



CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS

COMUNIDADE EVANGÉLICA LUTERANA "SÃO PAULO"
Credenciado pelo Decreto de 06/07/2000 - D.O.U. nº 130 de 07/07/2000

MICHAEL SCHUENCK DOS SANTOS

**Utilização de *Web Services* na plataforma .NET para
a criação de um aplicativo visualizador de notícias para
dispositivos móveis**

Palmas
2003

SUMÁRIO

SUMÁRIO	VI
1 INTRODUÇÃO	13
2 REVISÃO DE LITERATURA	15
2.1 A TECNOLOGIA DE <i>WEB SERVICES</i>	15
2.2 XML	17
2.3 <i>NAMESPACES</i>	19
2.4 XML SCHEMA.....	21
2.4.1 <i>Elementos</i>	22
2.4.2 <i>Atributos</i>	23
2.4.3 <i>Tipos de dados</i>	24
2.5 SOAP	24
2.5.1 <i>Envelope SOAP</i>	26
2.5.1.1 <i>SOAP Fault</i>	28
2.5.2 <i>Regras de codificação</i>	29
2.5.3 <i>Serialização XML</i>	30
2.5.4 <i>SOAP RPC</i>	31
2.5.5 <i>Ligação entre HTTP e SOAP</i>	32
2.6 HTTP-GET E HTTP-POST	34
2.7 WSDL.....	36
2.7.1 <i>Elemento types</i>	37
2.7.2 <i>Elementos message</i>	38
2.7.3 <i>Elementos portType</i>	39
2.7.4 <i>Elementos binding</i>	39
2.7.5 <i>Elemento service</i>	40
2.8 UDDI	41
2.9 CRIAÇÃO DE <i>WEB SERVICES</i> DA PLATAFORMA .NET	42
2.9.1 <i>Arquivos ASMX</i>	43
2.9.2 <i>Autenticação de Web Services</i>	47
2.10 CONSUMO DE <i>WEB SERVICES</i> NA PLATAFORMA .NET	49
2.10.1 <i>Geração da classe proxy</i>	49
2.10.2 <i>Compilação da classe proxy</i>	50
2.11 DISPOSITIVOS MÓVEIS BASEADOS NA PLATAFORMA POCKET PC	51
3 MATERIAL E MÉTODOS	53
3.1 LOCAL E PERÍODO	53
3.2 MATERIAL	53
3.3 METODOLOGIA	54
4 RESULTADOS E DISCUSSÃO	55
4.1 BANCO DE DADOS DE NOTÍCIAS	56
4.2 CLASSE PARA ACESSO AO BANCO DE DADOS.....	57
4.3 CLASSES PARA REPRESENTAÇÃO DAS NOTÍCIAS.....	58
4.4 CLASSE PARA REPRESENTAÇÃO DO <i>WEB SERVICE</i>	59
4.5 A APLICAÇÃO PARA DISPOSITIVOS MÓVEIS	62

5 CONSIDERAÇÕES FINAIS.....	67
REFERÊNCIAS BIBLIOGRÁFICAS	69

LISTA DE FIGURAS

<i>Figura 1: Aplicação na Internet acessando diferentes Web Services.....</i>	16
<i>Figura 2: Documento XML.....</i>	18
<i>Figura 3: Documento XML utilizando Namespace</i>	20
<i>Figura 4: Documento XML utilizando Namespace com prefixos.....</i>	21
<i>Figura 5: Elementos simples em XML Schema.....</i>	22
<i>Figura 6: Elemento complexo em XML Schema.....</i>	23
<i>Figura 7: Atributo em XML Schema.....</i>	24
<i>Figura 9: Etapas na comunicação entre servidores.....</i>	25
<i>Figura 10: Representação de um envelope SOAP (MSDN, 2003B).....</i>	26
<i>Figura 11: Envelope SOAP (BALLINGER, 2003).....</i>	27
<i>Figura 12: Envelope SOAP reportando um erro (SOAP, 2003)</i>	29
<i>Figura 13: Classe a ser codificada (BALLINGER, 2003)</i>	30
<i>Figura 14: Resultado da codificação SOAP (BALLINGER, 2003)</i>	30
<i>Figura 15: Chamada remota a um método com SOAP (BALLINGER, 2003).....</i>	32
<i>Figura 16: Requisição SOAP utilizando HTTP</i>	33
<i>Figura 17: Resposta SOAP utilizando HTTP</i>	34
<i>Figura 18: Requisição HTTP-GET</i>	34
<i>Figura 19: Resposta HTTP-GET</i>	35
<i>Figura 20: Requisição HTTP-POST.....</i>	35
<i>Figura 18: Elementos message em um documento WSDL</i>	36
<i>Figura 19: Elemento types em um documento WSDL.....</i>	37
<i>Figura 20: Elementos message em um documento WSDL</i>	38
<i>Figura 21: Elementos portType em um documento WSDL</i>	39
<i>Figura 22: Elementos binding em um documento WSDL.....</i>	40
<i>Figura 23: Elementos service em um documento WSDL</i>	41
<i>Figura 24: Exemplo de arquivo ASMX.....</i>	44
<i>Figura 25: Visualização de um arquivo ASMX em um browser.....</i>	45
<i>Figura 26: Interface para inserção dos parâmetros do método somar</i>	46
<i>Figura 27: Resultado da execução do método somar</i>	47
<i>Figura 28: Classe para representação de cabeçalho de SOAP</i>	48
<i>Figura 29: Serviço autenticado através de cabeçalho de SOAP.....</i>	48

Figura 30: Classe proxy para o Web Service definido na figura 24 **Erro! Indicador não definido.**

Figura 31: Aplicação C# invocando o método somar do Web Service da figura 24 50

Figura 32: Tela principal do emulador de Pocket PC 52

Figura 33: Modelo da arquitetura desenvolvida..... 55

Figura 34: Esquema do banco de dados para armazenamento das notícias 56

Figura 35: Classe Banco, usada para acesso ao banco de dados..... 57

Figura 36: Classe Noticia, usada para representar notícias..... 59

Figura 37: WebMethod NoticiasDaInstituicao 60

Figura 38: Resultado da execução do método NoticiasDaInstituicao 61

Figura 39: Tela do emulador exibindo os títulos das notícias dos cursos 63

Figura 40: Método preencherLV, da classe FormConsumer..... 64

Figura 41: Tela do emulador exibindo os detalhes de uma notícia 65

Figura 42: Método setNoticia, da classe FormDetalhes..... 66

LISTA DE ABREVIATURAS

ASMX = *Active Server Method Extension*

DTD = *Document Type Definition*

HTTP-GET = *HyperText Transfer Protocol - Get*

HTTP-POST = *HyperText Transfer Protocol - Post*

IIS = *Internet Information Services*

IRI = *Internationalized Resource Identifiers*

PDA = *Personal Digital Assistant*

SOAP = *Simple Object Access Protocol*

UDDI = *Universal Description, Discovery and Integration*

URI = *Uniform Resource Identifier*

URL = *Uniform Resource Locators*

W3C = *World Wide Web Consortium*

WAP = *Wireless Application Protocol*

WSA = *Web Services Achitecture*

WSDL = *Web Services Description Language*

XML = *eXtensible Markup Language*

XML-RPC = *XML Remote Procedure Call*

RESUMO

A tecnologia de *Web Services* vêm se destacando como uma boa opção para comunicações remotas. Isto se deve ao fato desta tecnologia utilizar XML, o que permite que aplicações de diferentes plataformas se comuniquem. Da mesma forma, a utilização de dispositivos computacionais móveis tem se tornado cada vez mais popular. Assim, este trabalho apresenta um modelo desenvolvido para a utilização destas duas tecnologias, utilizando-se a plataforma .NET, concretizada em uma aplicação para visualização de notícias em dispositivos móveis.

Palavras-chaves: XML, *Web Services*, Dispositivos Móveis

ABSTRACT

The Web Services technology has being emerged as a good option to remote communications. This is because this techonology allows the use of remote methods, obtaining well portability, that is possible by the use of XML. In the same way, the use of mobile devices, have becoming very used. Therefore, this work presents a model developed to use these two technologies, using the .NET framework, materialized in a application to visualization of news on mobile devices.

Keywords: XML, *Web Services*, Mobile Devices

1 INTRODUÇÃO

A tecnologia de *Web Services* oferece recursos para a utilização, através da Internet, de classes e métodos remotos. Seu objetivo é permitir que aplicações de diferentes plataformas se comuniquem. Para isto, é utilizada a XML, que é o formato de dados utilizado para o transporte das informações trocadas, e que é caracterizado pela interoperabilidade. Assim, esforços têm sido dispendidos para que um *Web Service* desenvolvido na plataforma .NET possa ser acessado por uma aplicação desenvolvida na plataforma Java, e vice-versa, por exemplo.

O código dos *Web Services* da plataforma .NET, que foi utilizada no contexto deste trabalho, é semelhante ao código de classes normais, em linguagem de programação, alterando-se apenas algumas diretivas.

Para que as comunicações sejam possíveis, é necessário um conhecimento por parte da aplicação final, de quais são as funcionalidades oferecidas pelo *Web Service*, o que é feito através de documentos WSDL. Para a comunicação efetiva, podem ser utilizados alguns protocolos: HTTP-POST, HTTP-GET e SOAP, que permite a representação de dados complexos. Tanto estes protocolos quanto os documentos WSDL são baseados em XML, sendo este aspecto que possibilita a interoperabilidade, pois através destes artefatos é que as mensagens são transportadas.

Outra tecnologia utilizada neste trabalho são os dispositivos computacionais móveis, denominados PDA's, que representam uma forma de comunicação e tratamento de dados, caracterizados pela mobilidade, e que têm se tornado bastante populares.

Assim, o objetivo principal deste trabalho é apresentar um modelo para utilização conjunta destas duas tecnologias emergentes, exemplificada através de um aplicativo para rodar em PDA's que utilizem a plataforma operacional *Pocket PC*. A função do aplicativo

desenvolvido é apresentar notícias localizadas em um banco de dados remoto, que são acessadas por meio dos métodos do *Web Service* desenvolvido.

Este trabalho está organizado da seguinte forma: o capítulo 2 apresenta os aspectos teóricos agregados pela tecnologia de *Web Services* e pelos dispositivos móveis; o capítulo 3 apresenta os materiais e métodos utilizados para a realização do trabalho; em seguida, o capítulo 4 apresenta os detalhes do banco de dados de notícias, bem como da implementação do *Web Service* e da aplicação para dispositivos móveis; o capítulo 5 apresenta as considerações finais sobre todo o trabalho; e por fim, são apresentadas as referências bibliográficas utilizadas para a execução deste trabalho.

2 REVISÃO DE LITERATURA

Como parte integrante deste trabalho, foi realizada uma revisão bibliográfica sobre a tecnologia de *Web Services*, além dos recursos disponibilizados pela plataforma .NET para a sua criação e utilização. Foram analisadas, ainda, as particularidades de se construir aplicações para dispositivos móveis, utilizando-se a plataforma .NET. Estes itens serão vistos com mais detalhes através das próximas seções.

2.1 A tecnologia de *Web Services*

A plataforma .NET oferece uma vasta lista de funcionalidades para a realização de tarefas diversas, porém, centradas na orientação a objetos e em uma grande integração com a Internet. Dentre os recursos oferecidos, incluem-se os *Web Services* (Serviços da *Web*), que também são oferecidos por outras plataformas, como Java, por exemplo.

A tecnologia de *Web Services* permite que aplicações também se comuniquem. Ou seja, o conteúdo pode ser transmitido através da Internet, por um *Web Service*, e recebido por uma aplicação remota, pela qual os dados podem ser utilizados de variadas maneiras: processados e exibidos em uma página da Internet, em uma aplicação *desktop*, ou em um dispositivo móvel (utilizado na implementação deste trabalho para acessar dados de um *Web Service*).

Neste sentido, pode-se fazer uma analogia à orientação a objetos tradicional, sendo que os *Web Services* são utilizados da mesma maneira que classes, no cliente, com a diferença que, em se tratando de *Web Services*, os métodos podem ser acessados remotamente (AMUNDSEN, 2002). Isto permite que uma aplicação utilize dados e processamento de diferentes lugares na Internet simultaneamente. Esta situação é

exemplificada pela figura 1, em que uma aplicação que utiliza a Internet, localizada em Porto Alegre, possui módulos cujos dados, ou mesmo somente a lógica (que é o caso da utilização do *Web Service* localizado em Palmas), são acessados através dos métodos e propriedades disponibilizados por *Web Services* localizados em servidores de Brasília, Rio de Janeiro e Palmas.

