FIAP ON

WEBSERVICES &
RESTFUL TECHNOLOGIES

XML
XSD
XSLT
SOAP

EDUARDO GALEGO

2

LISTA DE COMANDOS DE PROMPT DO SISTEMA OPERACIONAL

SUMÁRIO

2 XML – XSD – XSLT – SOAP

Por se tratar de uma tecnologia um pouco mais antiga (comparada ao REST), por vezes os Web Services SOAP podem ser encontrados em sistemas legados e suites SOA (conjunto de ferramentas que viabiliza a Arquitetura Orientada a Serviços).

Então, você talvez esteja pensando: “Por que devo estudar uma tecnologia defasada?”

Em certas ocasiões, desenvolver Web Services em SOAP é a melhor escolha. Exemplo disso é o caso da Nota Fiscal Eletrônica.

A escolha por trás da tecnologia pode estar no fato de que SOAP contém diversas especificações que estendem suas funcionalidades, como controle de transacionalidade, encriptografia dos dados e assinatura digital.

Se precisar de transações compatíveis com o conceito ACID (*Atomicity*, *Consistency*, *Isolation*, *Durability*), Web Service SOAP pode ser uma boa escolha!

Web Services SOAP são fortemente tipados e orientados a uma interface pública e bem definida. Assim, se precisar de controle total sobre os documentos que são enviados e recebidos (como é o caso da NFe), SOAP também pode ser uma boa alternativa.

Por existir há mais tempo, há uma gama de softwares que já trabalham com SOAP. É o caso das Suites SOA que, em sua base, implementam todo o controle sobre documentos XML.

É inegável o crescimento de outras tecnologias Web Services, mas enquanto uma delas não se torna tão madura quanto o SOAP, não podemos desconsiderá-lo em nossos projetos de software.

2.1 XML

Todos os dados trafegados por um Web Service SOAP são em formato XML. Assim, precisamos conhecê-lo bem!

XML é o acrônimo de *eXtensible Markup Language*. Ele dispõe os dados utilizando marcadores, atributos e valores. Sua estrutura é em forma de árvore. Em

outras palavras, o valor de um marcador pode ser um ou mais marcadores. Atualmente a especificação encontra-se na versão 1.1 e é mantido pela W3C.

A seguir pode-se observar o mesmo exemplo do capítulo anterior, mas com alguns comentários adicionais.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Insured CustomerID="5">
   <CarPolicy PolicyType="Auto">
      <Vehicle Category="Sedan">
         <Make>Honda</Make>
         <Model>Accord</Model>
      </Vehicle>
      <Vehicle Category="Sport">
         <Make>Ford</Make>
         <Model>Mustang</Model>
      </Vehicle>
   </CarPolicy>
   <CarPolicy PolicyType="Antique">
      <Vehicle Category="Sport">
         <Make>Triumph</Make>
         <Model>Spitfire</Model>
      </Vehicle>
   </CarPolicy>
</Insured>
```

Comando de prompt 2.1 – Exemplo de um XML de dados de veículos
Fonte: Elaborado pelo autor (2021)

Não custa dizer que a indentação foi adicionada para facilitar a leitura humana. Os interpretadores de XML ignoram as tabulações, quebras de linhas e espaços entre os marcadores.

Na linha 1, encontramos os **metadados** do arquivo XML, como versão do XML utilizado e encode do arquivo.

Na linha 2, temos o primeiro **marcador** chamado *Insured,* que é encerrado na linha 19 por um marcador de mesmo valor, mas com uma barra no início. Marcadores sem valores adicionam a barra ao final (exemplo: `<Vehicle/>`).

O que temos logo em seguida ao marcador (*CustomerId*) é chamado de **atributo**. São características que são associadas a um marcador. Marcadores sempre são representados como chave=valor.

A partir da linha 3, temos o conteúdo do marcador *Insured*. Repare que temos um outro marcador como valor. Assim, podemos considerar esse marcador como sendo um **tipo complexo**.

Diferentemente do *Insered*, o marcador da linha 5, chamado *Make,* tem em seu conteúdo um valor texto. Dessa forma, ele é considerado como um **tipo simples**.

Repare que o marcador *CarPolicy* se repete nas linhas 3 e 13. Essa é a forma como representamos uma lista em XML.

Para os mais atentos, a estrutura XML se parece muito com um HTML. No link a seguir, o autor descreve as diferenças entre os dois: <https://www.w3.org/standards/xml/core>.

2.2 XML Schema

A especificação XML apenas define como os dados devem ser apresentados, mas não impõe nenhuma restrição sobre seu conteúdo, o nome dos marcadores, quais desses são obrigatórios e a quantidade de vezes que um marcador pode ser incluído no documento.

Todas essas outras funções são declaradas em outro documento, conhecido como XSD (XML *Schema Definition*) ou simplesmente XML *Scheme*. Existe ainda outro formato chamado DTD (*Document Type Definition*) que faz a mesma função, mas que vem sendo desencorajado em seu uso.

XSD é, portanto, um padrão para validação de documentos XML. Também é uma especificação da W3C e está na versão 1.1.

Segue um exemplo de arquivo contendo um XML Scheme:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
<xs:element name="addresses">
<xs:complexType>
<xs:sequence>
<xs:element ref="address" minOccurs='1' maxOccurs='unbounded' />
</xs:sequence>
</xs:complexType>
</xs:element>
```

```
  <xs:element name="address">
<xs:complexType>
<xs:sequence>
<xs:element ref="name" minOccurs='0' maxOccurs='1' />
<xs:element ref="street" minOccurs='0' maxOccurs='1' />
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="name" type='xs:string' />
<xs:element name="street" type='xs:string' />
</xs:schema>
```

