**Programando Web Services com SOAP**

**Apêndice B. Noções Básicas de XML Schema**

A especificação do XML Schema é longa e complexa. Para criar XML de SOAP e WSDL, você deve saber como o XML Schema especifica tipos de dados. Este apêndice é uma introdução rápida ao tópico, com exemplos. Você não sairá daqui como um especialista em Schema; mas será capaz de acompanhar WSDL.

**B.1 Tipos Simples e Complexos**

Em um XML Schema, todos os tipos de dados são primitivos ou derivados. Um tipo de dado primitivo é aquele que não pode ser expresso em termos de qualquer outro tipo de dado. A especificação do XML Schema dá o exemplo de um *float*, “um conceito matemático bem definido que não pode ser definido em termos de outros tipos de dados”, enquanto um *integer* é um derivado do tipo de dado *decimal*. Nesse caso, um *float* é primitivo e um *integer* é derivado.

Todos os tipos de dados primitivos são atômicos. Ou seja, o valor do tipo de dado não pode ser decomposto além do que já é. Por exemplo, o número 1 é um valor atômico. Tipos de dados derivados podem ou não ser atômicos. Por exemplo, um *integer*, como já vimos, é um tipo de dado derivado que possui um valor atômico. Um número de telefone, entretanto, também é um tipo de dado derivado cujo valor não é atômico; ele é, na verdade, uma coleção de três valores atômicos individuais.

Os tipos de dados são principalmente derivados por **restrição** ou **extensão** (existem outras maneiras, mas estas são as mais comuns). Na derivação por restrição, o valor do tipo de dado é restringido de alguma forma. Por exemplo, um *integer* é uma derivação do tipo de dado *decimal* que permite um intervalo mais estreito de valores do que um *decimal*; um *integer*, em outras palavras, é permitido conter um subconjunto restrito de valores *decimal*. Derivação por extensão significa que várias restrições do tipo de dado base estão sendo removidas para permitir valores adicionais que, de outra forma, não seriam permitidos. Por exemplo, um tipo de dado número de telefone pode ser estendido para incluir um campo de código de país.

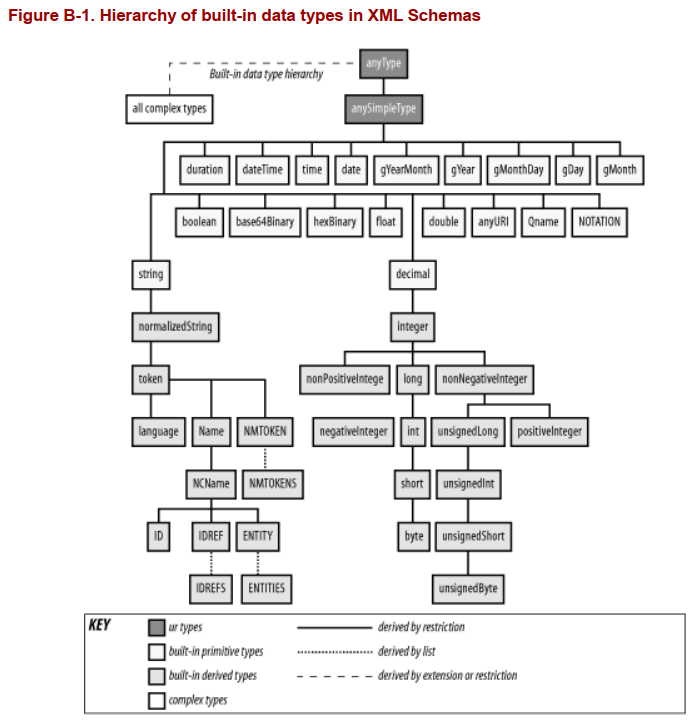
Isso é algo análogo a classes e objetos Java. Todas as classes Java são tipos de objetos Java. Todos os objetos Java são do tipo java.lang.Object. Quando crio uma nova classe Java derivada de java.lang.Object, na maioria das vezes estou adicionando novas funcionalidades (uma nova operação, uma nova propriedade, etc.). Isso é derivação por extensão. Quando sobrescrevo uma operação existente (como a operação toString()), estou derivando por restrição. Essa analogia obviamente não suporta um exame mais profundo, mas pode ser útil mesmo assim.