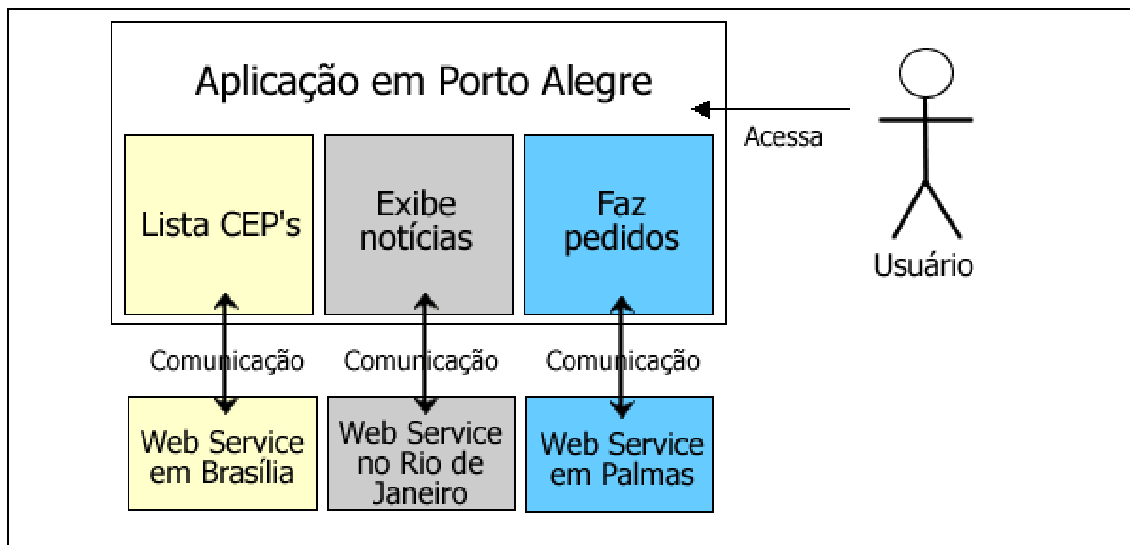


Figura 1: Aplicação na Internet acessando diferentes *Web Services*

Outra característica desta tecnologia é o fato de os dados trocados serem baseados no protocolo SOAP, que é baseado em XML. Sendo SOAP um protocolo aberto, é possível que aplicações de diferentes plataformas se comuniquem (AMUNDSEN, 2002). Neste sentido, o W3C tem trabalhado, no desenvolvimento de uma definição padrão, a WSA, que objetiva fazer com que as implementações para a tecnologia de *Web Services* de diferentes empresas sigam princípios básicos para que a interoperabilidade seja mantida (WSA, 2003).

Uma definição para o termo “Serviços da *Web*”, no contexto das características já apresentadas e de outras que serão ainda apresentadas, é a seguinte:

“Um *Web Service* é um sistema de software projetado para suportar interação máquina-a-máquina sobre uma rede. Ele tem uma interface descrita em um formato processável por máquina (especificamente WSDL). Outros sistemas interagem com o *Web Service* em uma maneira aconselhada por esta descrição usando mensagens SOAP, tipicamente transportadas usando HTTP com uma

serialização XML em conjunção com outros padrões relacionados à *Web*.”
(WSA, 2003)

Em outras palavras, esta definição informa que *Web Services* são artefatos de software acessados por aplicações remotas. Para isto, as mensagens trocadas são formatadas no protocolo SOAP, o que garante a interoperabilidade entre diferentes plataformas, em um processo denominado serialização XML. Porém, antes que as mensagens SOAP sejam trocadas, suas características são explicitadas através de documentos WSDL, que descrevem quais dados estarão sendo trocados, e como estes dados estarão organizados nas mensagens SOAP.

Como as mensagens SOAP e WSDL são, na verdade, documentos XML, elas são estruturadas por esquemas descritos através de *XML Schemas*, que, por sua vez, baseiam-se na utilização de *Namespaces*.

Adicionalmente, os *Web Services* podem ser tornados públicos através de um mecanismo denominado por UDDI. Este é um formato para armazenamento de referências para *Web Services* em repositórios acessíveis pela Internet. Assim, se um desenvolvedor precisa de um *Web Service* para realização de uma determinada tarefa, pode encontrar o *Web Service* que mais se adequa à sua necessidade.

Normalmente, o protocolo de rede utilizado no transporte destas mensagens é HTTP.

Após ter sido apresentado o modelo da interação entre as tecnologias utilizadas neste trabalho, as seções seguintes apresentarão com mais detalhes cada uma destas tecnologias.

2.2 XML

A XML (XML, 2003) é um padrão do W3C para representação de dados. Possui um formato simples e muito útil para o intercâmbio de dados, o que é conseguido através da característica de marcação da linguagem.

Uma característica marcante na XML é a possibilidade de definição de novas linguagens de marcação. Isto é possível pelo fato de as *tags* de marcação poderem ser

definidas pelo usuário. Por isto, pode-se dizer que dados semi-estruturados são bem representados pela XML.

A XML tem duas formas primárias de composição de informações: os elementos e os atributos. Os elementos são estruturas que permitem a atribuição de dados simples, tais como seqüências de números e letras (elementos simples) e de outros elementos (elementos complexos). Seus nomes são definidos entre os sinais “<” (menor) e “>” (maior), e a este conjunto se dá o nome de *tag* ou nó. Cada *tag* que é aberta deve ser fechada, o que é feito com a repetição da *tag* de abertura, porém, utilizando-se “/” (barra) antes do nome do elemento. Existe ainda, um tipo de *tags* que não contém mais dados que apenas seu nome e atributos. Neste caso, antes do sinal de maior, deve ser colocada a barra. O conteúdo do elemento é colocado entre as *tags* de abertura e a de fechamento. Os atributos são ainda mais simples: constituem-se de estruturas para atribuição de valores alfanuméricos e são colocados junto às *tags* de abertura dos elementos. A figura 2 exibe um documento XML.

1	<documentos>
2	<artigo id="a1">
3	<autor> Manoel de Sousa </autor>
4	<titulo> Dados Semi-Estruturados </titulo>
5	<instituicao> CEULP/ULBRA </instituicao>
6	</artigo>
7	<artigo id="a2">
8	<autor> Joana dos Santos </autor>
9	<titulo> Armazenamento de XML em SGBDOO's </titulo>
10	<instituicao> CEULP/ULBRA </instituicao>
11	<area> Bancos de Dados </area>
12	</artigo>
13	</documentos>

Figura 2: Documento XML

O documento apresentado é formado de elementos compostos, tais como documentos e artigo, que são compostos por outros elementos. Exemplos de elementos simples são autor e titulo, que contém dados propriamente ditos. Além destas representações de dados, este documento contém atributos, com o nome id, nos elementos artigo.

Os documentos XML podem ser validados contra estruturas determinadas, de forma que se possa ter vários documentos que sigam a mesma estruturação (ANDERSON, 2001). Estas estruturas podem ser definidas através de DTD's e XML Schemas. Em se

tratando de *Web Services*, vários dos componentes desta tecnologia são, na verdade, documentos XML. É o caso das mensagens SOAP e documentos WSDL, que possuem sua estrutura representada através de *XML Schemas* (que são também, documentos XML), que serão apresentados na próxima seção, juntamente com a tecnologia de *namespaces*.

2.3 Namespaces

Normalmente, um documento XML bem formado possui uma estrutura equivalente que o defina, ainda que esta não esteja explicitamente definida. Ou seja, quando se cria um documento XML sobre um determinado assunto, automaticamente, ele possui uma estruturação que pode ou não estar representada de alguma maneira (DTD ou *XML Schema*, principalmente). Isto sugere que cada documento deve ter seus dados validados por apenas uma estrutura.

Porém, um único documento XML pode tratar de vários domínios simultaneamente, fazendo com que a estrutura que representa o documento aborde todos os domínios suportados. Por outro lado, muitas vezes, representações dos domínios utilizados podem ter sido definidos anteriormente por outras pessoas. Assim, a reutilização destas estruturas seria uma boa saída para a construção de documentos XML com mais rapidez (sem a necessidade de se pensar na organização do documento novamente), permitindo a existência de estruturas padrão para a representação de determinados domínios em XML, além de, adicionalmente, permitir que sistemas computacionais interpretem, de forma adequada, os diferentes módulos do documento (NAMES, 2003). Desta forma, surge uma abordagem que permite que cada documento XML agregue dados que seguem estruturações diferentes.

Por outro lado, um documento com trechos que seguem estruturas diferentes pode acabar possuindo elementos e atributos com mesmo nome, mas significados diferentes (BALLINGER, 2003). Neste sentido, para permitir que documentos sejam representados por mais de uma estrutura simultaneamente, sem que ocorram conflitos entre os nomes, surgiu a tecnologia de *Namespaces* (espaços de nomes). Um documento utilizando esta tecnologia é exemplificado na figura 3.

1	<biblioteca>
2	<publicacao xmlns="http://schuenck/art" id="A1">
3	<autor xmlns="http://schuenck/art">Marcus Barbosa</autor>
4	<titulo xmlns="http://schuenck/art">Esquemas para XML</titulo>
5	<ano xmlns="http://schuenck/art">2003</ano>
6	<local xmlns="http://schuenck/art">CEULP/ULBRA</local>
7	<evento xmlns="http://schuenck/art">ENCOINFO V</evento>
8	</publicacao>
9	<publicacao xmlns="http://schuenck/liv" id="L1">
10	<autor xmlns="http://schuenck/liv">Keith Ballinger</autor>
11	<titulo xmlns="http://schuenck/liv">.NET Web Services: Architecture and Implementation</titulo>
12	<ano xmlns="http://schuenck/liv">2003</ano>
13	<editora xmlns="http://schuenck/liv">Addison-Wesley</editora>
14	<local xmlns="http://schuenck/liv">Boston</local>
15	</publicacao>
16	</biblioteca>

Figura 3: Documento XML utilizando *Namespace*

Neste documento, existe um artigo e um livro, cada um com seus respectivos dados. O valor do atributo `xmlns` é denominado URI, que será mais claramente apresentado adiante. Cada um dos elementos filhos do elemento `publicacao` que se inicia na linha 2, incluindo ele próprio, possui sua URI igual a “`http://schuenck/art`”, e por isto, segue a estrutura indicada pela *namespace* identificada por este nome. No caso do elemento `publicacao` com início na linha 9, seus elementos estão de acordo com a *namespace* identificada por “`http://schuenck/liv`”. No caso de atributos, eles seguem o que está definido na *namespace* do seu respectivo elemento.

Nota-se, no exemplo apresentado, que as duas ocorrências do elemento `local` possuem conotações diferentes. Na linha 6, “`local`” refere-se à instituição onde foi realizado o evento no qual o artigo foi publicado, enquanto que, na linha 14, “`local`” refere-se à cidade onde está situada a editora que publicou o livro.

Em se tratando de XML, as *namespaces* são identificadas por IRI’s, que são, na verdade, aprimoramentos de URI’s a fim de suportar um conjunto maior de caracteres (NAMES, 2003). Assim, URI’s e IRI’s são nomes que identificam unicamente um recurso. Usualmente, eles têm o formato de URL’s, mas isto não é obrigatório (BALLINGER, 2003).

É possível utilizar, também, prefixos que simplificam o uso das *namespaces*, sendo definidos no elemento raiz. Um exemplo deste tipo de construção é apresentado na figura 4.

1	<biblioteca xmlns:art="http://schuenck/art" xmlns:liv="http://schuenck/liv"
2	xmlns="http://schuenck/bib">
3	<art:publicacao id="A1">
4	<art:autor>Marcus Barbosa</art:autor>
5	<art:titulo>Esquemas para XML</art:titulo>
6	<art:ano>2003</art:ano>
7	<art:local>CEULP/ULBRA</art:local>
8	<art:evento>ENCOINFO V</art:evento>
9	</art:publicacao>
10	<liv:publicacao id="L1">
11	<liv:autor>Keith Ballinger</liv:autor>
12	<liv:titulo>.NET Web Services: Architecture and
13	Implementation</liv:titulo>
14	<liv:ano>2003</liv:ano>
15	<liv:editora>Addison-Wesley</liv:editora>
16	<liv:local>Boston</liv:local>
17	</liv:publicacao>
18	</biblioteca>

Figura 4: Documento XML utilizando *Namespace* com prefixos

Na primeira linha é informado que o prefixo *art* refere-se à *namespace* definida por “http://schuenck/art”, que *liv* refere-se a “http://schuenck/liv” e que a URI padrão é “http://schuenck/bib”. Isto significa que, se algum elemento não for identificado com um prefixo, será utilizada a *namespace* definida por “http://schuenck/bib”.

A criação das estruturas que definem *namespaces* é feita utilizando-se XML *Schema*, que será o mote da próxima seção.

2.4 XML Schema

XML Schema é uma tecnologia para representação de esquemas de dados XML recomendada pelo W3C em maio de 2001 (XSD, 2001A), representando uma alternativa importante à DTD, que foi proposta no início da utilização de XML, e que é bastante popular, sendo muito utilizada ainda. Suas principais características são as seguintes (PINTO, 2003):

- É um documento XML.
- Determina os elementos e atributos que podem existir nos documentos por eles validados, bem como sua ordem.