Comando de prompt 2.2 – Exemplo de um XML Scheme de dados de endereços
Fonte: Elaborado pelo autor (2021)

Repare que um arquivo XML Scheme também é formatado como XML. No endereço a seguir, é possível fazer o download do XML Scheme "**do** XML Scheme": <https://www.w3.org/2009/XMLSchema/XMLSchema.xsd>. O documento começa com o marcador chamado **Schema**. Logo em seguida, ele define um **element** chamado *addresses* como sendo do tipo complexo.

Como visto há pouco, tipos complexos são aqueles que possuem uma estrutura de outros marcados em seu conteúdo. No caso de *addresses*, seu valor pode conter o marcador chamado *address*, podendo aparecer uma ou mais vezes em sequência (valor *unbounded* do atributo *maxOccurs*).

Na linha 10, há a definição do marcador address: um tipo complexo, podendo ter em seu valor os elementos *name* e *street* aparecendo em sequência. Ambos os elementos são opcionais (valor 0 do atributo *minOccurs*).

As linhas 18 e 19 contêm a definição dos elementos *name* e *street*: Ambos são elementos simples, ou seja, contêm um valor texto em vez de uma estrutura de dados.

O arquivo XML abaixo é um exemplo de um arquivo válido pelo XML Scheme que vimos há pouco:

```
<?xml version="1.0" encoding="utf-8"?>
<addresses>
  <address>
    <name>Joe Tester</name>
    <street>Baker street 5</street>
```

```
  </address>
</addresses>
```

Comando de prompt 2.3 – Exemplo de um XML válido a partir do XML Scheme de endereços
Fonte: Elaborado pelo autor (2021)

Para os atributos simples, XML Scheme já define uma lista de tipos de dados. Os mais utilizados são:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Embora neste exemplo não haja, é possível especificar o atributo de um marcador por meio de `xs:attribute`.

É possível aplicar uma série de restrições sobre os valores de um tipo simples, tais como: menor valor, maior valor, intervalo de dados, conjunto de valores (semelhante a um Enum), padrão de formato, número mínimo e máximo de caracteres, entre outros.