Os autores da especificação do XML Schema perceberam que, embora tivessem um mecanismo de tipagem de dados simples e extensível, ainda precisavam definir um pequeno conjunto de tipos de dados incorporados que refletissem cenários de uso comuns. Assim, os desenvolvedores de aplicativos não precisariam ficar reinventando os mesmos tipos de dados comuns repetidamente, o que só resultaria na mesma confusão que interfere na interoperabilidade entre plataformas de programação. Assim nasceram os tipos de dados incorporados do XML Schema e agora temos coisas como *string*, *integer*, *float*, *boolean*, *URI* e *time* finalmente definidos de uma forma comum que todas as plataformas de aplicativos são capazes de entender.

Esses tipos de dados formam uma hierarquia que pode ser rastreada até um único tipo de dado atômico primitivo chamado **anyType**. Todos os outros tipos de dados usados em XML Schemas derivam desse único tipo primitivo.

Existem dois tipos de dados que podem ser derivados de *anyType*: **tipos simples** e **tipos complexos**. Tipos simples representam todos os tipos de dados atômicos derivados incorporados no XML Schema. Isso inclui coisas como *string*, *integer* e *boolean*. Tipos complexos representam todos os tipos de dados derivados não atômicos — o número de telefone, por exemplo.

A Figura B-1, adaptada da usada na especificação de tipos de dados do XML Schema, ilustra a hierarquia dos tipos de dados incorporados.



Observe que todo “tipo simples” incorporado não deriva diretamente de *anyType*, mas do tipo de dado *anySimpleType*, que por sua vez é derivado de *anyType*. Como regra, a especificação do XML Schema dita que qualquer derivado de *anySimpleType* não pode ser derivado por extensão. Basicamente, isso significa que o elemento não pode conter nenhum atributo ou elemento filho, em termos de expressão do tipo de dado como XML. Novamente, se isso não estiver fazendo muito sentido agora, fará logo ao olharmos alguns exemplos simples.

Uma rápida revisão: introduzimos o fato de que existem essencialmente dois tipos de dados definidos por um XML Schema. Estes incluem **tipos simples**, que são atômicos — tipos de dados de valor único que podem ou não ser derivados por restrição de outros tipos simples. O outro tipo de dado definido por um XML Schema é o **tipo complexo**, que é composto de coleções de tipos simples e deve ser derivado de outros tipos complexos ou de tipos simples.

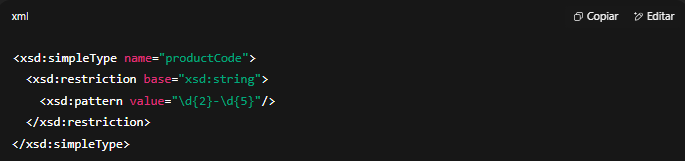
**B.2 Alguns Exemplos**

Embora o mecanismo de tipagem de dados do XML Schema seja, na verdade, bastante fácil de usar, descobrimos que muitas vezes é algo difícil de explicar. Vamos passar por alguns exemplos simples para esclarecer as coisas.

**B.2.1 Tipos Simples**

Vamos praticar definindo um tipo de dado simples. Suponha que temos um tipo de dado productCode. Esse código de produto deve começar com dois números, seguidos por um hífen e mais cinco números. O **Exemplo B-1** ilustra como expressar esse tipo de dado dentro de um XML Schema.

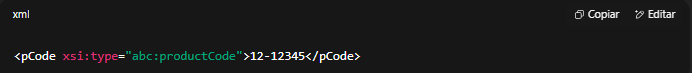
**Exemplo B-1. productCode**



Aqui, vemos que productCode é um derivado do tipo de dado incorporado string do XML Schema que foi restringido para permitir apenas valores que correspondam à expressão regular \d{2}-\d{5}.

Se fôssemos expressar uma instância desse tipo de dado em XML, ficaria algo como o **Exemplo B-2**.