- Permite definição mais restrita de tipos de dados para elementos e atributos, em relação à DTD.
- Determina a cardinalidade dos elementos, controlando a quantidade exata de ocorrências de um elemento.
- Suporta a definição de tipos criados pelo usuário.
- Permite a reutilização de código (através da utilização de *namespaces*).
- Possui grande compatibilidade com bancos de dados relacionais.

Nos exemplos que serão apresentados nas próximas seções, será utilizado o prefixo `xsd`, que refere-se à *namespace* representada pela URI “`http://www.w3.org/2001/XMLSchema`”, sendo que este é o conjunto prefixo-URI normalmente utilizado em documentos XML *Schema*. A definição de elementos e atributos será apresentada nas subseções a seguir.

2.4.1 Elementos

Elementos simples são definidos pelo elemento `element`, que utiliza os atributos `name` e `type`. Além disso, pode-se definir restrições através do uso dos elementos `simpleType` (ao invés de `element`) e `restriction` (XSD, 2001A). A figura 5 apresenta um exemplo que define dois elementos simples: `nome` e `idade`, podendo esta última ser menor que 130.

1	<code><xsd:element name="nome" type="xsd:string"/></code>
2	<code><xsd:simpleType name="idade"></code>
3	<code> <xsd:restriction base="xsd:positiveInteger"></code>
4	<code> <xsd:maxExclusive value="130"/></code>
5	<code> </xsd:restriction></code>
6	<code></xsd:simpleType></code>

Figura 5: Elementos simples em XML *Schema*

Para a definição de elementos complexos utiliza-se o elemento `complexType`, que possui o elemento `name` para indicação do nome do elemento. Dentro deste elemento pode existir um elemento `sequence`, que possuirá elementos do tipo `element` (XSD, 2001A),

vistos anteriormente. A figura 6 apresenta um exemplo da definição de elementos complexos.

```

1 <xsd:complexType name="Noticia">
2   <xsd:sequence>
3     <xsd:element minOccurs="1" maxOccurs="1" name="id_noticia" type="xsd:int"/>
4     <xsd:element minOccurs="1" maxOccurs="4" name="titulo" type="xsd:string"/>
5   </xsd:sequence>
6 </xsd:complexType>

```

Figura 6: Elemento complexo em XML *Schema*

Nas linhas 3 e 4 estão sendo definidos os elementos simples, que possuem os atributos `minOccurs` e `maxOccurs`. Estes permitem controlar a quantidade de ocorrências dos elementos, sendo possível a definição de qualquer número como o mínimo ou o máximo de ocorrências, desde que o máximo seja maior ou igual ao mínimo. Para permitir que o máximo de ocorrências seja ilimitado, utiliza-se `maxOccurs="unbounded"`.

É permitido o uso de três tipos de delimitadores para definição da ordem dos elementos: `sequence`, visto anteriormente, que impõe que a ordem dos elementos no documento deve ser aquela em que a definição dos elementos aparece no documento XML *Schema*; `choice`, em que apenas um dos elementos definidos poderá existir no documento XML; e por fim, `all`, que diz que os elementos podem aparecer em qualquer ordem.

2.4.2 Atributos

Os atributos em XML *Schema* são definidos de forma semelhante aos elementos simples, porém, utilizando o elemento `attribute`, juntamente com os atributos `name` e `type`. É possível a utilização de outros atributos no XML *Schema*, para funções específicas: `use`, que especifica a obrigatoriedade ou não de um atributo; `fixed`, para atributos com valor fixo; e `default`, para a definição de um valor padrão. É permitido ainda, especificar mais restrições, através da utilização dos elementos `simpleType` e `restriction` (XSD, 2001A). Um exemplo é apresentado na figura 7.

```

1 <xsd:attribute name="idade">
2   <xsd:simpleType>
3     <xsd:restriction base="xsd:integer">

```

```
4     <xsd:minInclusive value="0"/>  
5     <xsd:maxInclusive value="130"/>  
6     <xsd:restriction/>  
7     </xsd:simpleType>  
8 </xsd:attribute>
```

Figura 7: Atributo em XML *Schema*

Considerando que o trecho apresentado na figura 7 está situado dentro de um `complexType`, ele tem a função de definir um atributo cujo nome é idade, que deve ser inteiro, cujo valor deve variar entre zero e cento e trinta.

2.4.3 Tipos de dados

Como visto nos exemplos anteriores, *XML Schema* define uma grande restrição aos tipos de dados dos elementos e atributos. Os tipos de dados se dividem principalmente em tipos complexos (definidos pelo usuário, como elementos complexos) e tipos simples. Entre os tipos simples, alguns exemplos são: `time`, `date`, `boolean`, `float`, `decimal`, `double`, `base64binary`.

Uma tipagem forte dos dados permite maior compatibilidade dos dados apresentados em documentos XML, por linguagens de programação e outras tecnologias, como é o caso dos dados representados por SOAP, por exemplo, será visto na próxima seção.

2.5 SOAP

O protocolo SOAP é um padrão aberto que foi concebido em 1999, como uma recomendação do W3C, originada da proposta conjunta da Microsoft, IBM e Userland.com, entre outras empresas, sendo baseado nas especificações da XML-RPC, criada por David Winer, da Userland.com, em 1998 (AMUNDSEN, 2002). Neste trabalho será tomada como base, a versão 1.1 do SOAP, porém já existe uma recomendação da versão 1.2, do W3C. Assim, diferenças entre as duas versões também estarão sendo evidenciadas.

Embora tenha sido projetado para ser utilizado sobre vários protocolos de transporte da Internet, tais como SMTP e TCP (MSDN, 2003B), o protocolo SOAP é muito utilizado com o protocolo HTTP (JORGENSEN, 2002).

A problemática a ser solucionada pelo SOAP baseia-se no fato de que, em tratando-se de aplicações Web convencionais, os dados transportados que eram padronizados, com o uso do protocolo HTTP, eram apenas a URL, o verbo HTTP (GET ou POST) e o cabeçalho HTTP. As demais informações dependiam das estruturas específicas de cada servidor Web. Por outro lado, em tratando-se de comunicações entre aplicações de diferentes plataformas, existe a necessidade de um “idioma” comum, para que seja possível a comunicação entre os servidores (JORGENSEN, 2002). Assim, o SOAP funciona como este “idioma” comum, já que seu objetivo básico é utilizar a XML como forma de descrever informações de bibliotecas de tipos (AMUNDSEN, 2002), possibilitando a comunicação entre duas extremidades, dois servidores, sem a dependência de sistemas operacionais específicos. A troca de mensagens com SOAP é ilustrada pela figura 9.

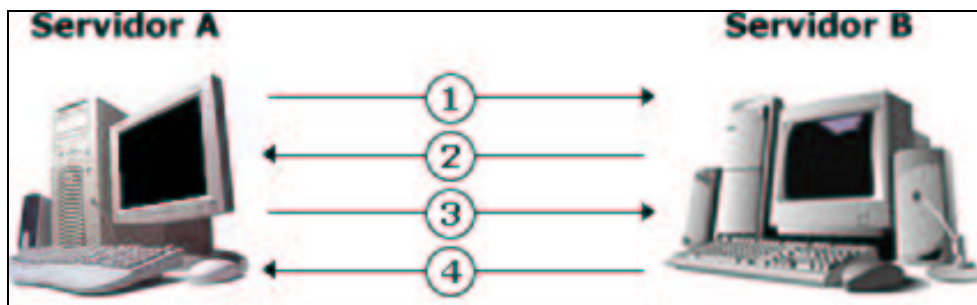


Figura 9: Etapas na comunicação entre servidores

Na figura 9, a comunicação representada pela seta de número um significa que o servidor A enviou uma solicitação em formato SOAP, pedindo informações sobre biblioteca de tipos ao servidor B, o qual responde ao servidor A na segunda seta. A seta de número três indica que o servidor A envia uma chamada a um método, devidamente formatado em um envelope SOAP. Em seguida, o servidor B responde, na quarta seta, com um envelope SOAP que contém os resultados da execução do método anteriormente solicitado.

O protocolo SOAP é definido por três partes: o envelope, as regras de codificação e a representação de RPC, os quais serão apresentados nas próximas três seções.

2.5.1 Envelope SOAP

As mensagens SOAP são representadas por envelopes SOAP, que têm a função de dizer o que tem na mensagem e para quem ela é destinada. Os envelopes, por sua vez, são constituídos por um cabeçalho (*header*) e por um corpo (*body*) (SOAP, 2000). Um exemplo esquemático é apresentado na figura 10.

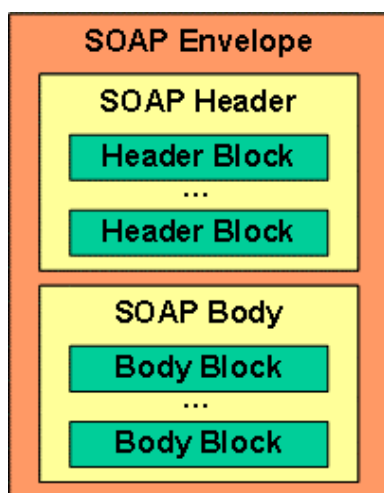


Figura 10: Representação de um envelope SOAP (MSDN, 2003B)

Sendo uma mensagem SOAP um documento XML, o envelope é representado por um elemento envelope, que obrigatoriamente deve estar na mensagem SOAP. Ele pode conter um atributo `encodingStyle`, que é representado por uma URI que indica as regras para a desserialização da mensagem SOAP (SOAP, 2000).

Como elemento filho de envelope, pode existir um elemento Header, que define atributos a serem utilizados por seus elementos filhos. Estes atributos são utilizados para a indicação do destinatário, representado pelo atributo `actor` (`role`, na versão 1.2 de SOAP) e da obrigatoriedade ou não do cabeçalho ser processado pelo destinatário, representado pelo atributo `mustUnderstand`. Os elementos filhos de Header são chamados blocos de cabeçalho (SOAP, 2000). É importante destacar que o termo “destinatário”, mencionado anteriormente, refere-se ao servidor para o qual a mensagem é destinada, o que significa que a mensagem não será processada por servidores intermediários, os roteadores.

Outro elemento filho obrigatório de envelope é Body, que deve estar localizado imediatamente depois do elemento header, caso ele exista. Os elementos filhos imediatos de Body são chamados blocos de corpo, nos quais podem ser definidos atributos encodingStyle, que são usados para indicar as regras de desserialização dos dados (SOAP, 2000).

Tanto os blocos de cabeçalho quanto os blocos de corpo devem ser qualificados através de *namespaces* (SOAP, 2000).

Por definição, um exemplo de envelope SOAP pode ser exemplificado pela figura 11.

1	<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
2	<soap:Header>
3	<AuthHeader soap:actor="http://theFirewall"
4	xmlns="http://keithba.com/security">
5	<UserName>Mel</UserName>
6	<Pwd>MelRocks!</Pwd>
7	</AuthHeader>
8	<Context xmlns=http://keithba.com/context
9	soap:mustUnderstand="true">
10	http://keithba.com/alerts/1234321
11	</Context>
12	</soap:Header>
13	<soap:Body>
14	<Alert xmlns="http://keithba.com/alerts">
15	Melissa is online!
16	</Alert>
17	</soap:Body>
18	</soap:Envelope>

Figura 11: Envelope SOAP (BALLINGER, 2003)

A mensagem apresentada na figura 11, o cabeçalho é constituído por dois blocos de cabeçalho: AuthHeader, e Context. O primeiro contém os elementos UserName e Pwd, o que sugere que esta mensagem tem o objetivo de realizar uma autenticação. O atributo soap:actor="http://theFirewall" define que o destinatário desta mensagem é representado pela URI "http://theFirewall". O bloco de cabeçalho identificado pelo elemento Context possui o atributo soap:mustUnderstand="true", que significa que todo o conteúdo presente no cabeçalho deve ser processado pelo destinatário. Esta mensagem possui apenas um bloco de corpo, Alert, cujo conteúdo é "Melissa is online!".

2.5.1.1 SOAP Fault

SOAP provê uma forma para reportar erros ocorridos no processamento de suas mensagens. Este mecanismo é representado pelo elemento Fault, que deve ser filho de Body, sendo constituído pelos seguintes elementos filhos (SOAP, 2003), segundo a versão 1.2 do SOAP:

- Code (representado pelo elemento faultcode, da versão 1.1): apresenta um código para retratar computacionalmente o erro ocorrido, sendo obrigatório um elemento filho Value, e opcional, um elemento Subcode, que possui um elemento filho obrigatório Value e um outro filho opcional Subcode.
- Reason (representado pelo elemento faultstring, da versão 1.1): tem a função de exibir uma mensagem entendível pelo usuário. É constituído por um ou mais elementos Text, que apresenta o texto referente ao erro.
- Detail (representado pelo elemento detail, na versão 1.1): apresenta detalhes, na forma de elementos filhos, sobre o erro. Ele é apresentado caso o erro tenha ocorrido no processamento do elemento Body.
- Node (correspondente ao elemento actor, na versão 1.1): identifica o elemento da mensagem SOAP onde ocorreu o erro.

