Operação *newShape* da WSDL.

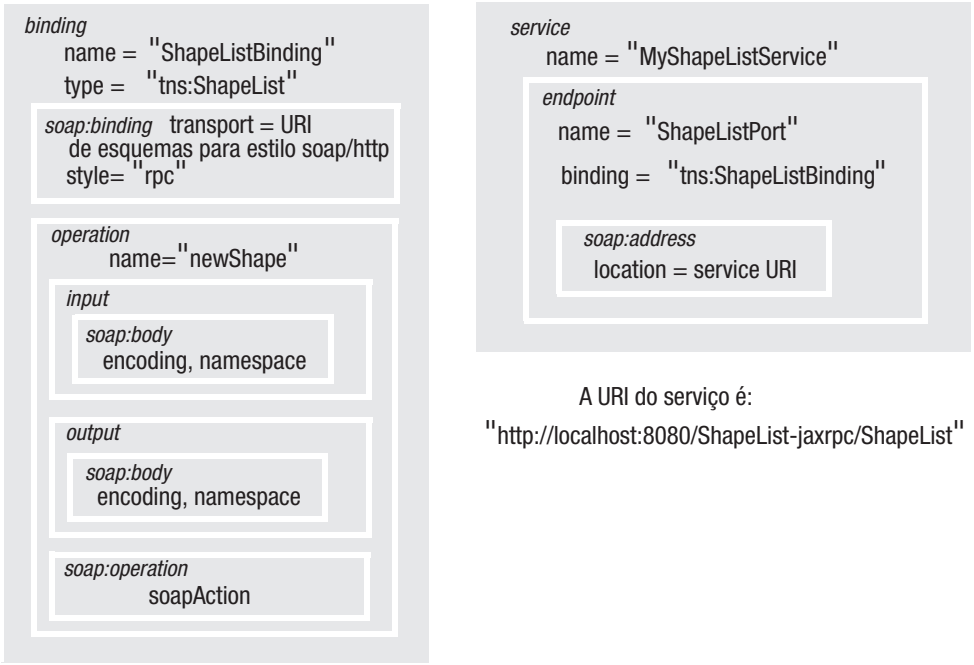


Figura 9.14 Definições de vínculo e serviço SOAP.

que tenha essas duas partes. Essa mensagem pode, então, ser usada como entrada ou saída na definição da operação.

Herança: toda interface WSDL pode estender uma ou mais outras interfaces WSDL. Essa é uma forma simples de herança, na qual uma interface suporta as operações de todas as interfaces que ela estende, além daquelas que ela mesma define. A definição recursiva de interfaces não é permitida; isto é, se a interface B estende a interface A, então a interface A não pode estender a interface B.

Parte concreta • A parte restante (concreta) de um documento WSDL consiste na escolha de protocolos, definido pelo elemento vínculo (*binding*) e na escolha do ponto final ou do servidor por meio do elemento serviço (*service*). As duas estão relacionadas, pois a forma de endereço depende do tipo de protocolo que está sendo usado. Por exemplo, um ponto final SOAP usará um URI, enquanto um ponto final CORBA usaria um identificador de objeto específico do CORBA.

Vínculo: a seção vínculo (*binding)* em um documento WSDL informa quais formatos de mensagem e forma de representação de dados externa devem ser usados. Por exemplo, os serviços Web frequentemente usam SOAP, HTTP e MIME. Os vínculos podem ser associados a operações, ou interfaces, em particular, ou podem ficar livres para uso por uma variedade de diferentes serviços Web.

A Figura 9.14 mostra um exemplo de *vínculo*, incluindo um *soap:binding*, que especifica o URL de um protocolo para transmitir envelopes SOAP: o vínculo HTTP para SOAP. Os atributos opcionais desse elemento também podem especificar o seguinte:

- o padrão de troca de mensagem, que pode ser *rpc* (requisição-resposta) ou troca de *document* – o valor padrão é *document*;

- o esquema XML dos formatos de mensagem – o padrão é o *envelope* SOAP;
- o esquema XML da representação de dados externa – o padrão é a codificação SOAP da XML.

A Figura 9.14 também mostra os detalhes dos vínculos de uma das operações (*newShape*), especificando que a mensagem de *entrada* e a de *saída* devem ser passadas na seção *soap:body*, usando um estilo de codificação em particular e, além disso, que a operação deve ser transmitida como uma *Action* SOAP.

Serviço: cada elemento serviço (*service)* em um documento WSDL especifica o nome do serviço e um ou mais *pontos finais* (ou portas) onde uma instância do serviço pode ser contatada. Cada um dos elementos ponto-finais (*endpoint*) se refere ao nome do vínculo em uso e, no caso de um vínculo SOAP, utiliza um elemento *soap:address* para especificar o URI da localização do serviço.

Documentação • Informações legíveis para seres humanos e pela máquina podem ser inseridas em um elemento documento (*documentation*) na maioria dos pontos dentro da WSDL. Essas informações podem ser removidas antes que a WSDL seja usada para processamento automático, por exemplo, por compiladores de *stub*.

Uso de WSDL • Documentos WSDL completos podem ser acessados por meio de seus URIs por clientes e servidores, direta ou indiretamente, por intermédio de um serviço de diretório como o UDDI. Estão disponíveis ferramentas para gerar definições WSDL a partir das informações fornecidas por meio de uma interface gráfica com o usuário, eliminando a necessidade de envolvimento dos usuários com os detalhes complexos e com a estrutura da WSDL. Por exemplo, a *Web Services Description Language* do *toolkit* Java permite a criação, representação e manipulação de documentos WSDL descrevendo serviços [wsdl4j.sourceforge.org]. As definições WSDL também podem ser geradas a partir de definições de interface escritas em outras linguagens, como JAX-RPC Java, discutida na Seção 9.2.2.

9.4 Um serviço de diretório para uso com serviços Web

Existem muitas maneiras pelas quais os clientes podem obter descrições de serviço; por exemplo, qualquer um que forneça um serviço Web de mais alto nível, como o serviço de agente de viagens, discutido na Seção 9.1, quase certamente faria uma página Web anunciando que o serviço e os clientes em potencial se depariariam com a página ao procurar serviços desse tipo.

Entretanto, qualquer organização que pretenda basear suas aplicações em serviços Web achará mais conveniente usar um serviço de diretório para tornar esses serviços disponíveis para os clientes. Esse é o objetivo do serviço UDDI (Universal Description, Discovery and Integration) [Bellwood *et al*. 2003], que fornece um serviço de nome e um serviço de diretório (veja a Seção 13.3). Isto é, as descrições de serviço WSDL podem ser pesquisadas pelo nome (um serviço de catálogo telefônico) ou pelo atributo (um serviço de páginas amarelas). Elas também podem ser acessadas diretamente por meio de seus URLs, o que é conveniente para desenvolvedores que estejam projetando programas clientes que utilizam o serviço.

Os clientes podem usar a estratégia das páginas amarelas para pesquisar uma categoria de serviço em particular, como um agente de viagens ou uma livraria, ou podem

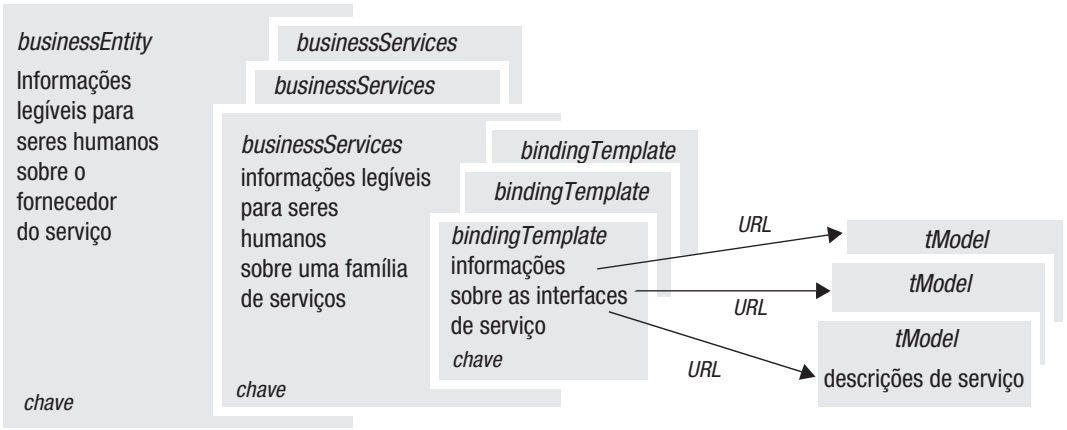


Figura 9.15 As principais estruturas de dados UDDI.

usar a estratégia do catálogo telefônico para pesquisar um serviço com referência à organização que o fornece.