No exemplo acima, repare que os tipos complexos contêm a sua definição dentro do próprio valor do elemento. Existe a possibilidade de separar o elemento de sua definição. Segue um exemplo:

```
<xs:element name="employee" type="personinfo"/>
<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Comando de prompt 2.4 – Exemplo de uma definição de tipo fora da declaração do elemento
Fonte: https://www.w3schools.com/xml/schema_complex.asp

O marcador *sequence* é um exemplo de **indicadores**. Nele, todos os elementos definidos devem ser informados de forma sequencial.

Existem ainda os indicadores:

- **All**: Indica que os elementos devem ser informados pelo menos uma vez, mas em qualquer ordem.
- **Choice**: Indica que um ou outro elemento deve ser informado.

Dentro do elemento, podemos definir a quantidade mínima (**minOccurs**) e máxima (**maxOccurs**) que devem ser informadas. No exemplo acima, temos isso declarado para os elementos *address*, *name* e *street*. Para informar que um elemento pode ser informado um número ilimitado de vezes, utilize o valor **unbounded**.

A especificação ainda permite dezenas de outras possibilidades de declarações. Para conhecer mais sobre elas, recomendamos a leitura do seguinte material do W3C: <https://www.w3schools.com/xml/schema_intro.asp>.

2.3 XSLT

Outra especificação muito utilizada da W3C (que pertence ao mesmo grupo do XML e XSD) é o XSLT (eXtensible Stylesheet Language Transformations). Por meio dela, é possível fazer a transformação dos dados de um XML para outro formato, podendo ser XML ou outro formato qualquer.

Assim como as especificações anteriores, também é mantida pela W3C e atualmente está na versão 3.0.

XSLT faz parte de um grupo de outras especificações chamado XSL (eXtensible Stylesheet Language). Outras especificações que pertencem ao mesmo grupo:

- XPath: Linguagem para navegar em documentos XML.
- XQuery: Linguagem para consulta em documentos XML.

A capacidade de transformação de modelos de dados é importante quando queremos transformar um documento XML em outros formatos (como HTML, YAML ou JSON, por exemplo). Também é importante quando, dentro de uma composição de serviços, precisamos transformar um conjunto de dados de um formato XML para outro formato XML.

Para exemplificar: digamos que o usuário do seu E-Commerce informe o CEP para calcular o frete, mas o serviço que efetua o cálculo exige todos os dados de endereço. Assim, é necessário primeiro fazer uma chamada para um serviço contendo uma base de CEP (exemplo: dos Correios), informando o CEP e obtendo os dados do endereço.

Por se tratar de dois sistemas distintos, é muito provável que não seja o mesmo formato de dados. Assim, é necessário fazer a transformação dos dados obtidos na base de CEP para adequar ao esperado pelo serviço de cálculo de frete.

Digamos que esse serviço de busca de CEP retorne o seguinte documento:

```
<?xml version="1.0"?>
<query-results>
  <result id="1">
    <streetNo>1793</streetNo>
    <street>Avenida Paulista</street>
    <city>São Paulo</city>
    <state>SP</state>
    <zipcode>01311-200</zipcode>
  </result>
</query-results>
```

Comando de prompt 2.5 – Exemplo de um documento XML com dados de endereços

Fonte: Elaborado pelo autor (2021)

Agora, digamos que o serviço de cálculo de frete espera algo parecido com este exemplo:

```
<?xml version="1.0"?>
<calculo-frete>
  <frete tipo="simples">
    <rua>Avenida Paulista</rua>
    <numero>1793</numero>
    <cidade>São Paulo</cidade>
    <estado>SP</estado>
    <cep>01311-200</cep>
  </frete>
</calculo-frete>
```

Comando de prompt 2.6 – Exemplo de um documento XML com dados para cálculo de frete

Fonte: Elaborado pelo autor (2021)

Para fazermos a transformação da saída do serviço de busca de CEP em um documento válido pelo serviço de cálculo de frete, será necessário aplicar a seguinte transformação:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="3.0">
  <xsl:output omit-xml-declaration="yes" indent="yes"/>  <xsl:template
match="/">
    <calculo-frete>
      <xsl:for-each select="query-results/result">
        <frete tipo="simples">
          <rua><xsl:value-of select="street"/></rua>
<numero><xsl:value-of select="streetNo"/></numero>
<cidade><xsl:value-of select="city"/></cidade>
          <estado><xsl:value-of select="state"/></estado>          <cep>
<xsl:value-of select="zipcode"/></cep>
</frete>
</xsl:for-each>
    </calculo-frete>
  </xsl:template>
</xsl:stylesheet>
```

Comando de prompt 2.7 – Exemplo de XSLT para transformação dos dados de endereço para cálculo de frete

Fonte: Elaborado pelo autor (2021)

Primeira consideração: todo arquivo XSLT também é um arquivo XML. O XML Schema de um arquivo XSLT pode ser encontrado em: <https://www.w3.org/TR/2017/REC-xslt-30-20170608/>.

O documento começa com a marcação **stylesheet**. Isso define que se trata de um documento XSLT.

Logo em seguida, definimos algumas características de saída na marcação **xsl:output**. Isso auxilia os softwares que manipulam as transformações no modo como deve ser gerado o resultado. Nesse exemplo, foi parametrizado que não queremos o cabeçalho do arquivo XML (`omit-xml-declaration="yes"`) e que a saída deve ser indentada (`indent="yes"`).