Um exemplo de envelope SOAP indicando um erro no processamento da mensagem é apresentado na figura 12.

```

1  <?xml version='1.0' ?>
2  <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
   xmlns:rpc='http://www.w3.org/2003/05/soap-rpc'>
3    <env:Body>
4      <env:Fault>
5        <env:Code>
6          <env:Value>env:Sender</env:Value>
7          <env:Subcode>
8            <env:Value>rpc:BadArguments</env:Value>
9          </env:Subcode>
10       </env:Code>
11       <env:Reason>
12         <env:Text xml:lang="en-US">Processing error</env:Text>
13         <env:Text xml:lang="cs">Chyba zpracování</env:Text>
14       </env:Reason>
15       <env:Detail>
16         <e:myFaultDetails xmlns:e="http://travelcompany.example.org/faults">
17           <e:message>Name does not match card number</e:message>

```

```

18 |         <e:errorCode>999</e:errorCode>
19 |     </e:myFaultDetails>
20 | </env:Detail>
21 | </env:Fault>
22 | </env:Body>
23 | </env:Envelope>

```

Figura 12: Envelope SOAP reportando um erro (SOAP, 2003)

É possível notar no exemplo da figura 12, a representação de um erro, definido pelo elemento Fault, com os elementos Code, Reason e Detail.

2.5.2 Regras de codificação

Pode-se dizer que as regras de codificação SOAP têm a utilidade de direcionar a conversão dos conteúdos que são originalmente expressos em linguagens de programação, para o formato XML, de forma a seguir o formato do protocolo SOAP (BALLINGER, 2003). Esta conversão permite que os objetos sejam transportados via Internet, utilizando protocolo HTTP.

Desta forma, as regras para a codificação podem ser resumidas nos seguinte itens (SOAP, 2000):

1. Valores simples, de tipos específicos como strings e inteiros, são representados como valores de elementos, ou seja, um *character data*. O tipo do dado pode ser explicitado de três maneiras: através de um atributo indicando o tipo, estando o elemento dentro de um elemento que representa um array, ou de uma forma que o nome do elemento já identifica o tipo do dado por ele contido.
2. Valores compostos, tal como endereços, por exemplo, são representados como seqüências de elementos.
3. Arrays contidos em estruturas são descritos por elementos complexos com todos os valores do array como elementos filhos. Além disto, o novo elemento conterá um atributo “id”, do tipo ID, que funcionará como uma chave estrangeira pela declaração do array, relacionando-se com um atributo “href” do elemento que representa o array.

Considerando a classe apresentada na figura 13, expressa na linguagem C#, apresentando uma classe Address, o resultado da sua codificação é representado pela figura 14.

```

1 public class Address
2 {
3     Public String[] Street;
4     Public String City;
5     Public String State;
6     Public String ZipCode;
7 }
```

Figura 13: Classe a ser codificada (BALLINGER, 2003)

```

1 <tns:Address xmlns:tns="http://keithba.com">
2     <Street href="#id2" />
3     <City xsi:type="xsd:string">Redmond</City>
4     <State xsi:type="xsd:string">WA</State>
5     <ZipCode xsi:type="xsd:string">98045-0001</ZipCode>
6 </tns:Address>
7 <soapenc:Array id="id2" soapenc:arrayType="xsd:string[2]">
8     <Item>1 Microsoft Way</Item>
9     <Item>Suite 1</Item>
10 </soapenc:Array>
```

Figura 14: Resultado da codificação SOAP (BALLINGER, 2003)

No procedimento de codificação apresentado nas figuras 13 e 14, a classe Address tornou-se um elemento complexo com os conteúdos de seus atributos expressos como elementos simples. Seguindo o exposto no terceiro item, no caso do atributo Street, que é um array, tornou-se um elemento vazio, com o atributo href referenciando o id do elemento soapenc:Array, criado após o elemento Address, com as duas ruas constantes deste array.

2.5.3 Serialização XML

O processo de serialização XML consiste da utilização de recursos de programação para a transformação de dados de objetos para o formato XML, permitindo o transporte sobre a Internet. Ou seja, a serialização permite que classes sejam mapeadas para esquemas XML e que instâncias de objetos sejam mapeadas para instâncias XML do seu esquema

correspondente (BALLINGER, 2003). Para tanto, a serialização XML segue as regras de codificação definidas anteriormente.

Na plataforma .NET, a classe utilizada para a serialização é `XmlSerializer`, que oferece métodos para serialização, antes do envio dos dados, e desserialização, que é o processo realizado quando os dados chegam ao destino e precisam voltar ao formato de objetos (MSDN, 2003D).

Porém, o processo de serialização possui algumas particularidades, tal como a possibilidade de serializar apenas propriedades e atributos públicos. Além disso, ele não inclui informações sobre tipos abstratos de dados, não garantindo assim, que objetos sejam deserializados para seus tipos de dados originais (MSDN, 2003D).

Este serviço é normalmente oferecido pelas plataformas que implementam *Web Services*. Assim, a serialização é realizada automaticamente para retornar os dados, no formato SOAP, referentes à execução de cada método invocado.

2.5.4 SOAP RPC

Para que invocações a métodos possam ser transportadas, é necessária a formatação dos dados envolvidos segundo o protocolo SOAP. Assim, esta seção apresentará como os dados dos métodos devem ser organizados nas mensagens SOAP.

O estilo do mecanismo de chamada a métodos do SOAP é parecido com o estilo de tecnologias como CORBA e RMI, porém, de forma mais simplificada (BALLINGER, 2003). Para que um método possa ser chamado, é necessária a URI do objeto de destino, o nome do método, os parâmetros do método com seus respectivos tipos, e opcionalmente, a assinatura do método e dados adicionados ao elemento Header da mensagem (SOAP, 2000).

Desta forma, para que o método definido em C# por `public Address SubmitAddress(Address addr, bool dontSave)`, seja chamado, é necessário utilizar a representação mostrada na figura 15.

1	<pre><soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns="http://soapinterop.org"</pre>
---	--

```

    xmlns:types="http://soapinterop.org/encodedTypes"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
2  <soap:Body
    soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
3  <tns:SubmitAddress>
4  <addr href="#id1" />
5  <dontSave xsi:type="xsd:boolean">>false</dontSave>
6  </tns:SubmitAddress>
7  <tns:Address id="id1" xsi:type="tns:Address">
8  <Street href="#id2" />
9  <City xsi:type="xsd:string">Redmond</City>
10 <State xsi:type="xsd:string">WA</State>
11 <ZipCode xsi:type="xsd:string">98045-0001</ZipCode>
12 </tns:Address>
13 <soapenc:Array id="id2" soapenc:arrayType="xsd:string[2]">
14 <Item>1 Microsoft Way</Item>
15 <Item>Suite 1</Item>
16 </soapenc:Array>
17 </soap:Body>
18 </soap:Envelope>

```

Figura 15: Chamada remota a um método com SOAP (BALLINGER, 2003)

É possível perceber, entre as linhas 3 e 6, o elemento `SubmitAddress`, que representa o método que se deseja executar, tendo como seus elementos filhos, os parâmetros `dontSave`, do tipo `boolean`, e `addr`, referente à classe `Address`, que é especificada através do elemento `Address`, conforme as regras de codificação do SOAP.

2.5.5 Ligação entre HTTP e SOAP

SOAP é passível de ser implementado sob vários protocolos de transporte. Porém, normalmente é implementado para trabalhar com o protocolo HTTP, e por isto serão apresentados aspectos gerais do trabalho conjunto destes dois protocolos.

HTTP é um protocolo para transferência de informações de Web Sites. Por isto, nas requisições realizadas, o protocolo inclui nos cabeçalhos das mensagens, várias informações, dentre as quais, algumas sobre formatos que o cliente suporta (SOARES, 1995).

No trabalho com *Web Services*, o protocolo HTTP inclui em suas mensagens o conteúdo SOAP. Assim, como o modelo de funcionamento permite que pares de solicitação/resposta sejam automaticamente correlacionados, através dos dados definidos pelo protocolo HTTP, é possível a uma aplicação de *Web Services*, decidir por processar ou não a mensagem recebida (SOAP, 2003). Ou seja, quando uma mensagem é enviada,

passa por servidores intermediários, caso o destinatário especificado no cabeçalho HTTP seja o *host* atual, este a processa, caso contrário, a mensagem prossegue em sua rota.

Um exemplo do uso do protocolo HTTP para uma requisição SOAP é apresentado na figura 16.

1	POST /tcc/noticia.aspx HTTP/1.1
2	Host: nds03
3	Content-Type: text/xml; charset=utf-8
4	Content-Length: nnnn
5	SOAPAction: "http://nds03/tcc/NoticiaPorID"
6	
7	<?xml version="1.0" encoding="utf-8"?>
8	<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
9	<soap:Body>
10	<NoticiaPorID xmlns="http://nds03/tcc">
11	<id_noticia>10</id_noticia>
12	</NoticiaPorID>
13	</soap:Body>
14	</soap:Envelope>

Figura 16: Requisição SOAP utilizando HTTP

Entre as linhas 1 e 5, está colocado o cabeçalho HTTP, onde é possível identificar que esta é uma mensagem de requisição, através de POST, na primeira linha, que também indica o recurso para o qual a mensagem é destinada, e a versão do protocolo HTTP. A segunda linha identifica o nome do *host* de destino. A terceira indica o tipo de conteúdo, que, segundo a especificação do SOAP, é obrigatório que seja igual a “text/xml” (SOAP, 2000). A terceira linha define ainda, o tipo de codificação de caracteres, no caso, “UTF-8”. A linha é preenchida com o tamanho do conteúdo da mensagem, e SOAPAction, na quinta linha, define o método que será invocado na máquina remota. Conforme a especificação de SOAP RPC, as linhas de 10 a 12 indicam o método e seu parâmetro. Uma possível resposta a esta requisição é apresentada na figura 17.

1	HTTP/1.1 200 OK
2	Content-Type: text/xml; charset=utf-8
3	Content-Length: nnnn
4	
5	<?xml version="1.0" encoding="utf-8"?>
6	<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

7	<soap:Body>
8	<NoticiaPorIDResponse xmlns="http://nds03/tcc">
9	<NoticiaPorIDResult>
10	<id_noticia>10</id_noticia>
11	<titulo>30 anos da ULBRA</titulo>
12	<resumo>A Universidade Luterana do Brasil comemora os seus 30
13	anos.</resumo>
13	</NoticiaPorIDResult>
14	</NoticiaPorIDResponse>
15	</soap:Body>
16	</soap:Envelope>

Figura 17: Resposta SOAP utilizando HTTP

Na resposta, o cabeçalho HTTP contém menos informações, até mesmo porque, cada resposta é relacionada pelo protocolo, a sua respectiva requisição. Assim, na mensagem de resposta, não existem os campos host, nem SOAPAction. Os dados referentes ao retorno do método estão entre as linhas 9 e 13.

2.6 HTTP-GET e HTTP-POST

Além do protocolo SOAP, também é utilizado o próprio protocolo HTTP, através de seus verbos GET e POST, para a representação dos dados a serem trocados. Uma das diferenças entre estes protocolos e o SOAP, é sua funcionalidade limitada, no sentido de que os dados transportados são baseados em pares nome/valor (PAYNE, 2001). Desta forma, apenas SOAP consegue transportar tipos de dados mais complexos, tal como classes, por exemplo. Por outro lado, em comunicações envolvendo grandes volumes de dados ou em situações em que existe escassez de recursos computacionais, a utilização dos protocolos HTTP-GET e HTTP-POST é mais recomendada, já que, por agregarem menos dados, garantem maior desempenho.

A principal característica do protocolo HTTP-GET é o fato de as requisições acrescentarem pares nome/valor referentes a parâmetros na própria *querystring* (PAYNE, 2001). Um exemplo de requisição utilizando HTTP-GET é apresentado na figura 18.

1	GET /tcc/noticia.asmx/NoticiaPorID?id_noticia=10 HTTP/1.1
2	Host: nds03

Figura 18: Requisição HTTP-GET

Pode-se notar na primeira linha, os termos destacados referem-se a um par nome/valor, que são adicionados à *querystring* referente ao método NoticiaPorID. A resposta a esta solicitação é apresentada na figura 19.

1	HTTP/1.1 200 OK
2	Content-Type: text/xml; charset=utf-8
3	Content-Length: nnnn
4	
5	<?xml version="1.0" encoding="utf-8"?>
6	<NoticiaDetalhe xmlns="http://nds03/tcc">
7	<id_noticia>10</id_noticia>
8	<titulo>30 anos da ULBRA</titulo>
9	<resumo>A Universidade Luterana do Brasil comemora os seus 30
	anos.</resumo>
10	</NoticiaDetalhe>

Figura 19: Resposta HTTP-GET

Como se pode notar, o resultado da execução do método, utilizando-se o protocolo HTTP-GET apresenta maior simplicidade que aquele retornado pelo protocolo SOAP, apresentado na figura 17. Entre as linhas 1 e 3 está disposto o cabeçalho HTTP. A linha 5 identifica que os dados seguem a versão 1.0 da XML e que o esquema de codificação de caracteres utilizado é o UTF-8. Entre as linhas 6 e 10 estão os dados resultantes da execução do método NoticiaPorID, que possui um elemento raiz NoticiaDetalhe, com os elementos atômicos id_noticia, titulo e resumo.