Estruturas de dados • As estruturas de dados que suportam UDDI são projetadas de forma a permitir todos os estilos de acesso anteriores e podem incorporar qualquer volume de informações legíveis para seres humanos. Os dados são organizados em termos das quatro estruturas mostradas na Figura 9.15, cada uma das quais podendo ser acessada individualmente, por meio de um identificador chamado de *chave* (além de *tModel*, que pode ser acessado por um URL):

businessEntity: descreve a organização que fornece esses serviços Web, dando seu nome, endereço, atividades etc.;

businessServices: armazena informações sobre um conjunto de instâncias de um serviço Web, como seu nome e uma descrição de seu propósito; por exemplo, agente de viagens ou livraria;

bindingTemplate: contém o endereço de uma instância de serviço Web e referências para descrições do serviço;

tModel: contém descrições de serviço, normalmente documentos WSDL, armazenadas fora do banco de dados e acessadas por meio de URLs.

Pesquisa (lookup) • O UDDI fornece uma API para pesquisar (*lookup*) serviços com base em dois conjuntos de operações de consulta:

- O conjunto de operações *get_xxx* inclui *get_BusinessDetail*, *get_ServiceDetail*, *get_bindingDetail* e *get_tModelDetail*; elas recuperam uma entidade correspondente a uma dada chave;
- O conjunto de operações *find_xxx* inclui *find_business*, *find_service*, *find_binding* e *find_tModel*; elas recuperam o conjunto de entidades que correspondem a um conjunto de critérios de busca em particular, fornecendo um resumo dos nomes, descrições, chaves e URLs.

Assim, os clientes que estiverem de posse de uma chave em particular podem usar uma operação *get_xxx* para recuperar diretamente a entidade correspondente. Outros clientes podem usar navegação para ajudar nas buscas – começando com um conjunto de resulta-

dos abrangente e reduzindo-o gradualmente. Por exemplo, eles podem começar usando a operação *find_business* para obter uma lista contendo um resumo das informações sobre os provedores correspondentes. A partir desse resumo, o usuário pode usar a operação *find_service* para refinar a busca, fazendo-a corresponder ao tipo de serviço exigido. Nos dois casos, ele encontrará a chave de um *bindingTemplate* conveniente e, com isso, encontrará o URL para recuperar o documento WSDL de um serviço apropriado.

Além disso, o UDDI fornece uma interface de publicação/assinatura, por meio da qual os clientes registram o interesse em um conjunto de entidades em particular, em um registro UDDI, e obtêm notificações sobre alterações de forma síncrona ou assíncrona.

Publicação • O UDDI fornece uma interface para publicar (anunciar) e atualizar informações sobre serviços Web. Na primeira vez que uma estrutura de dados (veja a Figura 9.15) é publicada em um servidor UDDI, ele recebe uma chave na forma de um URI (por exemplo, *uddi:cdk5.net:213*) e esse servidor se torna seu proprietário.

Registros • O serviço UDDI é baseado em dados replicados armazenados em registros. Um registro UDDI consiste em um ou mais servidores UDDI, cada um dos quais tendo uma cópia do mesmo conjunto de dados. Os dados são replicados entre os membros de um registro. Cada um deles pode responder às consultas e publicar informações. As alterações em uma estrutura de dados devem ser enviadas ao seu proprietário – isto é, o servidor em que ela foi publicada pela primeira vez. É possível um proprietário transmitir sua posse para outro servidor UDDI no mesmo registro.

Esquema de replicação: os membros de um registro propagam cópias das estruturas de dados uns para os outros como segue: um servidor que fez alterações notifica os outros servidores no registro, os quais, então, solicitam as alterações. Uma forma de carimbo de tempo vetorial é usada para determinar quais das alterações devem ser propagadas e aplicadas. O esquema é simples, em comparação com outros esquemas de replicação que usam carimbo de tempo vetoriais, como Gossip (Seção 18.4.1) ou Coda (Seção 18.4.3), pois:

1. Todas as alterações em uma estrutura de dados em particular são feitas no mesmo servidor.
2. As atualizações de um servidor em particular são recebidas em ordem sequencial pelos outros membros, mas nenhuma ordenação em particular é imposta entre as operações de atualizações executadas por diferentes servidores.

Interação entre servidores: conforme descrito anteriormente, os servidores interagem uns com os outros para transmitir o esquema de replicação. Eles também podem interagir para transferir a posse de estruturas de dados. Entretanto, a resposta a uma operação de pesquisa é dada por um único servidor, sem nenhuma interação com outros servidores no registro, ao contrário do serviço de diretório X.500 (Seção 13.5), no qual os dados são particionados entre os servidores, que cooperam uns com os outros na busca do servidor relevante para uma requisição em particular.

9.5 Aspectos de segurança em XML

A segurança em XML consiste em um conjunto de projetos relacionados do W3C para assinatura, gerenciamento de chaves e criptografia. Ela se destina a ser usada no trabalho cooperativo pela Internet, envolvendo documentos cujo conteúdo talvez precise ser autenticado ou cifrado. Normalmente, os documentos são criados, trocados, armazenados

e, depois, novamente trocados, possivelmente após serem modificados por uma série de usuários diferentes.

A WS-Security [Kaler 2002] é outra estratégia de segurança relativa à aplicação de integridade, confidencialidade e autenticação de mensagens no SOAP.

Como exemplo de um contexto no qual a segurança em XML seria útil, considere um documento contendo os registros médicos de um paciente. Diferentes partes desse documento são usadas no consultório médico local e em várias clínicas e hospitais especializados visitados pelo paciente. Ele será atualizado por médicos, especialistas e enfermeiras que estejam tomando notas sobre condições e tratamento, por administradores que estejam marcando compromissos e por farmacêuticos para fornecer remédios. Diferentes partes do documento serão visíveis pelas diferentes funções mencionadas anteriormente e, possivelmente, também pelo paciente. É fundamental que certas partes do documento, por exemplo, as recomendações quanto ao tratamento, possam ser atribuídas à pessoa que as fez e possa haver garantias de que não tenham sido alteradas.

Essas necessidades não podem ser atendidas pelo TLS (anteriormente conhecido como SSL, Seção 11.6.3), usado para criar um canal seguro para a comunicação de informações. Ele permite que os processos nos dois extremos do canal negociem a necessidade de autenticação, de criptografia e as chaves e os algoritmos a serem usados, tanto quando um canal for estabelecido, quanto durante seu tempo de vida. Por exemplo, os dados sobre uma transação financeira poderiam ser assinados e enviados à vontade, até que informações sigilosas, como detalhes do cartão de crédito, fossem fornecidas, no ponto em que a criptografia seria aplicada.

Para possibilitar o novo tipo de utilização mencionado anteriormente, a segurança deve ser especificada e aplicada no próprio documento, em vez de ser uma propriedade do canal que o transmitirá de um usuário para outro.