Um documento XSLT é composto por um conjunto de marcações **template**. Elas definem a transformação de um sub-conjunto de documento, de acordo com o valor da propriedade **match**. Todo documento deve conter pelo menos uma marcação **template** com a propriedade `match="/"`. Trata-se do template que lida com a primeira marcação do documento XML, portanto é a partir dele que tudo começa.

Repare que temos uma marcação chamada `<calculo-frete>`. Ela será escrita literalmente no documento de saída.

Em nosso exemplo de entrada, há apenas uma marcação **result**. Mas trata-se de uma lista (podíamos ter mais de um endereço mapeado para um CEP). Por se tratar de uma coleção, é importante que a linguagem de transformação saiba iterar sobre os elementos dessa lista.

É o que fazemos ao definir a marcação `xsl:for-each`. Ela irá executar para cada vez que houver um item na lista de **query-results**. O valor da propriedade **select** espera uma expressão XPath.

Para que o valor de marcação seja impresso na saída do documento, utilizamos a marcação `xsl:value-of`. Da mesma forma, o valor da propriedade select espera um expressão XPath.

Quando temos mais um **template** em um documento, podemos invocá-lo de duas formas:

- Especificando exatamente qual **template** executar, informando o valor do atributo **name.** Para isso utilizamos a marcação `xsl:call-template`.
- Deixando que o software determine qual é o melhor **template** a ser executado, comparando o bloco que está sendo processado no momento com o valor da propriedade **match**. Para isso, utilizamos a marcação `xsl:apply-template`.

Ainda existem outras características não exploradas neste curso. Para saber mais, sugiro o seguinte site: <https://www.w3schools.com/xml/xsl_intro.asp >.

Uma funcionalidade adicionada nessa última versão é a capacidade de lidar com outros tipos de dados, como JSON. Recomendo a leitura do seguinte artigo para conhecer essa e outras características: <https://www.xml.com/articles/2017/02/14/why-you-should-be-using-xslt-30/>.

2.4 SOAP

Como a própria detentora da especificação, a W3C define SOAP como sendo “um protocolo leve projetado para troca de informações estruturadas em um ambiente descentralizado e distribuído”. Atualmente está na versão 1.2.

É um protocolo todo estruturado em XML e pode trafegar os dados utilizando outros protocolos, além do HTTP e HTTPS (embora estes sejam os mais utilizados). Por essa razão, o SOAP acaba redefinindo alguns conceitos que já estão presentes no HTTP (como *header* e *body*, por exemplo).

Os dados trafegados precisam ser “envelopados” para que sejam interpretados. Na prática, significa que os dados precisam estar no conteúdo de algumas marcações XML. Segue um exemplo de uma requisição e uma resposta SOAP:

```
<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Header>
    <Authentication xmlns="https://www.fazenda.sp.gov.br/codif/ws"
User="string" Password="string"/>
  </soap12:Header>
  <soap12:Body>
    <ValidarXmlSchema xmlns="https://www.fazenda.sp.gov.br/codif/ws">
      <Documento>string</Documento>    </ValidarXmlSchema>
  </soap12:Body>
</soap12:Envelope>
```

Comando de prompt 2.8 – Exemplo de uma requisição SOAP
Fonte: Extraído de
<https://www.fazenda.sp.gov.br/WSAlcoolCombustivel/ws/operacoes.asmx?op=ValidarXmlSchema>

```
<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <ValidarXmlSchemaResponse
xmlns="https://www.fazenda.sp.gov.br/codif/ws">
```

```
        <ValidarXmlSchemaResult>boolean</ValidarXmlSchemaResult>
    </ValidarXmlSchemaResponse>
  </soap12:Body>
</soap12:Envelope>
```

Comando de prompt 2.9 – Exemplo de uma resposta SOAP

Fonte: Extraído de <https://www.fazenda.sp.gov.br/WSAlcoolCombustivel/ws/operacoes.asmx?op=ValidarXmlSchema>

Repare que os dados que “realmente importam” ao consumidor do Web Service estão dentro de diversas outras marcações. Essas outras marcações auxiliam a plataforma nos aspectos de segurança e validação dos dados.

Como a forma mais comum de trafegar os dados é utilizando HTTP ou HTTPS, os serviços são identificados por uma URL. Mas, diferentemente do REST, não existe relação semântica entre os *paths* e o recurso que se deseja manipular. Isso fica a critério do desenvolvedor do Web Service.

Além disso, é comum você encontrar na URL a ação que se deseja executar, por exemplo: listarProdutos ou cadastrarCategoria. Diferentemente do REST, o único método HTTP utilizado pelo SOAP é POST.