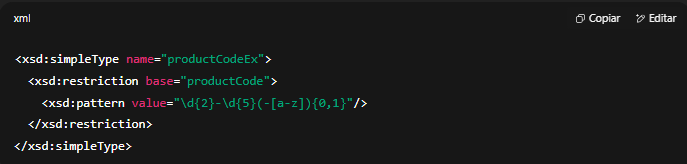
**Exemplo B-2. Instância de productCode**



Neste exemplo simples, demonstramos várias coisas: o productCode é um tipo simples derivado com um valor atômico, e derivamos o productCode restringindo o conjunto possível de valores que seu tipo de dado base (nesse caso, string) pode conter.

Agora vamos criar um código de produto estendido que possa ou não ter um adicional -[a-z] (um hífen seguido por qualquer letra minúscula). Podemos fazer isso derivando um novo tipo simples productCodeEx e alterando o padrão para \d{2}-\d{5}(-[a-z]){0,1}, como no **Exemplo B-3**.

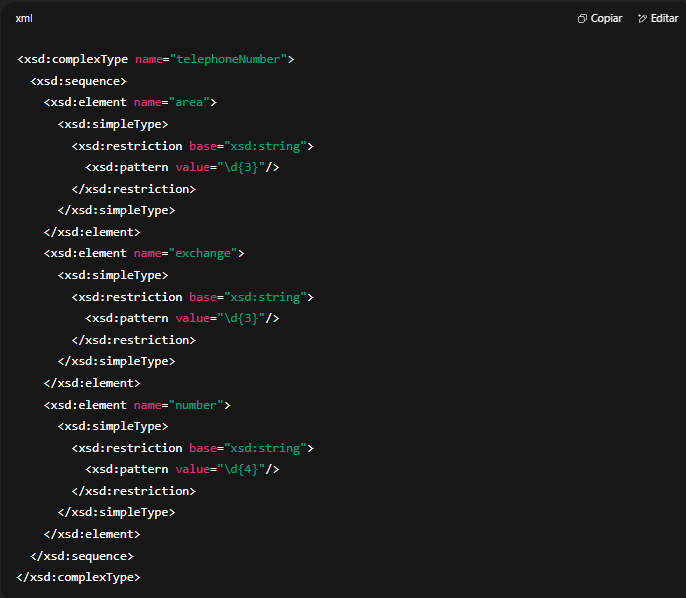
**Exemplo B-3. productCode estendido**



**B.2.2 Tipos Complexos**

Agora que provavelmente você já entendeu os tipos simples, vamos ver os tipos complexos. Um tipo complexo é qualquer tipo de dado que contém uma coleção de outros tipos de dados primitivos. Um número de telefone é um exemplo: ele contém três partes distintas de informação. Um tipo complexo número de telefone em XML Schema se parece com o **Exemplo B-4**.

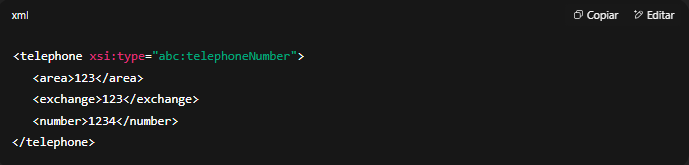
**Exemplo B-4. Tipo telephoneNumber**



O tipo de dado telephoneNumber consiste em uma sequência de três elementos de dados, cada um dos quais é um derivado restrito do tipo de dado string do XML Schema.

Uma instância desse tipo de dado ficaria como no **Exemplo B-5**.

**Exemplo B-5. Instância de telephoneNumber**



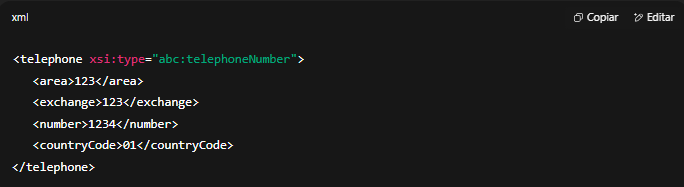
Se eu quisesse criar uma versão estendida desse tipo de dado que incluísse um código de país, eu faria isso criando um novo complexType derivado por extensão. Isso é mostrado no **Exemplo B-6**.