Isso é possível em XML, ou em outros formatos de documento estruturados, nos quais podem ser usados metadados. Marcas (*tags*) XML podem ser usadas para definir as propriedades dos dados no documento. Em particular, a segurança em XML depende de novas *tags* que possam ser usadas para indicar o início e o fim de seções de dados cifrados, ou assinados, e de assinaturas. Uma vez que a segurança necessária tenha sido aplicada dentro de um documento, ele pode ser enviado para uma variedade de usuários diferentes, até por meio de *multicast*.

Requisitos básicos • A segurança em XML deve fornecer pelo menos o mesmo nível de proteção do TLS. Isto é:

Ser capaz de cifrar um documento inteiro ou apenas algumas partes selecionadas dele: por exemplo, considere as informações sobre uma transação financeira, as quais incluem o nome de uma pessoa, o tipo de transação e os detalhes sobre o cartão de crédito ou débito que está sendo usado. Em um caso, apenas os detalhes do cartão poderiam ser ocultos, tornando possível identificar a transação antes de decifrar o registro. No outro caso, o tipo de transação também poderia ser oculto, para que intrusos não pudessem identificar, por exemplo, se é um pedido ou um pagamento.

Ser capaz de assinar um documento inteiro ou apenas algumas partes selecionadas dele: quando um documento se destina a ser usado para trabalho cooperativo por um grupo de pessoas, podem existir algumas partes críticas do documento que devem ser assinadas para garantir que foram feitas por uma pessoa em particular

ou que não foram alteradas. Contudo, também é útil poder ter outras partes que possam ser alteradas durante o uso do documento – essas não devem ser assinadas.

Requisitos básicos adicionais • Mais requisitos surgem da necessidade de armazenar documentos, possivelmente para modificá-los e enviá-los para uma variedade de destinatários diferentes:

Fazer adições a um documento assinado e assinar o resultado final: por exemplo, Alice pode assinar um documento e passá-lo para Bob, que "atesta a assinatura dela" adicionando uma observação nesse sentido e depois assina o documento inteiro. (A Seção 11.1 apresenta os nomes, incluindo Alice e Bob, usados como protagonistas nos protocolos de segurança.)

Autorizar vários usuários diferentes a ver partes distintas de um documento: no caso de um registro médico, um pesquisador pode ver uma seção em particular dos dados médicos, um administrador pode ver detalhes pessoais e um médico pode ver ambos.

Fazer adições em um documento que já contém seções cifradas e cifrar parte da nova versão, possivelmente incluindo algumas das seções já cifradas.

A flexibilidade e os recursos de estruturação da notação XML tornam possível fazer tudo o que foi descrito anteriormente, sem quaisquer adições no esquema derivado dos requisitos básicos.

Requisitos relativos aos algoritmos • Os documentos XML seguros são assinados e/ou cifrados bem antes de qualquer consideração com relação a quem os acessará. Se o criador não estiver mais envolvido, não será possível negociar os protocolos e o uso de autenticação ou criptografia. Portanto:

O padrão deve especificar um conjunto de algoritmos a serem fornecidos em qualquer implementação de segurança em XML: pelo menos um algoritmo de criptografia e um de assinatura devem ser obrigatórios para permitir a maior interoperabilidade possível. Outros algoritmos opcionais devem ser fornecidos para uso dentro de grupos menores.

Os algoritmos usados para criptografia e autenticação de um documento em particular devem ser selecionados a partir desse conjunto, e os nomes dos algoritmos em uso devem ser referenciados dentro do próprio documento XML: se os lugares onde o documento será usado não podem ser previstos, então um dos protocolos exigidos deve ser usado.

A segurança em XML define os nomes dos elementos que podem ser usados para especificar o URI do algoritmo em uso para assinatura ou criptografia. Portanto, para ser capaz de selecionar uma variedade de algoritmos dentro do mesmo documento XML, geralmente um elemento especificando um algoritmo é aninhado dentro de um elemento que contém informações assinadas ou cifradas.

Requisitos para localização de chaves • Quando um documento é criado, e sempre que ele for atualizado, as chaves apropriadas devem ser escolhidas, sem qualquer negociação com as partes que poderão acessar o documento no futuro. Isso leva aos seguintes requisitos:

Ajudar os usuários de documentos seguros na localização das chaves necessárias: por exemplo, um documento que inclui dados assinados deve conter informações quanto a chave pública a ser usada para validar a assinatura, como um nome

Tipo de algoritmo	*Nome do algoritmo*	*Exigido*	*referência*
Resumo de mensagem	SHA-1	Exigido	Seção 11.4.3
Codificação	base64	Exigido	[Freed e Borenstein 1996]
Assinatura	DSA com SHA-1	Exigido	[NIST 1994]
(assimétrica)	RSA com SHA-1	Recomendado	Seção 11.3.2
Assinatura MAC	HMAC-SHA-1	Exigido	Seção 11.4.2 e
(simétrica)			Krawczyk *et al.* [1997]
Canonização	XML canônica	Exigido	Página 409

Figura 9.16 Algoritmos exigidos para assinatura XML.

que possa ser usado para obter a chave, ou um certificado digital. Um elemento *KeyInfo* pode ser usado para esse propósito.

Tornar possível a usuários em cooperação ajudarem uns aos outros com chaves: desde que o próprio elemento *KeyInfo* não seja vinculado à assinatura por meio de criptografia, as informações podem ser adicionadas sem violar a assinatura digital. Por exemplo, Alice assina um documento e o envia para Bob com um elemento *KeyInfo* especificando apenas o nome da chave. Quando Bob recebe o documento, ele recupera as informações necessárias para validar a assinatura e adiciona isso no elemento *KeyInfo* ao passar o documento para Carol.

O elemento KeyInfo • A segurança em XML especifica um elemento *KeyInfo* para indicar a chave a ser usada para validar uma assinatura ou para decifrar alguns dados. Ele pode conter, por exemplo, certificados digitais, os nomes de chaves ou algoritmos de acordo de chave. Seu uso é opcional: o assinante talvez não queira revelar nenhuma informação sobre chave para todas as pessoas que acessam o documento e, em alguns casos, a aplicação que está usando a segurança em XML já pode ter acesso às chaves em uso.

XML canônica • Algumas aplicações podem fazer alterações que não têm efeito sobre o conteúdo das informações de um documento XML. Isso acontece porque há uma variedade de maneiras de representar o que é logicamente o mesmo documento XML. Por exemplo, os atributos podem estar em ordens diferentes e diferentes codificações de caractere podem ser usadas, embora o conteúdo da informação seja equivalente. A XML canônica [www.w3.org X] foi projetada para uso com assinaturas digitais, as quais são usadas para garantir que o conteúdo das informações de um documento não tenha sido alterado. Os elementos da XML se tornam canônicos antes de serem assinados e o nome do algoritmo de canonização é armazenado junto com a assinatura. Isso permite que o mesmo algoritmo seja usado quando a assinatura é validada.

A forma canônica é um padrão da XML através de uma disposição em série, como um fluxo de bytes. Ela adiciona atributos padrão e remove esquemas supérfluos, colocando os atributos e as declarações de esquema na ordem lexicográfica em cada elemento. Ela usa uma forma padrão para quebras de linha e a codificação UTF-8 para caracteres. Quaisquer dois documentos XML equivalentes têm a mesma forma canônica.

Quando um subconjunto de um documento XML, digamos, um elemento, torna-se canônico, a forma canônica inclui o contexto ancestral, isto é, os espaços de nomes declarados e os valores dos atributos. Assim, quando a XML canônica for usada em conjunto com assinaturas digitais, a assinatura de um elemento não passará em sua validação, caso esse elemento seja colocado em um contexto diferente.