A forma como o SOAP devolve ao cliente é um envelope do tipo FAULT como resposta (e não o Status Code, como o REST). Segue um exemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:m="http://www.example.org/timeouts">
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Sender</env:Value>
        <env:Subcode>
          <env:Value>m:MessageTimeout</env:Value>
        </env:Subcode>
      </env:Code>
      <env:Reason>
        <env:Text xml:lang="en">Sender Timeout</env:Text>
</env:Reason>
<env:Detail>
<m:MaxTime>P5M</m:MaxTime>
</env:Detail>
```

```
</env:Fault>
</env:Body></env:Envelope>
```

Comando de prompt 2.10 – Exemplo de uma mensagem de falha SOAP
Fonte: Extraído de <https://www.w3.org/TR/soap12-part1/#soapfault>

A seguir é apresentado um descritivo sobre cada um dos atributos do documento **Fault**:

- **Code**: uma identificação da falha. O elemento deve ter um dos seguintes valores:
  - DataEncodingUnknown
  - MustUnderstand
  - Receiver
  - Sender
  - VersionMismatch
- **Reason**: uma explicação do erro.
- **Node**: contém o URI do nó SOAP que gerou a falha.
- **Function**: informações opcionais sobre o objeto que causou a falha.
- **Details**: informações opcionais que variam com base na falha.

2.5 Extensões

SOAP possui diversas especificações chamadas de Extensões, cada uma lidando com aspectos auxiliares ao Web Service. A seguir, pode-se observar a relação das principais extensões.

- **WS-Security:** lida com criptografia e assinaturas digitais, permitindo criar aplicativos nos quais as mensagens não podem ser espionadas.
- **WS-Policy:** esta especificação se expande em WS-Security, permitindo especificar como e por quem um serviço pode ser usado
- **WS-I Basic Profile:** fornece um conjunto de padrões, práticas e taxonomias para que os desenvolvedores de softwares possam falar “uma mesma línguagem”.

- **WS-Transaction:** fornece uma forma para trabalhar os conceitos de transação e atomicidade de processos utilizando Web Services.
- **WS-BPEL:** fornece uma maneira de especificar interações entre serviços, como ramificação e processamento simultâneo.
- **WS-ReliableMessaging:** permite que você tenha certeza de que uma, e apenas uma , cópia da mensagem foi recebida e que (definitivamente) foi recebida.
- **WSRF (*Web Services Resource Framework*):** permite a criação de Web Services *Stateful* (que armazenam estado).
- **WSDM (*Web Services Distributed Management*):** discute a questão do gerenciamento e uso de serviços da web.

2.6 WSDL

Um grande aliado ao SOAP, é outra especificação conhecida como WSDL, utilizado para descrever formalmente um Web Service. Em outras palavras, define um **contrato** entre o consumidor e o fornecedor do serviço.

Acrônimo de Web Service Description Language, também é uma especificação da W3C e encontra-se na versão 2.0.

Em um WSDL especificamos:

- As operações fornecidas pelo Web Service.
- Parâmetros e tipo de dados das operações.
- Informações de vinculação e localização de publicação do Web Service.

Os desenvolvedores contam com diversas ferramentas que utilizam WSDL para gerar serviços, clientes e “esqueletos” de projetos para o Web Service. Além disso, as áreas de governança de TI utilizam WSDL para gestão de ativos de softwares.

Segue um exemplo de um arquivo WSDL:

```
<?xml version="1.0" encoding="utf-8"?>
<description xmlns="http://www.w3.org/ns/wsdl"
```