**Exemplo B-6. Estendendo telephoneNumber para incluir um código de país**



Uma instância do número de telefone estendido ficaria como no **Exemplo B-7**.

**Exemplo B-7. Instância de telephoneNumber estendido**



Observe que o elemento countryCode está no final da sequência de elementos de dados. Isso se deve à forma como o XML Schema impõe a ordem dos elementos dentro dos tipos de dados. Como estamos derivando por extensão, todos os novos elementos definidos no tipo de dado telephoneNumberEx devem aparecer depois dos elementos definidos no seu tipo base telephoneNumber.

Se quiséssemos que countryCode aparecesse primeiro na sequência, teríamos que derivar por restrição e redefinir cada um dos elementos de dados, como no **Exemplo B-8**.

**Exemplo B-8. telephoneNumber restrito**

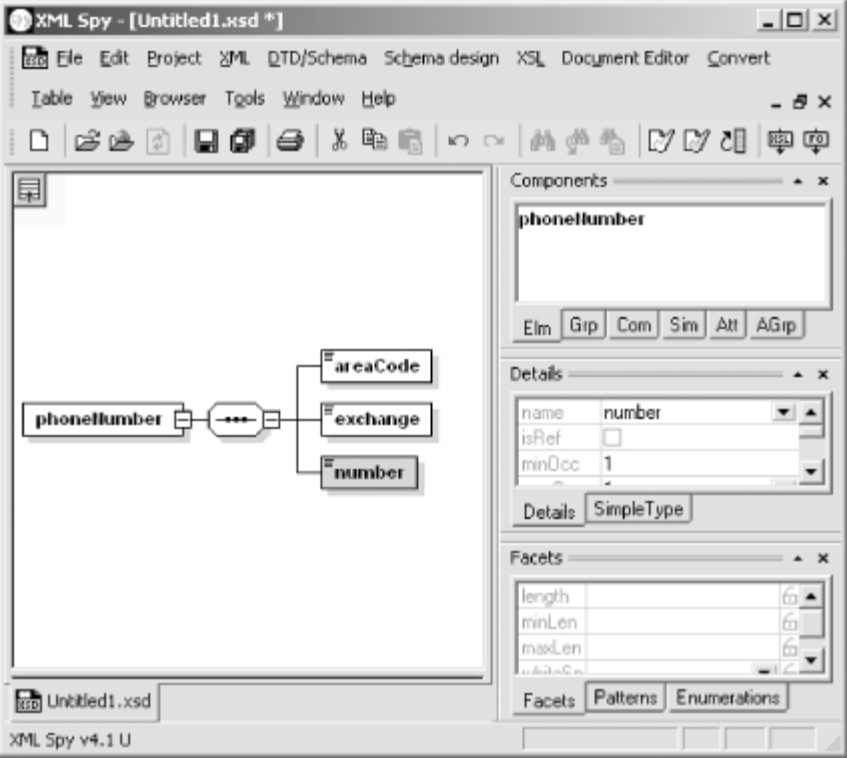


E isso é o básico para definir tipos de dados com XML Schemas. Há muitos detalhes que estamos deixando de lado. Vale a pena dedicar um tempo para aprender mais sobre XML Schemas.

**B.3 XML Spy**

O **XML Spy** é talvez o melhor produto disponível para trabalhar com XML Schemas. Seu ambiente de desenvolvimento XML permite projetar esquemas XML visualmente de forma rápida e fácil, ocultando a complexidade sintática que normalmente atrapalha as pessoas. A **Figura B-2** mostra uma captura de tela do editor visual de esquemas do XML Spy.

**Figura B-2.** Uma visão do editor visual de esquemas do XML Spy



O XML Spy é um produto comercial disponível em: [**http://www.xmlspy.com/**](http://www.xmlspy.com/). Embora não seja barato (algumas centenas de dólares americanos na época em que este texto foi escrito), ele vale bem o preço para desenvolvedores sérios.

**Sumário do Apêndice B:**

* **B. XML Schema Basics**
* **B.1 Simple and Complex Types**
* **B.2 Some Examples**
* **B.3 XML Spy**