Tipo de algoritmo	*Nome do algoritmo*	*Exigido*	*referência*
Cifra de bloco	TRIPLEDES, AES-128, AES-256	exigido	Seção 11.3.1
	AES-192	opcional	
Codificação	base64	exigido	[Freed e Borenstein 1996]
Transporte de chave	RSA-v1.5, RSA-OAEP	exigido	Seção 11.3.2 [Kaliski e Staddon 1998]
Envoltório de chave simétrica (assinatura por chave compartilhada)	TRIPLEDES KeyWrap, AES-128 KeyWrap, AES--256KeyWrap	exigido	[Housley 2002]
	AES-192 KeyWrap	opcional	
Acordo de chave	Diffie-Hellman	opcional	[Rescorla, 1999]

Figura 9.17 Algoritmos exigidos para criptografia (os algoritmos da Figura 9.16 também são exigidos).

Uma variação desse algoritmo, chamado Exclusive Canonical XML, omite o contexto da disposição em série. Isso poderia ser usado se a aplicação se destinasse a um elemento assinado em particular para ser usado em diferentes contextos.

Uso de assinaturas digitais em XML • A especificação para assinaturas digitais em XML [www.w3.org XII] é uma recomendação do W3C que define novos tipos de elemento XML para conter as assinaturas, os nomes dos algoritmos, as chaves e as referências para informações assinadas. Os nomes fornecidos nessa especificação são definidos no esquema de assinatura da XML, o qual inclui os elementos *Signature*, *SignatureValue*, *SignedInfo* e *KeyInfo*. A Figura 9.16 mostra os algoritmos que devem estar disponíveis em uma implementação de assinatura XML.

Serviço de gerenciamento de chaves • A especificação do serviço de gerenciamento de chaves em XML [www.w3.org XIII] contém protocolos para distribuir e registrar chaves públicas para uso em assinaturas XML. Embora não exija nenhuma infraestrutura de chave pública em particular, o serviço é projetado para ser compatível com as já existentes, por exemplo, certificados X.509 (Seção 11.4.4), SPKI (infraestrutura de chave pública simples, Seção 11.4.4) ou identificadores de chave PGP (Pretty Good Privacy, Seção 11.5.2).

Os clientes podem usar esse serviço para encontrar a chave pública de uma pessoa. Por exemplo, se Alice quiser enviar um *e-mail* cifrado para Bob, ela pode usar esse serviço para obter a chave pública dele. Em outro exemplo, Bob recebe um documento assinado de Alice, contendo o certificado X.509 dela e, então, pede ao serviço de informação de chave para que extraia a chave pública.

Criptografia XML • O padrão para criptografia em XML está definido na recomendação do W3C, que especifica a maneira de representar dados cifrados em XML e o processo para cifrá-los e decifrá-los [www.w3.org XIV]. Ele apresenta um elemento *EncryptedData* para englobar partes de dados cifrados.

A Figura 9.17 especifica os algoritmos de criptografia que devem ser incluídos em uma implementação de criptografia em XML. Algoritmos de cifra de bloco são usados para cifrar os dados, e codificação base64 é usada em XML para representar

1. O cliente solicita ao serviço de agente de viagens informações sobre um conjunto de serviços; por exemplo, voos, aluguel de carro e reservas em hotel.
2. O serviço de agente de viagens reúne as informações sobre preço e disponibilidade e as envia para o cliente, o qual escolhe uma das opções a seguir em nome do usuário:
   (a) refinar a consulta, possivelmente envolvendo mais provedores para obter mais informações; e, em seguida, repetir o passo 2;
   (b) fazer reservas;
   (c) sair.
3. O cliente solicita uma reserva e o serviço de agente de viagens verifica a disponibilidade.
4. *Ou* todos estão disponíveis;
   ou para os serviços que não estão disponíveis;
   ou são oferecidas alternativas para o cliente, que volta para o passo 3;
   ou o cliente volta para o passo 1.
5. Faz o depósito.
6. Fornece ao cliente um número de reserva como confirmação.
7. Durante o período decorrido até o pagamento final, o cliente pode modificar ou cancelar as reservas.

Figura 9.18 Cenário do agente de viagens.

assinaturas digitais e dados cifrados. Algoritmos de transporte de chave são algoritmos de criptografia de chave pública projetados para uso na criptografia e decifração das chaves em si.

Os algoritmos de envoltório de chave simétrica são algoritmos de criptografia de chave secreta compartilhada, projetados para cifrar e decifrar chaves simétricas por meio de outra chave. Isso poderia ser usado se uma chave precisasse ser incluída em um elemento *KeyInfo*.

Um algoritmo de acordo de chave permite que uma chave secreta compartilhada seja extraída a partir de um cálculo em duas chaves públicas. Esse algoritmo se tornou disponível para uso por aplicações que precisam concordar com uma chave compartilhada sem nenhuma troca. Ele não é aplicado pelo sistema de segurança da XML em si.

9.6 Coordenação de serviços Web

A infraestrutura SOAP suporta interações requisição-resposta entre clientes e serviços Web. Entretanto, muitas aplicações úteis envolvem várias requisições que precisam ser executadas em uma ordem em particular. Por exemplo, ao se fazer reservas para um voo, são reunidas as informações sobre preço e disponibilidade, antes que as reservas sejam feitas. Quando um usuário interage com páginas Web por intermédio de um navegador, por exemplo, para fazer reserva em um voo, ou para dar um lance em um leilão, a interface fornecida pelo navegador, que é baseada nas informações fornecidas pelo servidor, controla a sequência em que as operações são executadas.

Entretanto, se for um serviço Web que estiver fazendo reservas, como o serviço de agente de viagens mostrado na Figura 9.2, esse serviço precisará trabalhar a partir de

uma descrição da maneira apropriada para prosseguir ao interagir com outros serviços que realizam, por exemplo, aluguel de carros e reservas em hotel, assim como reservas em voos. A Figura 9.18 mostra um exemplo de tal descrição.

Esse exemplo ilustra a necessidade de serviços Web, como clientes, de receberem uma descrição de um protocolo em particular a ser seguido ao interagir com outros serviços Web. Contudo, também existe o problema da manutenção da consistência nos dados do servidor quando ele está recebendo e respondendo a requisições de vários clientes. Os Capítulos 16 e 17 discutirão as transações, ilustrando os problemas por meio de uma série de transações bancárias. Como um exemplo simples, em uma transferência de dinheiro entre duas contas bancárias, a consistência exige que o depósito em uma conta e o saque da outra devam ser realizados. O Capítulo 17 apresentará o protocolo de *commit* (confirmação) em duas fases, que é usado por servidores em cooperação para garantir a consistência de transações.

Em alguns casos, transações atômicas são adequadas aos requisitos de aplicações que usam serviços Web. Entretanto, atividades como aquelas do agente de viagens demoram um longo tempo para terminar e seria impraticável usar um protocolo de *commit* em duas fases para executá-las, pois envolve manter recursos bloqueados por longos períodos de tempo. Uma alternativa é usar um protocolo mais relaxado, no qual cada participante faz alterações no estado persistente, quando elas ocorrem. No caso de falha, um protocolo em nível de aplicação é usado para desfazer essas ações.

No *middleware* convencional, a infraestrutura fornece um protocolo de requisição-resposta simples, deixando outros serviços, como transações, persistência e segurança, serem implementados como serviços separados de mais alto nível, que podem ser usados quando forem necessários. O mesmo vale para serviços Web, em que o W3C e outros vêm trabalhando na definição de serviços de mais alto nível.

Tem-se trabalhado em um modelo geral para a coordenação de serviços Web, o qual é semelhante ao modelo de transação distribuída descrito na Seção 17.2, pois tem funções de coordenador e participante que são capazes de atuar em protocolos específicos, por exemplo, para executar uma transação distribuída. Esse trabalho, que é chamado WS-Coordination, é descrito por Langworthy [2004]. O mesmo grupo também mostrou como transações podem ser executadas dentro desse modelo. Para um estudo amplo sobre protocolos de coordenação de serviços Web, consulte Alonso *et al*. [2004].