A requisição via protocolo HTTP-POST é caracterizada pela disposição dos parâmetros diretamente na mensagem de solicitação, e não na *querystring*, conforme faz o HTTP-GET (PAYNE, 2001). Assim sendo, um exemplo de solicitação HTTP-POST é apresentado na figura 20.

1	POST /tcc/noticia.aspx/NoticiaPorID HTTP/1.1
2	Host: nds03
3	Content-Type: application/x-www-form-urlencoded
4	Content-Length: nnnn
5	
6	id_noticia=10

Figura 20: Requisição HTTP-POST

As linhas 1 e 4 apresentam o cabeçalho HTTP referente à solicitação, e na linha 6, é colocado o nome do parâmetro e o seu valor, para que o método `NoticiaPorID`, explicitado na primeira linha, possa ser executado. A resposta para esta requisição é a mesma do protocolo HTTP-GET, que é apresentada na figura 19.

É importante destacar que a escolha de qual protocolo utilizar é feita pelo consumidor do *Web Service*. No momento da criação do *Web Service*, o desenvolvedor não precisa se preocupar com este item, já que os documentos WSDL, que serão apresentados na próxima seção, descrevem as formas de comunicação através dos três tipos de protocolos.

2.7 WSDL

Antes que as comunicações via *Web Services* sejam iniciadas, é necessário que a máquina que vai consumi-los tenha conhecimento dos formatos de mensagens que serão enviadas e recebidas do servidor. Para este fim, foi criada a WSDL (WSDL, 2003), que é uma linguagem baseada em XML, para a representação das funcionalidades de um *Web Service*.

A estrutura dos documentos WSDL divide-se em uma parte abstrata, representada pelos elementos `types`, `message` e `portType`, e por uma parte concreta, representada pelos elementos `binding` e `service`. Nas seções abstratas são descritos os dados e as operações suportadas pelo *Web Service*, enquanto que nas seções concretas são definidas informações concretas para a realização das operações oferecidas (BALLINGER, 2003). Todos estes elementos são colocados como filhos do elemento raiz, `definitions`, seguindo a mesma ordem com que foram apresentados. Como documentos WSDL utilizam *namespaces*, a declaração do elemento `definitions` é feita conforme apresentado na figura 18.

```
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:s0="http://nds03.org/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://nds03.org/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
```

Figura 18: Elementos message em um documento WSDL

Os exemplos apresentados nas seções 2.3.1, 2.3.2, 2.3.3, 2.3.4 e 2.3.5 são referentes à descrição de um *Web Service* que possui um método `noticiasCursos`, que tem a função de retornar um objeto `Noticia`.

2.7.1 Elemento types

O elemento `types` é responsável por definir um esquema para os dados que serão trafegados nas requisições e respostas dos métodos. A linguagem padrão utilizada para a representação deste esquema é XML Schema (WSDL, 2003). Um exemplo da utilização deste elemento em um documento WSDL é apresentado na figura 19.

```

1 <types>
2   <s:schema elementFormDefault="qualified" targetNamespace="http://nds03.org/">
3     <s:element name="noticiasCursos">
4       <s:complexType />
5     </s:element>
6     <s:element name="noticiasCursosResponse">
7       <s:complexType>
8         <s:sequence>
9           <s:element minOccurs="0" maxOccurs="1"
10          name="noticiasCursosResult" type="s0:Noticia" />
11         </s:sequence>
12       </s:complexType>
13     </s:element>
14     <s:complexType name="Noticia">
15       <s:sequence>
16         <s:element minOccurs="1" maxOccurs="1" name="id_noticia" type="s:int" />
17         <s:element minOccurs="0" maxOccurs="1" name="titulo" type="s:string" />
18         <s:element minOccurs="0" maxOccurs="1" name="resumo" type="s:string" />
19       </s:sequence>
20     </s:complexType>
21     <s:element name="Noticia" nillable="true" type="s0:Noticia" />
22 </s:schema>
</types>

```

Figura 19: Elemento `types` em um documento WSDL

Um aspecto importante deste exemplo está na linha 2, em que existe um atributo chamado `targetNamespace`, cujo valor é igual a “`http://nds03.org`”. Isto significa que a *namespace* definida por “`http://nds03.org`” conterá os nomes que serão definidos nesta seção (SIDDQUI, 2001), permitindo a sua utilização nas seções seguintes do documento

WSDL, através do prefixo “s0”, que está declarado no elemento definitions (vide figura 18).

É possível a utilização de outras linguagens de representação de esquemas, desde que a estrutura dos elementos do esquema estejam identificados através das *namespaces* correspondentes (WSDL, 2003).

2.7.2 Elementos message

Os elementos message definem formatos abstratos para os dados que são enviados (valor do atributo name possui o sufixo “In”) ou recebidos (valor do atributo name possui o sufixo “Out”) pelos *Web Services* (WSDL, 2001). Em outras palavras, pode-se dizer que elementos message representam os parâmetros dos métodos (SIDDIQUI, 2001), comumente chamados de “operações”, em termos de WSDL.

Estes elementos são compostos por elementos part, nos quais é estabelecida uma ligação com a seção types, através do atributo element. Exemplos da utilização destes elementos são exibidos na figura 20.

1	<message name="noticiasCursosSoapIn">
2	<part name="parameters" element="s0:noticiasCursos" />
3	</message>
4	<message name="noticiasCursosSoapOut">
5	<part name="parameters" element="s0:noticiasCursosResponse" />
6	</message>
7	<message name="noticiasCursosHttpGetIn" />
8	<message name="noticiasCursosHttpGetOut">
9	<part name="Body" element="s0:Noticia" />
10	</message>
11	<message name="noticiasCursosHttpPostIn" />
12	<message name="noticiasCursosHttpPostOut">
13	<part name="Body" element="s0:Noticia" />
14	</message>

Figura 20: Elementos message em um documento WSDL

No exemplo apresentado, estão representados os tipos de parâmetros e os tipos de retornos possíveis para cada um dos protocolos utilizados pelo *Web Service* em questão.

2.7.3 Elementos portType

Os elementos `portType` representam conjuntos abstratos de operações que são suportadas pelo *Web Service*. Estas operações são definidas concretamente através dos elementos `binding`, cuja ligação é estabelecida através do atributo `type`, de `binding`, e `name`, de `portType` (SIDDIQUI, 2001) (WSDL, 2001). Além disto, estes elementos relacionam os pares requisição-resposta definidos anteriormente, nos elementos `message`, através de atributos `message` (MSDN, 2003E).

Na versão 2.0 da WSDL, os elementos `portType` foram renomeados para interfaces (WSDL, 2003). A figura 21 apresenta um exemplo da utilização destes elementos.

```

1 <portType name="NoticiasSoap">
2   <operation name="noticiasCursos">
3     <input message="s0:noticiasCursosSoapIn" />
4     <output message="s0:noticiasCursosSoapOut" />
5   </operation>
6 </portType>
7 <portType name="NoticiasHttpGet">
8   <operation name="noticiasCursos">
9     <input message="s0:noticiasCursosHttpGetIn" />
10    <output message="s0:noticiasCursosHttpGetOut" />
11  </operation>
12 </portType>
13 <portType name="NoticiasHttpPost">
14   <operation name="noticiasCursos">
15     <input message="s0:noticiasCursosHttpPostIn" />
16     <output message="s0:noticiasCursosHttpPostOut" />
17   </operation>
18 </portType>

```

Figura 21: Elementos `portType` em um documento WSDL

Pode-se notar a existência de três elementos `portType`, equivalentes a cada um dos protocolos envolvidos: SOAP, HTTP-GET e HTTP-POST.

2.7.4 Elementos binding

Os elementos `binding` apresentam informações concretas referentes a respectivos `portType`, sobre o protocolo que será utilizado no transporte das mensagens, sobre os formatos das mensagens (WSDL, 2001) e sobre a operação a ser executada (MSDN, 2003E). Como elemento filho de `binding`, existirá um elemento que definirá o protocolo

utilizado. Este é o caso dos elementos soap:binding e http:binding, nas linhas 2 e 10 da figura 22, por exemplo.

```

1 <binding name="NoticiasSoap" type="s0:NoticiasSoap">
2   <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
   style="document" />
3   <operation name="noticiasCursos">
4     <soap:operation soapAction="http://nds03.org/noticiasCursos"
   style="document" />
5     <input> <soap:body use="literal" /> </input>
6     <output> <soap:body use="literal" /> </output>
7   </operation>
8 </binding>
9 <binding name="NoticiasHttpGet" type="s0:NoticiasHttpGet">
10  <http:binding verb="GET" />
11  <operation name="noticiasCursos">
12    <http:operation location="/noticiasCursos" />
13    <input> <http:urlEncoded /> </input>
14    <output> <mime:mimeXml part="Body" /> </output>
15  </operation>
16 </binding>
17 <binding name="NoticiasHttpPost" type="s0:NoticiasHttpPost">
18  <http:binding verb="POST" />
19  <operation name="noticiasCursos">
20    <http:operation location="/noticiasCursos" />
21    <input> <mime:content type="application/x-www-form-urlencoded" />
   </input>
22    <output> <mime:mimeXml part="Body" /> </output>
23  </operation>
24 </binding>

```

Figura 22: Elementos binding em um documento WSDL

Na linha 2, o atributo transport="http://schemas.xmlsoap.org/soap/http" diz que para o transporte das mensagens será utilizado o protocolo HTTP. Entre as linhas 3 e 6, é declarada a operação cujo nome é noticiasCursos, que, na linha 4, é definido que a ação a ser executada via SOAP é definida pela URI http://nds03.org/noticiasCursos. Além disso, nas linhas 5 e 6 são definidos os tipos de entradas e saídas para a execução da operação definida na linha 4, utilizando-se SOAP. Situações equivalentes ocorrem nos demais elementos binding, que utilizam o protocolo HTTP, com seus verbos GET e POST.

2.7.5 Elemento service

O elemento `service` é um conjunto de elementos `port` relacionados, os quais se referem a elementos `binding` previamente definidos e que definem o *endpoint* atual, ou seja, o endereço lógico do *Web Service* (SIDDIQUI, 2001). A figura 23 apresenta um exemplo de elemento `service`.

1	<service name="Noticias">
2	<port name="NoticiasSoap" binding="s0:NoticiasSoap">
3	<soap:address location="http://nds03/tcc/teste.asmx" />
4	</port>
5	<port name="NoticiasHttpGet" binding="s0:NoticiasHttpGet">
6	<http:address location="http://nds03/tcc/teste.asmx" />
7	</port>
8	<port name="NoticiasHttpPost" binding="s0:NoticiasHttpPost">
9	<http:address location="http://nds03/tcc/teste.asmx" />
10	</port>
11	</service>

Figura 23: Elementos `service` em um documento WSDL

Em resumo, pode-se dizer que a seção `types` define todas as estruturas que serão utilizadas pelo *Web Service*. Os elementos `message` definem os formatos dos dados trocados, ou seja, dos parâmetros e dos retornos, através da utilização das estruturas definidas no elemento `types`. Os elementos `portTypes` montam as estruturas dos métodos, por meio dos elementos `operations`, que definem os nomes dos métodos, além de reunirem pares solicitação-resposta, através dos parâmetros definidos nos elementos `message`. As seções `binding` estabelecem a ligação entre as estruturas dos métodos, representadas pelos elementos `portType`, com as ações concretas que o *Web Service* suporta, definidas pelos elementos `soap:operation` e `http:operation`. Por fim, o elemento `service` define a localização concreta do *Web Service*, além de estabelecer ligações com as operações representadas pelos elementos `binding`.

2.8 UDDI

No processo de desenvolvimento de uma aplicação, pode ser necessária a utilização de serviços específicos, que muitas vezes encontram-se em localidades distantes, em empresas cujo nome o desenvolvedor nunca ouviu falar. Com a finalidade de facilitar a localização de *Web Services*, foi criada a UDDI (UDDI, 2003).

Esta tecnologia surgiu como uma iniciativa conjunta da Microsoft, IBM e Ariba, em 2000, e é caracterizada pela existência de bancos de dados abertos, que permitem a busca e publicação de *Web Services*, através de seus meta-dados (BALLINGER, 2003), que são compostos de acordo com o protocolo UDDI (UDDI, 2003).