```
      xmlns:tns="http://yoursite.com/MyService"
      xmlns:stns="http://yoursite.com/MyService/schema"
      xmlns:wsoap="http://www.w3.org/ns/wsdl/soap"
      xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
      xmlns:wsdlx="http://www.w3.org/ns/wsdl-extensions"
      targetNamespace="http://yoursite.com/MyService">
  <documentation>This document describes my Service. You can find
additional information in the following web page:
http://yoursite.com/MyService/help.html</documentation>
  <types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
               xmlns="http://yoursite.com/MyService/schema"
               targetNamespace="http://yoursite.com/MyService/schema">
      <xs:element name="checkServiceStatus" type="tCheckServiceStatus"/>
        <xs:complexType name="tCheckServiceStatus">
          <xs:sequence>
            <xs:element name="checkDate" type="xs:date"/>
            <xs:element name="serviceName" type="xs:string"/>
</xs:sequence>
</xs:complexType>
<xs:element name=" checkServiceStatusResponse" type="xs:double"/>
<xs:element name="dataError" type="xs:string"/>
</xs:schema>
</types>
<interface name="myServiceInterface">
<fault name="dataFault" element="stns:dataError"/>
<operation name="checkServiceStatusOp"
              pattern="http://www.w3.org/ns/wsdl/in-out"
              style=" http://www.w3.org/ns/wsdl/style/iri"
              wsdlx:safe="true">
<input messageLabel="In" element="stns:checkServiceStatus"/>
<output messageLabel="Out"
             element="stns:checkServiceStatusResponse"/>
<outfault messageLabel="Out" ref="tns:dataFault"/>
</operation>
</interface>
<binding name="myServiceInterfaceSOAPBinding"
interface="tns:myServiceInterface"
type="http://www.w3.org/ns/wsdl/soap"
wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">
<operation ref="tns:checkServiceStatusOp"
```

```
          wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>
<fault ref="tns:dataFault" wsoap:code="soap:Sender"/>
</binding>
<service name="myService" interface="tns:myServiceInterface">
<endpoint name="myServiceEndpoint"
binding="tns:myServiceInterfaceSOAPBinding"
address="http://yoursite.com/MyService"/>
</service>
</description>
```

Comando de prompt 2.11 – Exemplo de uma WSDL
Fonte: Extraído de <http://www.wideskills.com/wsdl/full-wsdl-20-example>

Como você já deve estar acostumado, vamos descrever um Web Service utilizando XML. Mais detalhes sobre o XML Scheme podem ser consultados em <https://www.w3.org/2007/06/wsdl/wsdl20.xsd>.

Iniciamos com a marcação **description**, seguida da marcação **documentation**, sendo essa última apenas um texto descritivo sobre o serviço.

Logo após, temos a marcação **types**. Nela especificamos o modelo de dados que será utilizado para definição dos serviços (dados de entrada e serviço). Os mais atentos notaram que se utiliza XSD (XML Scheme) para definição do modelo de dados.

Pode se optar por embutir o XML Scheme dentro do arquivo WSDL ou referenciar um arquivo externo, por meio da marcação **import**.

Na seção seguinte, definimos dentro da marcação **interface** todas as operações (métodos) do Web Service. É possível um ou mais serviços. Para cada operação, é importante especificar: a entrada (marcação **input**), a saída (marcação **output**) e uma mensagem de falha (marcação **outfault**).

Na marcação **binding,** declaramos em qual protocolo nosso Web Service vai trabalhar.

Na última marcação, chamada **service**, definimos onde o serviço está disponível. É aqui, mais precisamente dentro da propriedade **address** da marcação **endpoint**, onde informamos a URL em que está disponível o serviço.

Conclusão

Neste capítulo, discutimos em profundidade o Web Service do tipo SOAP e todos os protocolos que estão relacionados.

Começamos pelo XML, um formato de dados que os armazena em marcações. Depois avançamos para XML Scheme, uma forma de definir um modelo de dados para XML, e seguimos para XSLT, uma forma de transformar dados XML em outros formatos.

Por fim, encerramos falando de utilização de envelopes para trafegar os dados, mensagens de falha, extensões SOAP e definindo contrato utilizando WSDL. No próximo capítulo serão abordadas as especificações JAX-WS e JAX-RS.

REFERÊNCIAS

CHASE, Nicholas. **Understanding web services specifications, Part 1 SOAP**. IBM. 2006. Disponível em: <https://www.ibm.com/developerworks/webservices /tutorials/ws-understand-web-services1/ws-understand-web-services1.html>. Acesso em: 03 fev. 2021.

W3C. **SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)**. 2007. Disponível em: < https://www.w3.org/TR/soap12-part1/>. Acesso em: 06 fev. 2021.

W3C. **Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language**. 2007. Disponível em: <https://www.w3.org/TR/wsdl/>. Acesso em: 06 fev. 2021.

W3C. **XSL Transformations (XSLT) Version 3.0**. 2017. Disponível em: <https://www.w3.org/TR/2017/REC-xslt-30-20170608/>. Acesso em: 06 fev. 2021.

GLOSSÁRIO

SOAP	Simple Object Access Protocol
XSD	Acrônimo de XML Scheme
WSDL	Acrônimo de Web Service Description Language
XSLT	Acrônimo de eXtensible Stylesheet Language Transformations