No restante desta seção, descreveremos, em linhas gerais, as ideias ligadas à coreografia de serviço Web. Considere o fato de que seria possível descrever todos os caminhos alternativos válidos possíveis pelo conjunto de interações entre pares de serviços Web, trabalhando em uma tarefa conjunta, como o cenário do agente de viagens. Se tal descrição estivesse disponível, ela poderia ser usada como auxiliar na coordenação de tarefas conjuntas. Ela também poderia ser usada como uma especificação a ser seguida por novas instâncias de um serviço, como um novo serviço de reservas de voo que quisesse se juntar a uma colaboração.

O W3C usa o termo *coreografia* para se referir a uma linguagem baseada na WSDL para definir a coordenação. Por exemplo, a linguagem poderia especificar restrições sobre a ordem e as condições em que as mensagens são trocadas pelos participantes. Uma coreografia se destina a fornecer uma descrição global de um conjunto de interações, mostrando o comportamento de cada membro de um conjunto de participantes, visando melhorar a interoperabilidade.

Requisitos da coreografia • A coreografia se destina a suportar interações entre serviços Web que geralmente são gerenciados por diferentes empresas e organizações. Uma colaboração envolvendo vários serviços Web e clientes deve ser descrita em termos dos conjuntos de interações observáveis entre pares delas. Tal descrição poderia ser vista como um contrato entre os participantes. Ela poderia ser usada da seguinte forma:

- para gerar esboços de código para um novo serviço que quisesse participar;
- como base para gerar mensagens de teste para um novo serviço;
- para promover um entendimento comum da colaboração;
- para analisar a colaboração, por exemplo, para identificar possíveis situações de impasse.

O uso de uma descrição de coreografia comum por um conjunto de serviços Web em colaboração deve resultar em serviços mais robustos, com melhor interoperabilidade. Além disso, deve ser mais fácil desenvolver e introduzir novos serviços, tornando o serviço global mais útil.

O documento da minuta do trabalho do W3C, no endereço [www.w3.org XV], sugere que uma linguagem de coreografia deve incluir as seguintes características:

- composição hierárquica e recursiva das coreografias;
- a capacidade de adicionar novas instâncias de um serviço existente e novos serviços;
- caminhos concorrentes, caminhos alternativos e a capacidade de repetir uma seção de uma coreografia;
- tempos limites variáveis – por exemplo, diferentes períodos para manter reservas;
- exceções, por exemplo, para tratar das mensagens que chegam fora de sequência e ações do usuário, como cancelamentos;
- interações assíncronas (*callbacks*);
- passagem de referência, por exemplo, para permitir que uma operadora de aluguel de carros consulte um banco para uma verificação de crédito em nome de um usuário;
- marcação dos limites das transações separadas que ocorrem, por exemplo, para permitir recuperação;
- a capacidade de incluir documentação legível para seres humanos.

Um modelo baseado nesses requisitos está descrito em outro documento de minuta de trabalho do W3C [www.w3.org XVI].

Linguagens de coreografia • A intenção é produzir uma linguagem declarativa, baseada em XML, para definir coreografias que possam usar definições WSDL. O W3C elaborou uma recomendação para *Web Services Choreography Definition Language Version 1* [www.w3.org XVII]. Antes disso, um grupo de empresas enviou para o W3C uma especificação para a interface de coreografia de serviços Web [www.w3.org XVIII].

9.7 Aplicações de serviços Web

Atualmente, os serviços Web representam o paradigma dominante para programação de sistemas distribuídos. Nesta seção, discutiremos diversas áreas importantes em que os serviços Web têm sido extensivamente empregados: no suporte para arquitetura orientada a serviços, em grade (*grid*) e, recentemente, na computação em nuvem.

9.7.1 Arquitetura orientada a serviços

A *arquitetura orientada a serviços* (*SOA*, *Service-Oriented Architecture*) é um conjunto de princípios de projeto por meio do qual os sistemas distribuídos são desenvolvidos usando-se conjuntos de serviços pouco acoplados que podem ser descobertos dinamicamente e, então, comunicar-se uns com os outros, ou que são coordenados por meio de coreografia para fornecer serviços aprimorados. O modelo de arquitetura orientada a serviços é um conceito abstrato que pode ser implementado usando-se uma variedade de tecnologias, incluindo as estratégias de objeto distribuído ou componente, discutidas no Capítulo 8. Contudo, a principal maneira de concretizar a arquitetura orientada a serviços é por meio do uso de serviços Web, em grande medida devido ao baixo acoplamento inerente a essa estratégia (conforme discutido na Seção 9.2).

Esse estilo de arquitetura pode ser usado dentro de uma empresa ou organização para oferecer uma arquitetura de *software* flexível e obter interoperabilidade entre os vários serviços. No entanto, seu uso mais importante é na Internet, oferecendo uma visão comum dos serviços, tornando-os globalmente acessíveis e receptivos à composição subsequente. Isso torna possível transcender os níveis de heterogeneidade inerentes à Internet e também lidar com o problema de diferentes organizações adotarem internamente diferentes produtos de *middleware* – é possível que uma organização use CORBA e outra use .NET, mas que então ambas exponham interfaces usando serviços Web, estimulando, assim, a interoperabilidade global. A propriedade resultante é conhecida como *integração de empresa para empresa (B2B*, *business-to-business*). Já vimos um exemplo de necessidade da integração B2B, na Figura 9.18 (o cenário do agente de viagens), em que o agente pode lidar com uma ampla variedade de empresas que oferecem voos, carros e acomodação em hotel.

A arquitetura orientada a serviços também possibilita e estimula uma estratégia de *mashup* para desenvolvimento de *software*. *Mashup* é um novo serviço criado por um desenvolvedor externo, que combina dois ou mais serviços disponíveis no ambiente distribuído. A cultura do *mashup* conta com a pronta disponibilidade de serviços úteis, com interfaces bem definidas, complementados com uma comunidade de inovação aberta na qual indivíduos ou grupos se envolvem no desenvolvimento de serviços combinados experimentais e os tornam disponíveis para outros, para mais desenvolvimento. Agora as duas condições são satisfeitas pela Internet, particularmente com o surgimento da computação em nuvem como um serviço (conforme apresentado na Seção 7.7.1), na qual importantes desenvolvedores de *software*, como Amazon, Flickr e eBay, tornam serviços disponíveis para outros desenvolvedores por meio de interfaces publicadas. Como exemplo, consulte JBidwatcher [www.jbidwatcher.org], um *mashup* baseado em Java que faz interface com o eBay para gerenciar ofertas de maneira mais proativa em nome de um cliente, por exemplo, monitorando leilões e dando lances no último minuto para maximizar as chances de sucesso.

9.7.2 A grade

O termo grade (*grid*) é usado para se referir ao *middleware* projetado para permitir o compartilhamento de recursos como arquivos, computadores, *software*, dados e sensores em uma escala muito grande. Normalmente, os recursos são compartilhados por grupos de usuários em diferentes organizações, os quais estão colaborando na solução de problemas que exigem grandes números de computadores para resolvê-los, pelo compartilhamento de dados ou pelo compartilhamento de poder de computação. Esses recursos são necessariamente suportados por *hardware*, sistemas operacionais, lingua-

O World-Wide Telescope: uma aplicação de grade

Esse projeto está relacionado à distribuição de recursos de dados compartilhados pela comunidade de astronomia. Ele está descrito no trabalho de Szalay e Gray [2004], Szalay e Gray [2001] e Gray e Szalay [2002]. Os dados astronômicos consistem em repositórios de arquivo de observações, cada um dos quais abrangendo um período de tempo em particular, uma parte do espectro eletromagnético (ótico, raios X, rádio) e uma área em particular do céu. Essas observações são feitas por diferentes instrumentos instalados em vários lugares do mundo.

Um estudo sobre como os astrônomos compartilham seus dados é útil para extrair as características de uma aplicação típica de grade, pois eles compartilham livremente seus resultados uns com os outros e os problemas de segurança podem ser omitidos, tornando esta discussão mais simples.