Basicamente, existem duas formas de interação com os repositórios UDDI:

- Através de interface com usuário, oferecida pelos repositórios. Através destas interfaces, é possível a submissão de consultas manualmente, além do preenchimento de todos os dados necessários à publicação de um *Web Service*, sem a necessidade de conhecimento sobre a criação de documentos UDDI. Algumas destas interfaces podem ser acessadas pelos seguintes endereços: <http://uddi.ibm.com>, <http://uddi.microsoft.com>, <http://uddi.sap.com> e <http://www.ntt.com/uddi>.
- Programaticamente, através de uma API de programação. As funcionalidades desta API podem ser divididas em dois tipos: funções de investigação, responsáveis por recuperação de informações sobre serviços, agregando as seguintes operações: `find_binding`, `find_business`, `find_service`, `find_tModel`, `get_bindingDetail`, `get_businessDetail`, `get_BusinessDetailExt`, `get_serviceDetail` e `get_tModelDetail` (BALLINGER, 2003); e funções de publicação, responsáveis por realizar operações referentes à publicação de serviços em repositórios UDDI, compostas pelas seguintes operações: `save_binding`, `save_business`, `save_service`, `save_tModel`, `delete_binding`, `delete_business`, `delete_service`, `delete_tModel`, `get_authToken`, `discard_authToken` e `get_registeredInfo` (UDDI, 2002).

Com base no exposto, pode-se dizer, que o ciclo para o consumo de um *Web Service* inicia-se com a descoberta dos serviços que serão utilizados, que é realizada ainda em tempo de projeto. Por outro lado, a publicação é a última etapa a ser realizada pelo projetista de *Web Services*. Assim, as primeiras etapas da criação do serviço correspondem à sua implementação efetiva, que será apresentada na próxima seção.

2.9 Criação de *Web Services* da plataforma .NET

Na plataforma .NET as funcionalidades dos *Web Services* podem ser definidas em arquivos do tipo ASMX. Para tanto, podem ser utilizadas quaisquer linguagens de programação suportadas em .NET. Além deste, um outro requisito necessário para que as funcionalidades de um *Web Service* possam ser acessadas por outras aplicações, é que o arquivo ASMX esteja colocado em uma pasta virtual de um servidor Web que possua o IIS e a plataforma .NET instalada.

Estas tarefas garantem que um *Web Service* esteja criado e disponível para ser consumido por outros computadores. Adicionalmente, o desenvolvedor pode publicar seus *Web Services* em repositórios UDDI, com a finalidade de divulgar seus serviços.

Nas seções a seguir serão apresentados detalhes da criação de *Web Services* na plataforma .NET.

2.9.1 Arquivos ASMX

Os arquivos ASMX seguem a sintaxe dos programas escritos nas linguagens C#, Visual Basic .NET e JScript .NET (PAYNE, 2001). Porém, as principais diferenças dos arquivos ASMX são as seguintes (BALLINGER, 2003):

- A primeira linha possui uma diretiva *WebService* que deve ser composta da seguinte forma:

```
<%@ WebService Language="C#" Class="NomeDaClasse" %>
```
- Possuem referências às *namespaces* System e System.Web.Services. Neste caso, a palavra *namespace* possui significado diferente daquele apresentado na seção 2.3. Na plataforma .NET, *namespaces* referem-se a agrupamentos de classes, que são equivalentes aos pacotes em Java.
- São acrescentados atributos *WebMethod* nos métodos das classes. Isto significa que os métodos são passíveis de serem invocados remotamente.
- Recomenda-se que sejam adicionados às declarações das classes, o atributo *WebService*, com o objetivo de definir uma descrição e uma *namespace* referentes ao serviço.

Um exemplo de arquivo ASMX, para a realização de somas de dois valores é apresentado na figura 24.

```

1 <%@ WebService Language="C#" Class="Soma" %>
2
3 using System;
4 using System.Web.Services;
5
6 [WebService(Description="WebService para soma de dois
valores",Namespace="http://nds03/michael")]
7 public class Soma : WebService {
8
9     [WebMethod(Description="Soma entre dois valores float")]
10    public float somar(float valor1, float valor2)
11    {
12        return valor1 + valor2;
13    }
14 }

```

Figura 24: Exemplo de arquivo ASMX

Na primeira linha, é definido que este arquivo refere-se a um *Web Service*, que a linguagem utilizada é C# e que a classe a ser tornada pública é “Soma”. Nas linhas 3 e 4, são importadas respectivamente as *namespaces* System e System.Web.Services. A linha 6 define o atributo WebService, em que são determinadas também a descrição do serviço e um valor para a *namespace*, a qual corresponderá ao atributo *targetNamespace*, do documento WSDL referente a este serviço. Na linha 7 é definida a classe Soma, que herda da classe WebService, que se encontra na *namespace* System.Web.Services. Entre as linhas 9 e 12 é definido um método da Web, chamado somar, em que é definida uma descrição, por meio da propriedade Description, e que recebe dois valores do tipo float e retorna o resultado da soma deles.

Para o atributo WebMethod, as propriedades suportadas são as seguintes (PAYNE, 2001):

- *BufferResponse*: especifica se o retorno do método será armazenado em *buffer* no servidor, antes de ser enviado ao cliente. O valor padrão é “true”.
- *CacheDuration*: define o tempo pelo qual a resposta da execução de um método deve ser armazenada em *cache*. O valor padrão é “0”.
- *Description*: define uma descrição do método para o cliente, quando este acessa o arquivo ASMX através do navegador.
- *EnableSession*: especifica a ativação ou não do estado de sessão. É ativado por padrão.

- **MessageName:** define o nome pelo qual o método será invocado, que normalmente é definido pelo próprio nome do método, porém é útil quando existem dois métodos de mesma assinatura e funções diferentes.
- **TransactionOption:** determina o tipo de transação usada na execução do método. Os tipos de transação são: Disabled, NotSupported, Supported, Required, RequiresNew.
- **TypeID:** atributo herdado da classe Attribute, utilizado para identificar unicamente uma instância de objeto desta classe.

Para a realização de testes com os *Web Services* criados, pode-se acessá-los através do *browser*, permitindo o conhecimento de seus componentes. Assim, a visualização do arquivo apresentado é exibida na figura 25.

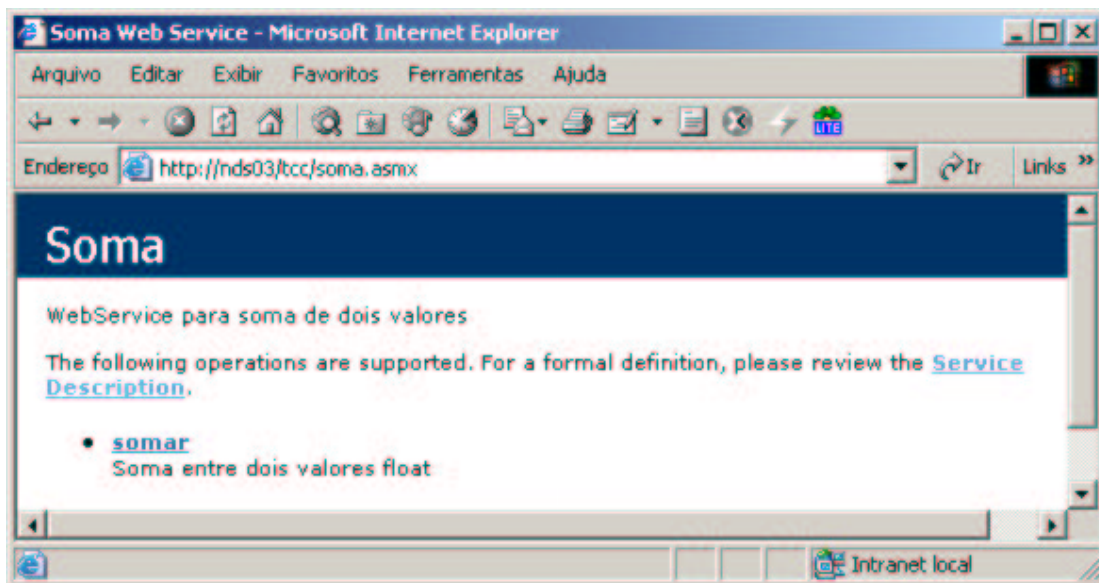


Figura 25: Visualização de um arquivo ASMX em um browser

Na figura 25 pode-se notar que o nome da classe criada é Soma, a qual possui um método chamado somar. Para acessar este método, basta clicar sobre o link. Será aberta uma nova janela para requerer os valores que são passados como parâmetros para a execução deste método. Esta janela é apresentada na figura 26.

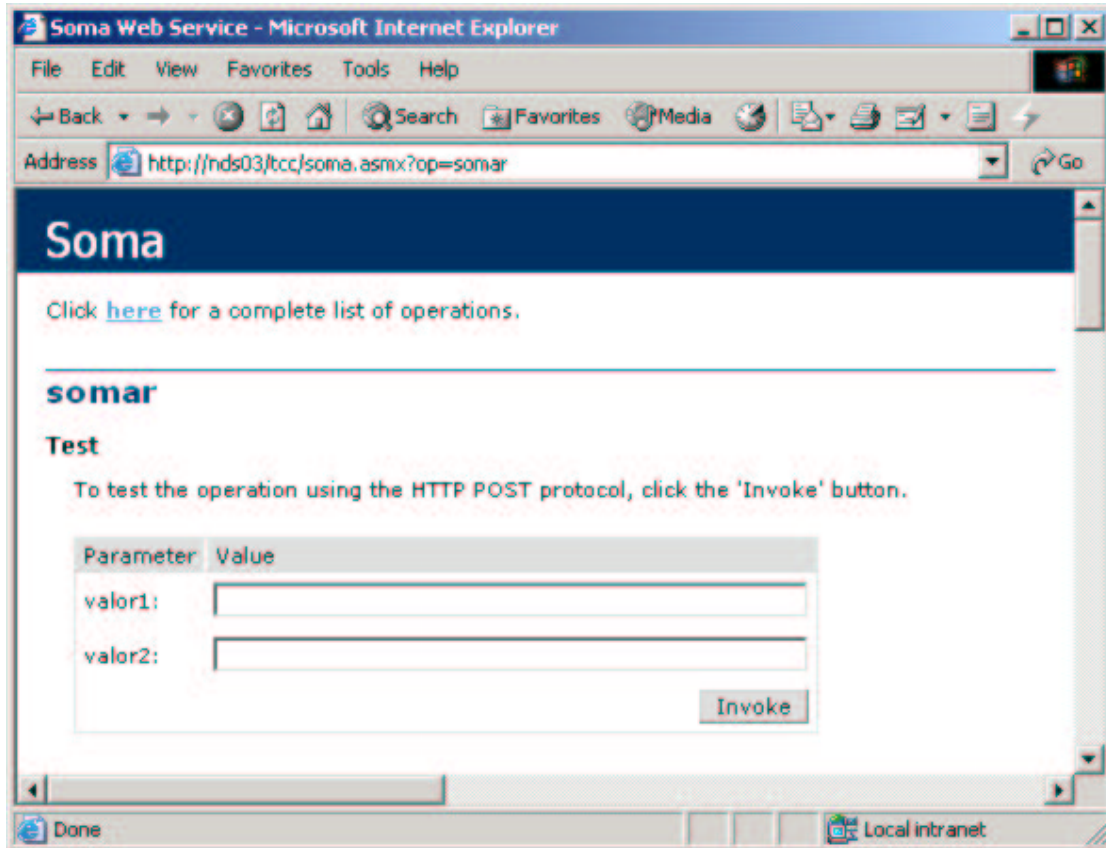


Figura 26: Interface para inserção dos parâmetros do método somar

Após serem inseridos valores para os parâmetros, pode-se invocar o método clicando no botão Invoke. No caso dos testes realizados via navegador, é utilizado o protocolo HTTP-POST para requisição e resposta. O resultado da execução do método é retornado em XML, seguindo o formato HTTP-POST de resposta para este método, conforme apresentado na figura 27.

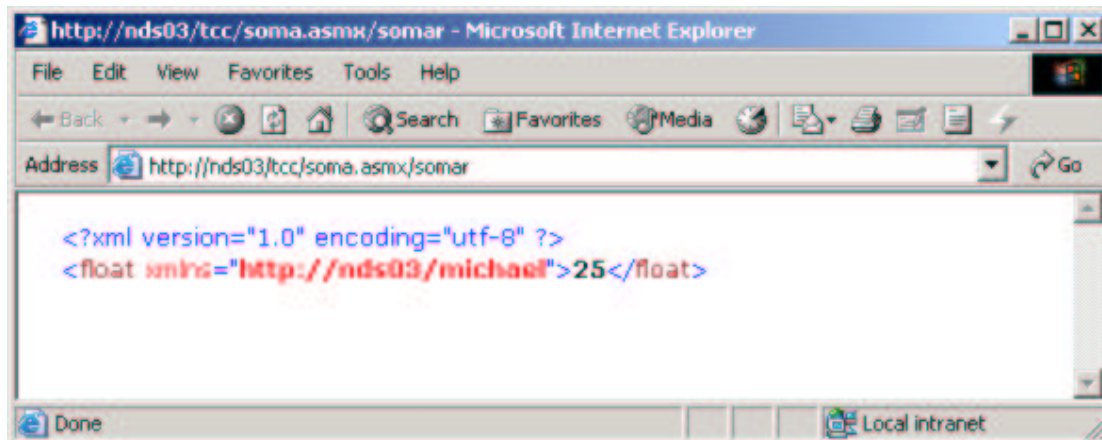


Figura 27: Resultado da execução do método somar

Na figura 27, para apresentar a resposta é criado um elemento float, cuja *namespace* é aquela definida no elemento WebService, e cujo conteúdo é o valor “25”.