Os astrônomos fazem estudos que precisam combinar dados sobre os mesmos objetos celestes, mas que envolvem períodos de tempo diferentes e diversas partes do espectro. A capacidade de usar observações de dados independentes é importante para a pesquisa. A visualização permite aos astrônomos verem os dados como mapas de dispersão bi ou tridimensionais.

As equipes que reúnem os dados os armazenam em imensos repositórios de arquivo (atualmente, na ordem de terabytes), os quais são gerenciados de forma local por cada equipe. Os instrumentos usados na coleta de dados estão sujeitos à lei de Moore. Por isso, o volume dos dados reunidos cresce exponencialmente. À medida que são reunidos, os dados brutos são analisados e processados em uma sequência de passos e armazenados como dados derivados para uso pelos astrônomos de todo o mundo. Porém, antes que os dados possam ser usados por outros pesquisadores, os cientistas que trabalham em um campo em particular precisam concordar com uma maneira comum de rotular seus dados.

Szalay e Gray [2004] mencionam que, no passado, os dados de pesquisa científica eram incluídos pelos autores em artigos e publicados em periódicos que ficavam em bibliotecas. Contudo, atualmente, o volume de dados é grande demais para ser incluído em uma publicação. Isso se aplica não apenas à astronomia, mas também aos campos da física de partícula, genoma e pesquisa biológica. A função do autor agora se relaciona às colaborações, o qual leva de 5 a 10 anos para construir sua experiência, antes de produzir os dados que são publicados para o mundo em repositórios de arquivo baseados na Web. Assim, os cientistas que trabalham em projetos se tornam geradores (anunciantes) de dados e bibliotecários, assim como autores.

Essa função adicional exige que todo projeto que gerencia um repositório de arquivo de dados o torne acessível para outros pesquisadores. Isso implica uma sobrecarga considerável, além da tarefa original de análise dos dados. Para tornar tal compartilhamento possível, os dados brutos exigem metadados para descrever, por exemplo, a hora em que foram coletados, a parte do céu observada e o instrumento utilizado. Além disso, os dados derivados precisam ser acompanhados de metadados que descrevam os parâmetros das operações com os quais foram processados.

O cálculo dos dados derivados exige suporte computacional pesado. Frequentemente, eles precisam ser recalculados, à medida que as técnicas melhoram. Tudo isso é uma despesa considerável para o projeto que possui os dados.

O objetivo do World-Wide Telescope é unificar os repositórios de arquivo de astronomia do mundo em um banco de dados gigante, contendo literatura, imagens, dados brutos, conjuntos de dados derivados e dados de simulação de astronomia.

gens de programação e aplicações heterogêneas. É necessário gerenciamento para coordenar o uso de recursos para garantir que os clientes obtenham o que precisam e que os serviços possam fornecê-los. Em alguns casos, são exigidas técnicas de segurança sofisticadas para garantir o uso correto dos recursos nesse tipo de ambiente. Para um

exemplo de aplicação de grade, veja o quadro, que apresenta o aplicativo World-Wide Telescope, desenvolvido na Microsoft Research.

Requisitos das aplicações de grade • O World-Wide Telescope é um caso típico de uma variedade de *aplicações de grade com uso intensivo de dados*, em que:

- os dados são reunidos por meio de instrumentos científicos;
- os dados são armazenados em repositórios de arquivo em um conjunto de *sites* separados, cujas localizações podem ser em diferentes lugares, em qualquer parte do mundo;
- os dados são gerenciados por equipes de cientistas pertencentes a organizações separadas;
- um volume imenso e cada vez maior (terabytes ou petabytes) de dados brutos é gerado a partir dos instrumentos;
- programas de computador serão usados para analisar e fazer resumos dos dados brutos, por exemplo, para classificar, calibrar e catalogar os dados brutos que representam objetos celestes.

A Internet torna todos esses repositórios de arquivo de dados potencialmente disponíveis para cientistas de todo o mundo. Eles poderão obter dados de diferentes instrumentos, extraídos em diferentes momentos e em diferentes locais. Entretanto, um cientista em particular, usando esses dados para sua própria pesquisa, estará interessado apenas em um subconjunto dos objetos presentes nos repositórios de arquivo.

O imenso volume de dados em um repositório de arquivo torna impraticável transferi-los para o local do usuário, antes de processá-los para extrair os objetos de interesse, devido às considerações sobre o tempo de transmissão e o espaço no disco local exigidos. Portanto, não é apropriado usar FTP ou acesso Web nesse contexto. O processamento dos dados brutos deve ocorrer no local onde eles são reunidos e armazenados em um banco de dados. Então, quando um cientista fizer uma consulta sobre objetos em particular, as informações presentes em cada banco de dados devem ser analisadas e, se necessário, visualizações devem ser produzidas, antes de retornar os resultados para a consulta remota.

O fato de os dados serem processados em muitos *sites* diferentes proporciona um paralelismo que efetivamente divide a imensa tarefa que está sendo executada.

A partir das características anteriores, são inferidos os seguintes requisitos:

R1: Acesso remoto aos recursos – isto é, às informações exigidas presentes nos repositórios de arquivo.

R2: Processamento de dados no local onde eles são armazenados e gerenciados, ou quando são reunidos ou em resposta a uma requisição. Uma consulta típica poderia resultar em uma visualização baseada nos dados reunidos para uma região do céu, gravados por diferentes instrumentos, em diferentes momentos. Isso envolverá selecionar um pequeno volume de dados de cada repositório de arquivo de dados maciço.

R3: O gerenciador de recursos de um repositório de arquivo de dados deve ser capaz de criar instâncias de serviço dinamicamente para lidar com a seção de dados em particular exigida, exatamente como no modelo de objeto distribuído, em que serventes são criados, quando necessário, para manipular diferentes recursos gerenciados por um serviço.

R4: Metadados para descrever:

- características dos dados em um repositório de arquivo, por exemplo, para astronomia: a área do céu, a data e a hora da coleta e os instrumentos utilizados;
- as características de um serviço que esteja gerenciando esses dados, por exemplo, seu custo, sua localização geográfica, seu gerador (anunciante) ou sua carga ou espaço disponível.

R5: Serviços de diretório baseados nos metadados acima.

R6: *Software* para gerenciar consultas, transferências de dados e reserva antecipada de recursos, levando em conta que os recursos geralmente são gerenciados pelos projetos que geram os dados e que o acesso a eles talvez precise ser racionado.

Os serviços Web podem tratar dos dois primeiros requisitos, fornecendo uma maneira conveniente de permitir que os cientistas acessem operações sobre dados em repositórios de arquivo remotos. Isso exigirá que cada aplicação em particular forneça uma descrição do serviço que inclua um conjunto de métodos para acessar seus dados. O *middleware* de grade deve tratar dos requisitos restantes.

As grades também são usadas para *aplicações de uso intensivo de poder computacional*, como no processamento do enorme volume de dados produzido pelo acelerador de partículas de alta energia CMS no CERN [www.uscms.org], no teste dos efeitos de moléculas de possíveis remédios [Taufer *et al*. 2003, Chien 2004] ou no suporte de jogos *online* para muitos jogadores usando a capacidade ociosa em grupos de computadores [www.butterfly.net]. Quando aplicações de uso intensivo de poder computacional forem distribuídas em uma grade, o gerenciamento de recursos se preocupará com a alocação de recursos de computação e com a harmonização das cargas.

Finalmente, muitas aplicações em grade como, por exemplo, as médicas e as comerciais, precisarão considerar aspectos de segurança. Mesmo quando a privacidade dos dados não é problema, será importante estabelecer a identidade das pessoas que criaram os dados.

Middleware de grade • A arquitetura aberta de serviços de grade (OGSA, Open Grid Services Architecture) é um padrão para aplicações baseadas em grade [Foster *et al*. 2001, 2002]. Ela fornece uma estrutura em que os requisitos anteriores podem ser satisfeitos, baseada em serviços Web. Os recursos são gerenciados por serviços de grade específicos da aplicação. O *toolkit* Globus implementa a arquitetura.

O Projeto Globus começou em 1994 com o objetivo de fornecer *software* que integrasse e padronizasse as funções exigidas por uma família de aplicações científicas. Essas funções incluem serviços de diretório, segurança e gerenciamento de recursos. O primeiro *toolkit* Globus apareceu em 1997. O OGSA evoluiu a partir da segunda versão do *toolkit*, chamado de GT2, que está descrito em Foster e Kesselman [2004]. A terceira versão (GT3), que apareceu em 2002, foi baseada no OGSA e, portanto, construída em serviços Web. Ela foi desenvolvida pela Globus Alliance (www.globus.org) e está descrita em Sandholm e Gawor [2003]. Desde então, foram lançadas mais duas versões – a mais recente é denominada GT5 e está disponível como *software* de código-fonte aberto [www.globus.org].

Um estudo de caso da OGSA e do *toolkit* Globus (até o GT3) pode ser encontrado no *site* que acompanha o livro [www.cdk5.net/Web] (em inglês).

9.7.3 Computação em nuvem

A computação em nuvem foi apresentada no Capítulo 1 como um conjunto de serviços de aplicativo, armazenamento e computação baseados na Internet, suficientes para suportar a maioria das necessidades dos usuários, permitindo com isso que, em grande medida ou totalmente, eles prescindam de armazenamento de dados e de *software* aplicativo locais. A computação em nuvem também promove a visão de tudo como serviço, desde infraestrutura física ou virtual até *software*, frequentemente pago de acordo com a utilização, em vez de ser adquirido. Portanto, o conceito está intrinsecamente ligado a um novo modelo comercial de computação, no qual os fornecedores da nuvem oferecem diversos serviços computacionais, de dados e outros, para os clientes, conforme o exigido para seu uso diário – oferecendo, por exemplo, pela Internet, capacidade de armazenamento suficiente para atuar como serviço de repositório de arquivos ou *backup*.

O Capítulo 1 também comentou a sobreposição entre computação em nuvem e a grade. O desenvolvimento da grade precedeu o surgimento da computação em nuvem e foi um fator significativo para sua aparição. Elas têm o mesmo objetivo de fornecer recursos (serviços) na Internet. Enquanto a grade tende a se concentrar em aplicações de uso intensivo de dados de alta capacidade ou computacionalmente dispendiosas, a computação em nuvem é mais geral, oferecendo diversos serviços para computadores individuais e usuários finais. O modelo comercial associado à computação em nuvem também é uma característica distintiva. Portanto, é justo dizer que a grade é um exemplo primitivo de computação em nuvem; no entanto, desde então, a computação em nuvem se desenvolveu significativamente.

Com essa visão de tudo ser serviço, os serviços Web oferecem um caminho de implementação natural para computação em nuvem e, realmente, muitos fornecedores entram por ele. O produto mais notável nesse setor é o AWS (Amazon Web Services) [aws.amazon.com] – examinaremos brevemente essa tecnologia, a seguir. Veremos uma estratégia alternativa à computação em nuvem, no Capítulo 21, quando examinarmos a infraestrutura do Google e o mecanismo Google App Engine associado, os quais apresentam uma estratégia mais leve e de maior desempenho do que os serviços Web.

Amazon Web Services é um conjunto de serviços em nuvem implementados na extensa infraestrutura física pertencente à Amazon.com. Originalmente desenvolvida para propósitos internos no suporte para suas vendas a varejo eletrônicas, agora a Amazon oferece muitos dos recursos para usuários externos, permitindo a eles executar serviços independentes na sua infraestrutura. A implementação do AWS trata dos principais problemas dos sistemas distribuídos, como o gerenciamento da disponibilidade de serviço, escalabilidade e desempenho, permitindo que os desenvolvedores se concentrem no uso de seus serviços. Os serviços são disponibilizados usando-se os padrões de serviço Web descritos anteriormente neste capítulo. Isso tem a vantagem de que os programadores que conhecem os serviços Web podem usar o AWS prontamente e desenvolver *mashups* que incorporam Amazon Web Services em sua construção. Mais geralmente, a estratégia permite interoperabilidade na Internet. A Amazon também adota a estratégia REST, conforme defendida por Fielding [2000] e discutida na Seção 9.2.

A Amazon oferece um conjunto de serviços amplo e extensível, dos quais os mais importantes estão listados na Figura 9.19. Apresentamos o EC2 com mais detalhes. O EC2 é um serviço de computação elástico, no qual o termo *elástico* se refere à proprie-

Serviço web	*Descrição*
Amazon Elastic Compute Cloud (EC2)	Serviço baseado na Web que oferece acesso a máquinas virtuais de determinado desempenho e capacidade de armazenamento
Amazon Simple Storage Service (S3)	Serviço de armazenamento baseado na Web para dados não estruturados
Amazon Simple DB	Serviço de armazenamento baseado na Web para consultar dados estruturados
Amazon Simple Queue Service (SQS)	Serviço hospedado que suporta filas de mensagens (conforme discutido no Capítulo 6)
Amazon Elastic MapReduce	Serviço baseado na web para computação distribuída usando o modelo MapReduce (apresentado no Capítulo 21)
Amazon Flexible Payments Service (FPS)	Serviço baseado na Web que suporta pagamentos eletrônicos

Figura 9.19 Uma seleção de Amazon Web Services.

dade de oferecer capacidade de computação redimensionada de acordo com as necessidades do cliente. Em vez de uma máquina real, o EC2 oferece ao usuário uma máquina virtual, denominada *instância*, de acordo com a especificação desejada. Por exemplo, um usuário pode solicitar uma instância dos seguintes tipos:

- uma *instância padrão*, projetada para ser adequada para a maioria das aplicações;
- uma *instância de grande capacidade de memória*, que oferece capacidade de memória adicional, por exemplo, para aplicações que envolvem o uso de cache;
- uma instância de *grande capacidade de CPU*, projetada para suportar tarefas de computação intensa;
- uma *instância de computação em grupo*, que oferece um grupo de processadores virtuais com interconexão de alta largura de banda para tarefas de computação de alto desempenho.

Vários deles podem ser ainda mais refinados – por exemplo, para uma instância padrão é possível solicitar uma instância pequena, média ou grande, representando diferentes especificações em termos de poder de processamento, memória, armazenamento em disco, etc.

O ECS é construído sobre o hipervisor Xen, descrito na Seção 7.7.2. As instâncias podem ser configuradas para executar diversos sistemas operacionais, incluindo Windows Server 2008, Linux ou OpenSolaris. Também podem ser configuradas com uma variedade de *software*. Por exemplo, é possível solicitar uma instalação de Apache HTTP para suportar hospedagem na Web.

O EC2 suporta o interessante conceito de endereço IP elástico, que parece um endereço IP tradicional, mas é associado à conta do usuário e não a uma instância em particular. Isso significa que, se uma máquina (virtual) falha, o endereço IP pode ser reatribuído a uma máquina diferente, sem exigir a intervenção de um administrador de rede.

9.8 Resumo

Neste capítulo, mostramos que os serviços Web surgiram da necessidade de fornecer uma infraestrutura para suportar interligação em rede entre diferentes organizações. Essa infraestrutura geralmente usa o conhecido protocolo HTTP para transportar mensagens entre clientes e servidores pela Internet e é baseada no uso de URIs para se referir aos recursos. A XML, um formato textual, é usada para representação e empacotamento de dados.