Para o acesso ao documento WSDL que descreve este *Web Service*, é adicionado, ao final do endereço do serviço, na barra de endereços do navegador de Internet, o termo “?WSDL”.

2.9.2 Autenticação de *Web Services*

Muitas vezes, os *Web Services* são criados com contrapartida financeira, com o intuito de oferecer serviços como qualquer outro, incluindo a necessidade de pagamento. Para isto, é necessário que os *Web Services* sejam consumidos apenas por pessoas autorizadas. Com esta finalidade, são utilizadas entidades separadas, chamadas cabeçalhos de SOAP.

Para a utilização deste tipo de cabeçalhos, cria-se uma classe que herda da classe `System.Web.Services.Protocols.SoapHeader`, que servirá para reunir as informações de autenticação. Esta classe pode ser adicionada ao mesmo arquivo referente ao *Web Service* (PAYNE, 2001). A figura 28 apresenta um exemplo deste tipo de classe.

```

1 using System.Web.Services;
2 using System.Web.Services.Protocols;
3
4 public class Autenticacao : SoapHeader {

```

```

5 | public string login;
6 | public string senha;
7 | }

```

Figura 28: Classe para representação de cabeçalho de SOAP

Tendo sido criada uma classe `Autenticacao`, que herda de `SoapHeader`, declara-se uma instância desta classe na classe que se deseja proteger. Cada método a ser protegido deve então, utilizar o atributo `SoapHeader`, que se utilizará da instância criada (PAYNE, 2001). Além disto, a classe de cabeçalho de SOAP representará informações de autenticação de usuários como uma classe qualquer que possua esta finalidade. A estrutura da utilização desta classe é exemplificada na figura 29.

```

1 | using System;
2 | using System.Web.Services;
3 | using System.Web.Services.Protocols;
4 |
5 | public class Autenticacao : SoapHeader {
6 |     public string login;
7 |     public string senha;
8 | }
9 |
10 | public class ServicoSeguro : WebService {
11 |     public Autenticacao soapHeader;
12 |
13 |     [WebMethod()] [SoapHeader("soapHeader")]
14 |     public float somar(float valor1, float valor2)
15 |     {
16 |         if((soapHeader.login == "michael") && (soapHeader.senha == "schuenck"))
17 |             return valor1 + valor2;
18 |     }

```

Figura 29: Serviço autenticado através de cabeçalho de SOAP

Os dois novos aspectos do exemplo mostrado na figura 29 são, na linha 16, o uso do atributo `SoapHeader`, e, nas linhas 19 e 20, a verificação dos dados da classe de cabeçalho de SOAP com os dados corretos, antes que as ações do método sejam executadas.

Na utilização deste serviço em uma aplicação qualquer, deve-se definir os valores dos atributos de uma instância da classe `Autenticacao`, a qual deve ser atribuída ao atributo `soapHeader`, de uma instância da classe `ServicoSeguro` (PAYNE, 2001). Caso as informações sejam autenticadas, a invocação do método `somar` é realizada.

Tendo sido apresentadas as principais características da criação de *Web Services*, a próxima seção apresentará as metodologias para o consumo deles.

2.10 Consumo de *Web Services* na plataforma .NET

Na plataforma .NET, o consumo de *Web Services* pode ser realizado por aplicações escritas em quaisquer das linguagens suportadas. Porém, para que os serviços possam ser efetivamente utilizados nas aplicações finais, são necessárias algumas etapas, que serão mostradas nas próximas seções.

2.10.1 Geração da classe *proxy*

A geração de uma classe *proxy* é uma tarefa realizada automaticamente pela plataforma .NET, através do programa `wSDL`, que se utiliza do documento WSDL referente ao serviço (PAYNE, 2001). Para a geração da classe *proxy* do *Web Service* apresentado na figura 24, utiliza-se a seguinte instrução, na linha de comando:

```
wSDL http://nds03/tcc/soma.asmx?wSDL /out:soma.cs
```

Algumas das opções do comando `wSDL` são as seguintes:

- ***/language***: tem a função de definir a linguagem em que será gerada a classe *proxy*. A linguagem padrão é C#.
- ***/namespace***: define um nome para a *namespace* na qual a classe *proxy* estará inserida.
- ***/protocol***: define o protocolo que será utilizado nas comunicações do *Web Service*. As opções são HTTP-GET, HTTP-POST e SOAP, que é o protocolo padrão.
- ***/out***: define o nome do arquivo que será gerado.

A classe gerada com a execução da instrução apresentada é apresentada no anexo C.

Sendo criada a classe *proxy*, é necessária a sua compilação, para que possa ser efetivamente utilizada pelas aplicações que vão consumir os *Web Services*. Esta etapa será apresentada na próxima seção.

2.10.2 Compilação da classe *proxy*

A compilação da classe *proxy* é realizada com o compilador da linguagem na qual a classe foi gerada (PAYNE, 2001). Assim, programas escritos em C# são compilados com o programa `csc`, em VB .NET com o `vbc` e em Jscript .NET com o programa `jsc`. Para compilar a classe *proxy* gerada anteriormente, utiliza-se a seguinte instrução:

```
csc /target:library soma.cs
```

Esta instrução gera um arquivo chamado `soma.dll`, que deverá ser referenciado quando da compilação do programa que irá consumir o *Web Service*. Um exemplo de programa consumidor, na linguagem C#, é apresentado na figura 30.

```

1 using System;
2 class Consumidor {
3     public static void Main(String[] args){
4         Soma soma = new Soma();
5         float a = 5;
6         float b = 20;
7         Console.WriteLine("O valor da soma é: {0}",soma.somar(a,b).ToString());
8     }
9 }
```

Figura 31: Aplicação C# invocando o método `somar` do *Web Service* da figura 24

Esta é uma aplicação console, que possui uma classe `Consumidor`, cujo método principal é responsável por instanciar um objeto da classe `Soma`, e apresentar o resultado da invocação do método `somar`, em que são passados dois valores `float` como parâmetros. Para a compilação desta aplicação é utilizada a seguinte instrução.

```
csc /reference:soma.dll Consumidor.cs
```

Como se pode notar, em aplicações console não é utilizada nenhuma referência à classe *proxy* no próprio código. No caso de aplicações Web, é necessário que o arquivo `.dll`

gerado pela compilação seja colocado em um diretório chamado bin, filho do diretório que conterá as páginas que se utilizarão do *Web Service*. Neste caso, a compilação da classe *proxy* deverá ser realizada através da instrução apresentada abaixo.

```
csc /target:library /out:.\bin\soma.dll soma.cs
```

Além das opções apresentadas, o compilador de programas em C# possui várias outras, que incluem outras formas de destino, geração de documentos XML para documentação, habilitação de otimizações, entre outras. Da mesma forma, os compiladores vbc e jsc, possuem inúmeras opções de composição para as compilações.

A próxima seção apresentará alguns aspectos importantes envolvendo o desenvolvimento para aplicativos móveis, utilizando-se a plataforma .NET.

2.11 Dispositivos móveis baseados na plataforma Pocket PC

Um dos principais tipos de dispositivos móveis são os PDA's, que, apesar de se utilizarem de comunicações sem fio, não necessitam de conexões contínuas para que possa ser utilizado, como é o caso da tecnologia WAP, utilizada pelos telefones celulares (MSDN, 2003G).

A Microsoft disponibiliza uma plataforma operacional para PDA's, chamada *Pocket PC*, que se encontra na versão 2003 e utiliza a plataforma .NET Compact, que é um subconjunto da plataforma .NET para aplicações *desktop*. Isto permite fácil transição para a programação para *Pocket PC* (MSDN, 2003G). A Microsoft oferece ainda, a ferramenta Visual Studio .NET 2003, que possibilita a criação de aplicações para *Pocket PC*, além de oferecer um emulador, que permite a realização de testes, e que foi utilizado na implementação do trabalho apresentado neste relatório. A tela principal do emulador é vista na figura 32.



Figura 32: Tela principal do emulador de *Pocket PC*

Quando se pede para executar uma aplicação para *Pocket PC*, utilizando-se a ferramenta Visual Studio .NET 2003, o emulador é automaticamente carregado com a aplicação desenvolvida, sem a necessidade de um dispositivo real para a realização de testes.

Até mesmo pelo fato de constituir um subconjunto da plataforma .NET, a plataforma .NET Compact permite o desenvolvimento de aplicações que utilizam a tecnologia de *Web Services* para a comunicação. Este é um fator importante, já que, como existe uma certa limitação de recursos computacionais dos PDA's, motivada até mesmo pelo seus tamanhos, parte do processamento passa a ser função de servidores remotos.

Este capítulo apresentou os itens teóricos necessários para a realização do trabalho, cujos detalhes da implementação serão especificados no capítulo 4, após uma apresentação do material e métodos utilizados para a concretização deste trabalho.

3 MATERIAL E MÉTODOS

Neste capítulo, serão apresentados os detalhes referentes aos materiais e metodologia utilizados na implementação da aplicação descrita no primeiro capítulo, além da utilizada no processo de desenvolvimento desta monografia.

3.1 Local e Período

O trabalho foi desenvolvido no NDS (Núcleo de Desenvolvimento de Software) e no Labin 5 (Laboratório de Informática número 5) do Curso de Sistemas de Informação, no Centro Universitário Luterano de Palmas. Os trabalhos tiveram início no mês de agosto de 2003 e término em dezembro de 2003.

3.2 Material

O material utilizado podem ser divididos em três categorias: hardware, software e fontes bibliográficas. A primeira, é constituída por dois computadores: o primeiro, com processador Intel Pentium IV com clock de 1,7 GHz, 256 Mb de memória RAM e HD com capacidade para 40 Gb, localizado no Labin 5; enquanto que o segundo, possuem processador Intel Pentium III com clock de 750 MHz, 128 Mb de memória RAM e HD com capacidade para 20 Gb. O primeiro ficou responsável pelo armazenamento dos dados das notícias cadastradas, e pela criação e execução da aplicação consumidora do *Web*

Service, que, por sua vez, estava localizado no segundo computador, sob a mesma rede do primeiro.

Os softwares utilizados foram os seguintes:

- Microsoft Word, para a redação do relatório.
- Microsoft Access, para a realização de testes preliminares com *Web Services*
- Microsoft SQL *Server*, para o armazenamento das notícias.
- Microsoft .NET Framework 1.1, para execução de testes sobre os *Web Services*.
- Microsoft .NET Framework SDK (English) 1.1, para referência e utilização das classes providas pela plataforma .NET.
- Microsoft Visual Studio .NET 2003, para construção da aplicação para dispositivos móveis, além da execução de testes utilizando-se o emulador por ele provido.
- Adobe Acrobat Reader, para a leitura de referências bibliográficas.

3.3 Metodologia

A implementação pode ser dividida na criação do banco de dados de notícias, na implementação do *Web Service* e da aplicação final para PDA's.

A princípio o banco de dados foi criado em Microsoft Access, sobre o qual foram feitos testes iniciais sobre alguns *Web Services* experimentais. Posteriormente, os dados foram transferidos para o Microsoft SQL *Server*, utilizado pelo *Web Service* final.

Sobre a implementação do *Web Service*, foram realizados alguns testes iniciais, sem envolver acesso a bancos de dados. Em seguida, foram criados alguns *Web Services* para o acesso aos dados residentes no banco criado em Microsoft Access. E, por fim, foi criado o *Web Service* final, que foi utilizado pela aplicação móvel.

Quanto à aplicação para PDA's, sua implementação só se iniciou após uma primeira versão do *Web Service* final ter sido criado. Porém, no momento da criação da aplicação final, foram percebidas algumas necessidades de adaptações no *Web Service*, o que era imediatamente realizado.

4 RESULTADOS E DISCUSSÃO

Como parte integrante do trabalho, foi implementado um software para funcionar em PDA's que utilizem a plataforma *Pocket PC*. Sua função é apresentar notícias gerais, dos cursos e da pastoral do CEULP/ULBRA. Estas notícias residem em uma base relacional *SQL Server*, e são acessadas pelo *Web Service* desenvolvido (que é apresentado na seção 4.4), que encontra-se em uma outra máquina, na mesma rede local. Assim, o princípio do funcionamento da aplicação móvel é a apresentação dos resultados da invocação dos métodos do *Web Service*.

Resumidamente, este modelo pode ser ilustrado pela figura 33.