Duas influências distintas levaram à aparição de serviços Web. Uma delas é a adição de interfaces de serviço em servidores Web com o objetivo de permitir que os recursos de um *site* fossem acessados por programas clientes que não os navegadores, além de usar uma forma de interação mais rica. A outra é o desejo de fornecer algo como RPC na Internet, com base nos protocolos existentes. Os serviços Web resultantes fornecem interfaces com conjuntos de operações que podem ser chamadas de forma remota. Assim como qualquer outra forma de serviço, um serviço Web pode ser cliente de outro serviço Web, permitindo que um serviço Web integre um conjunto de outros serviços Web ou combine com ele.

SOAP é o protocolo de comunicação geralmente usado pelos serviços Web e seus clientes. Ele pode ser usado para transmitir mensagens de requisição e suas respostas entre cliente e servidor, ou pela troca assíncrona de documentos ou por uma forma de protocolo de requisição-resposta baseado em um par de trocas de mensagens assíncronas. Nos dois casos, a mensagem de requisição ou resposta é incluída em um documento formatado em XML, chamado de envelope. Geralmente, o envelope SOAP é transmitido por meio do protocolo HTTP síncrono, embora outros transportes possam ser usados.

Processadores de XML e SOAP estão disponíveis para todas as linguagens de programação e sistemas operacionais amplamente usados. Isso permite que os serviços Web e seus clientes sejam implantados em quase qualquer lugar. Essa forma de interligação em rede é possível pelo fato de que os serviços Web não estão ligados a nenhuma linguagem de programação em particular nem suportam o modelo de objeto distribuído.

Nos *middlewares* convencionais as definições de interface fornecem aos clientes os detalhes dos serviços. Entretanto, no caso de serviços Web, são usadas descrições de serviço. Uma descrição de serviço especifica o protocolo de comunicação a ser usado (por exemplo, SOAP) e o URI do serviço, assim como descreve sua interface. A interface pode ser descrita como um conjunto de operações, ou como um conjunto de mensagens, a serem trocadas entre cliente e servidor.

A segurança em XML foi projetada para fornecer a proteção necessária para o conteúdo de um documento trocado por membros de um grupo de pessoas, as quais têm tarefas diferentes para realizar nesse documento. Diferentes partes do documento estarão disponíveis para diferentes pessoas, algumas com a capacidade de adicionar ou alterar o conteúdo e outras de lê-lo. Para permitir uma flexibilidade completa em seu uso futuro, as propriedades de segurança são definidas dentro do próprio documento. Isso é obtido por meio da XML, que tem um formato auto-descritivo. Elementos da XML são usados para especificar as partes do documento que são cifradas ou assinadas, assim como os detalhes dos algoritmos usados e informações para ajudar a encontrar chaves.

Os serviços Web têm sido usados para diversos propósitos nos sistemas distribuídos. Por exemplo, eles fornecem uma implementação natural do conceito de arquitetura orientada a serviços, na qual seu baixo acoplamento permite interoperabilidade em apli-

cações na Internet – incluindo as de empresa para empresa (B2B). Seu baixo acoplamento inerente também suporta o surgimento de uma estratégia de *mashup* para a construção de serviços Web. Os serviços Web também servem de base para a grade, suportando colaborações entre cientistas ou engenheiros em organizações de diferentes partes do mundo. Muito frequentemente, seu trabalho é baseado no uso de dados brutos coletados por instrumentos em diferentes lugares e, depois, processados de forma local. O *toolkit* Globus é uma implementação da arquitetura que tem sido usada em uma variedade de aplicações de uso intenso de dados e poder computacional. Por fim, os serviços Web são expressivamente usados na computação em nuvem. Por exemplo, o AWS da Amazon é totalmente baseado em padrões de serviço Web, acoplados à filosofia REST de construção de serviço.

Exercícios

9.1 Compare o protocolo de requisição-resposta descrito na Seção 5.2 com a implementação de comunicação cliente-servidor no SOAP. Cite dois motivos pelos quais o uso de mensagens assíncronas pelo SOAP é mais apropriado para uso na Internet. Até que ponto o uso de HTTP pelo SOAP reduz a diferença entre as duas estratégias? *páginas 388*

9.2 Compare a estrutura dos URLs, conforme usados pelos serviços Web, com a das referências de objeto remoto, conforme especificadas na Seção 4.3.4. Cite, em cada caso, como elas são usadas para executar um pedido do cliente. *páginas 393*

9.3 Ilustre o conteúdo de uma mensagem SOAP de requisição e de sua mensagem de resposta correspondente para o serviço de *Election* dado no Exercício 5.11. Use, na sua resposta, uma versão pictórica da XML, como mostrado nas Figuras 9.4 e 9.5. *página 389*

9.4 Descreva em linhas gerais os cinco principais elementos de uma descrição do serviço WSDL. No caso do serviço *Election* definido no Exercício 5.11, cite o tipo de informação a ser usada pelas mensagens de requisição e de resposta – algum deles precisa ser incluído no espaço de nomes de destino? Para a operação *vote*, desenhe diagramas semelhantes às Figuras 9.11 e 9.13. *página 402*

9.5 Continuando com o exemplo do serviço *Election*, explique por que a parte da WSDL definida no Exercício 9.4 é referida como "abstrata". O que precisaria ser adicionado na descrição do serviço para torná-lo completamente concreto? *página 400*

9.6 Defina uma interface Java para o serviço *Election*, conveniente para uso como um serviço Web. Diga por que você acha que a interface que definiu é conveniente. Explique como um documento WSDL para o serviço é gerado e como se torna disponível para os clientes. *página 396*

9.7 Descreva o conteúdo de um *proxy* de cliente Java para o serviço *Election*. Explique como os métodos de empacotamento e desempacotamento podem ser obtidos para um *proxy* estático. *página 396*

9.8 Explique a função de um contêiner de *servlet* na distribuição de um serviço Web e na execução de uma requisição de cliente. *página 396*

9.9 No exemplo em Java ilustrado nas Figuras 9.8 e 9.9, o cliente e o servidor estão lidando com objetos, embora os serviços Web não suportem objetos distribuídos. Como isso pode acontecer? Quais são as limitações impostas sobre as interfaces de serviços Web em Java? *página 395*

9.10 Descreva, em linhas gerais, o esquema de replicação usado no UDDI. Supondo que carimbos de tempo vetoriais são usados para suportar esse esquema, defina duas operações para uso por registros que precisem trocar dados. *página 406*

9.11 Explique por que o UDDI pode ser descrito como serviço de nome e como serviço de diretório, mencionando os tipos de perguntas que podem ser feitas. O segundo "D" no nome UDDI se refere a descoberta – o UDDI é realmente um serviço de descoberta? *Capítulo 13 e página 404*

9.12 Descreva, em linhas gerais, a principal diferença entre TLS e segurança em XML. Explique por que a XML é particularmente conveniente para a função que desempenha, em termos dessas diferenças. *Capítulo 11 e página 406*

9.13 Os documentos protegidos pela segurança em XML podem ser assinados ou cifrados muito tempo antes que alguém possa prever quem serão os destinatários finais. Quais medidas são adotadas para garantir que estes últimos tenham acesso aos algoritmos usados pelo primeiro? *página 406*

9.14 Explique a relevância da XML canônica nas assinaturas digitais. Quais informações contextuais podem ser incluídas na forma canônica? Dê um exemplo de brecha de segurança em que o contexto é omitido da forma canônica. *página 409*

9.15 Um protocolo de coordenação poderia ser executado para coordenar as ações dos serviços Web. Descreva em linhas gerais uma arquitetura para (i) um protocolo centralizado e (ii) um protocolo de coordenação distribuída. Em cada caso, descreva as interações necessárias para estabelecer a coordenação entre dois serviços Web. *página 411*

9.16 Compare a semântica de chamada RPC com a semântica do *WS-ReliableMessaging:*

i) Cite as entidades às quais cada uma se refere.

ii) Compare os diferentes significados da semântica disponível (por exemplo, *pelo menos uma vez, no máximo uma vez, exatamente uma vez*). *Capítulo 5 e página 392*