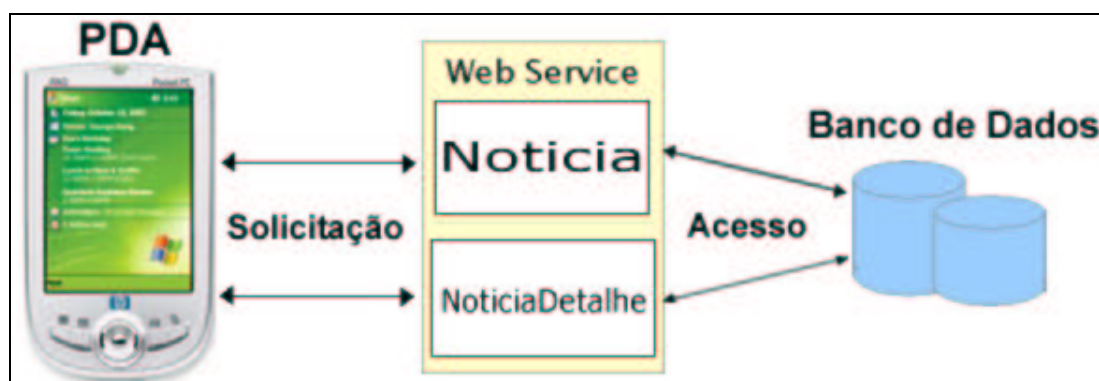


Figura 33: Modelo utilizado na implementação

Mais detalhes dos componentes deste modelo serão apresentados nas seções a seguir.

4.1 Banco de dados de notícias

Para a organização dos dados referentes às notícias, segundo um esquema relacional, foram utilizadas seis tabelas, com todos campos necessários, armazenadas em um banco de dados SQL Server. A representação das tabelas, de seus campos e de seus relacionamentos, gerada pelo próprio SQL Server, é apresentada na figura 34.

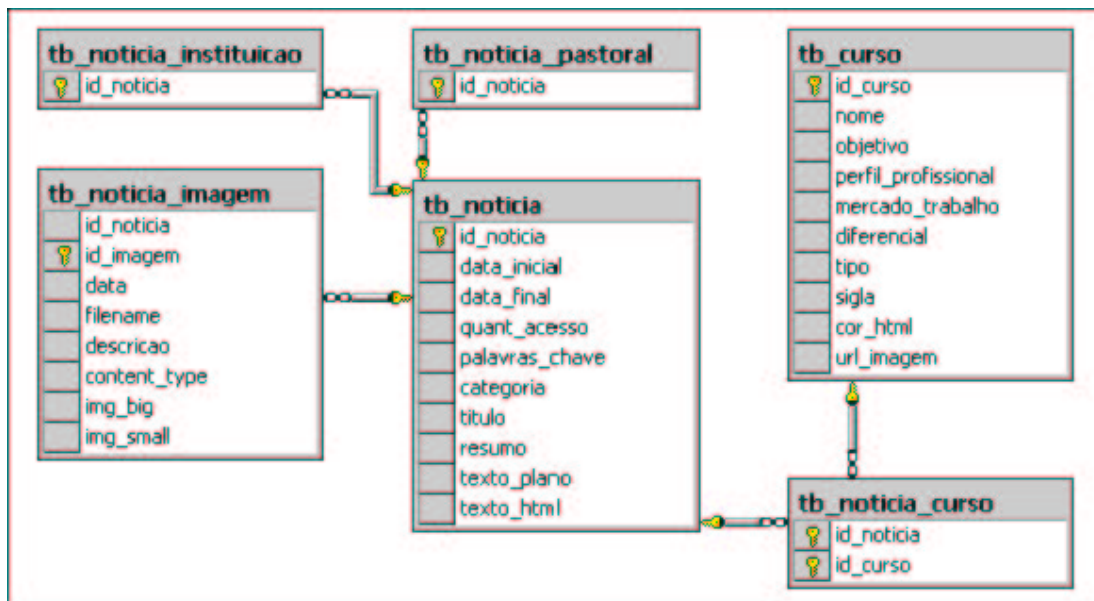


Figura 34: Esquema do banco de dados para armazenamento das notícias

A tabela `tb_noticia` é responsável pelo armazenamento das principais informações referentes às notícias. A tabela `tb_noticia_imagem` armazena as informações referentes às imagens que são apresentadas juntamente com as notícias. Para o relacionamento entre estas duas tabelas utiliza-se, como chave estrangeira, o campo `id_noticia`, da tabela `tb_noticia_imagem`, e, como chave primária, o campo `id_noticia`, da tabela `tb_noticia`.

Para a identificação da categoria à qual pertencem as notícias, existem três tabelas auxiliares, `tb_noticia_instituicao`, `tb_noticia_pastoral` e `tb_noticia_curso`, que possuem campos `id_noticia`, que são chave estrangeira em relação à chave primária `id_noticia`, da tabela `tb_noticia`.

Existe ainda, a tabela `tb_curso`, que serve para armazenar informações sobre os cursos representados apenas pelo campo `id_curso`, na tabela `tb_noticia_curso`.

4.2 Classe para acesso ao banco de dados

Como forma de facilitar o acesso aos dados armazenados no banco de dados, foi criada uma classe que representa os atributos e métodos necessários para acesso aos dados. Assim, não é necessário que, para cada WebMethod, sejam definidos objetos e demais instruções para que os dados sejam efetivamente consultados.

Esta classe, assim como todos os demais artefatos definidos nesta arquitetura, foi definida utilizando-se a linguagem C#. A classe, chamada Banco, é apresentada na figura 35.

```

1  using System;
2  using System.Data;
3  using System.Data.SqlClient;
4
5  public class Banco {
6      private String strConexao =
7  "Server=LABIN501;DataBase=Noticias;uid=michael;pwd=tcc";
8      private SqlConnection conexao;
9      private SqlCommand comando;
10     public SqlDataReader selectSQL(String strSelect) {
11         try {
12             conexao = new SqlConnection(strConexao);
13             comando = new SqlCommand(strSelect, conexao);
14             comando.Connection.Open();
15             return comando.ExecuteReader();
16         }
17         catch(SqlException) {
18             return null;
19         }
20     }
21     public void fecharConexao() {
22         comando.Connection.Close();
23     }
24 }

```

Figura 35: Classe Banco, usada para acesso ao banco de dados

Entre as linhas 1 e 3 são declaradas as *namespaces* que contêm as classes que serão utilizadas. A classe Banco possui um método chamado selectSQL, para o qual é passado como parâmetro, um objeto da classe String, e que tem a função de retornar um objeto da classe SqlDataReader. Para isto, são utilizados três objetos privados, declarados entre as

linhas 6 e 8, sendo que os dois primeiros, `strConexao` e `conexao`, são utilizados para o estabelecimento da conexão com o banco, e o terceiro, chamado `comando`, da classe `SqlCommand`, representará a execução SQL da *string* de consulta passada como parâmetro, sobre a conexão estabelecida previamente.

Entre as linhas 10 e 18 existe um bloco `try-catch`, que funciona como mecanismo para tratamento de exceções. Assim, se ocorrer algum problema na execução das instruções necessárias para que o método seja executado corretamente, o método retorna `null`, caso contrário, será retornado um objeto da classe `SqlDataReader` que represente os dados da consulta SQL.

Os `WebMethod's` do *Web Service* desenvolvido, no entanto, não retornam simplesmente o resultado da execução do método `selectSQL`: os resultados são adicionados em um *array* de objetos da classe `Noticia` e `NoticiaDetalhe`, que serão apresentadas na próxima seção.

4.3 Classes para representação das notícias

Foram criadas duas classes para representação das notícias: `Noticia` e `NoticiaDetalhe`. A primeira é responsável pela representação dos campos `id_noticia` e `titulo`, da tabela `tb_noticia`, enquanto que a segunda representa todos os dados que serão necessários para a aplicação para dispositivos móveis. Na figura 36 é apresentada a classe `Noticia`.

```

1      public class Noticia {
2          private int _id_noticia;
3          private string _titulo;
4          public Noticia(){ }
5          public Noticia(int Id_noticia, string Titulo){
6              this._id_noticia=Id_noticia;
7              this._titulo=Titulo;
8          }
9          public int id_noticia {
10             get {
11                 return _id_noticia;
12             }
13             set {
14                 _id_noticia = value;
15             }
16         }
17         public string titulo {

```

```

18         get {
19             return _titulo;
20         }
21         set {
22             _titulo = value;
23         }
24     }
25 }

```

Figura 36: Classe Noticia, usada para representar notícias

Nas linhas 2 e 3 são definidos os atributos `_id_noticia` e `_titulo`, que são respectivamente dos tipos inteiro e *string*. Entre as linhas 5 e 8 é definido um método construtor, que tem a função de receber dois valores e atribuí-los aos atributos definidos anteriormente. Entre as linhas 9 e 24 são definidas propriedades, para gravação e leitura dos respectivos atributos.

A existência de um método construtor padrão, definido na linha 4, é de uso obrigatório para que a classe possa ser utilizada pelo *Web Service*.

Da mesma forma que foi definida a classe *Noticia*, a classe *NoticiaDetalhe* segue a mesma estrutura, porém, possuindo os seguintes atributos: `_id_noticia`, do tipo inteiro; `_titulo`, do tipo *string*; `_resumo`, do tipo *string*; `_texto_plano`, do tipo *string*; e `_img_big`, que é um *array* de *bytes*.

Estas duas classes são utilizadas pelo *Web Service* criado, que será visto na seção a seguir.

4.4 Classe para representação do *Web Service*

O *Web Service* criado, cujo nome de classe é *Noticias*, possui quatro *WebMethod*'s:

- **NoticiasDosCursos:** retorna um objeto da classe *ArrayList*, contendo objetos da classe *Noticia*, cujos dados atendem aos seguintes requisitos: de serem notícias de cursos, ou seja, que as notícias retornadas tenham o valor do seu campo `id_noticia` cadastrado na tabela `tb_noticias_cursos`; e que a data atual seja maior ou igual ao campo `data_inicial` e menor ou igual ao campo `data_final` da tabela `tb_noticia`.

- **NoticiasDaPastoral:** retorna um objeto da classe ArrayList, contendo objetos da classe Noticia, cujos dados atendem aos seguintes requisitos: de serem notícias da pastoral, ou seja, que as notícias retornadas tenham o valor do seu campo id_noticia cadastrado na tabela tb_noticias_pastoral; e que a data atual seja maior ou igual ao campo data_inicial e menor ou igual ao campo data_final da tabela tb_noticia.
- **NoticiasDaInstituicao:** retorna um objeto da classe ArrayList, contendo objetos da classe Noticia, cujos dados atendem aos seguintes requisitos: de serem notícias da instituição, ou seja, que as notícias retornadas tenham o valor do seu campo id_noticia cadastrado na tabela tb_noticias_instituicao; e que a data atual seja maior ou igual ao campo data_inicial e menor ou igual ao campo data_final da tabela tb_noticia.
- **NoticiaPorID:** recebe como parâmetro, o valor id_noticia, da notícia cujos detalhes se pretende recuperar, retornando um objeto da classe NoticiaDetalhe.

O método NoticiasDaInstituicao é apresentado na figura 37. Os outros dois métodos que retornam objetos da classe ArrayList, seguem estruturas semelhantes, alterando-se apenas a string SQL para a consulta.

```

1 [WebMethod(Description="Retorna a lista de notícias dos cursos")]
2 [SoapRpcMethod]
3 public ArrayList NoticiasDaInstituicao(){
4     ArrayList lista = new ArrayList();
5     DateTime dt = DateTime.Now;
6     Banco banco = new Banco();
7     SqlDataReader reader = banco.selectSQL("SELECT n.id_noticia, n.titulo,
n.resumo, n.texto_plano, nim.img_big FROM tb_noticia_instituicao ni INNER JOIN
tb_noticia n ON ni.id_noticia = n.id_noticia INNER JOIN tb_noticia_imagem nim ON
n.id_noticia = nim.id_noticia WHERE (n.data_inicial <= '" + dt.Month + "/" +
dt.Day + "/" + dt.Year + "') AND (n.data_final >= '" + dt.Month + "/" + dt.Day +
"/" + dt.Year + "') ORDER BY n.id_noticia DESC");
8     banco.fecharConexao();
9     if(reader != null){
10         while(reader.Read()){
11             noticia = new Noticia(reader.GetInt32(0), reader["titulo"].ToString());
12             lista.Add(noticia);
13         }
14     }
15     return lista;
16 }

```

Figura 37: WebMethod NoticiasDaInstituicao

A primeira linha especifica que o método trata-se de um WebMethod, cuja descrição está explicitada. A linha 2 identifica que esta trata-se de uma chamada remota a um método, informação requerida pela aplicação que utilizará o método. Entre as linhas 4 e 7 são instanciados os objetos que serão utilizados pelo método. Na verdade, estes objetos são atributos privados do *Web Service*, mas foram colocados desta forma no exemplo para melhor visualização. Na linha 7, o resultado da consulta em SQL é armazenado em um objeto da classe *SqlDataReader*. Então, caso o conteúdo deste objeto não seja nulo, seus registros são armazenados em objetos da classe *Noticia*, para posterior adição ao *ArrayList*, processo que é realizado entre as linhas 8 e 13. Por fim, a linha 14 retorna o objeto *lista*, da classe *ArrayList*.

Executando-se o método *NoticiasDaInstituicao* através do navegador de Internet, o resultado é apresentado na figura 38.

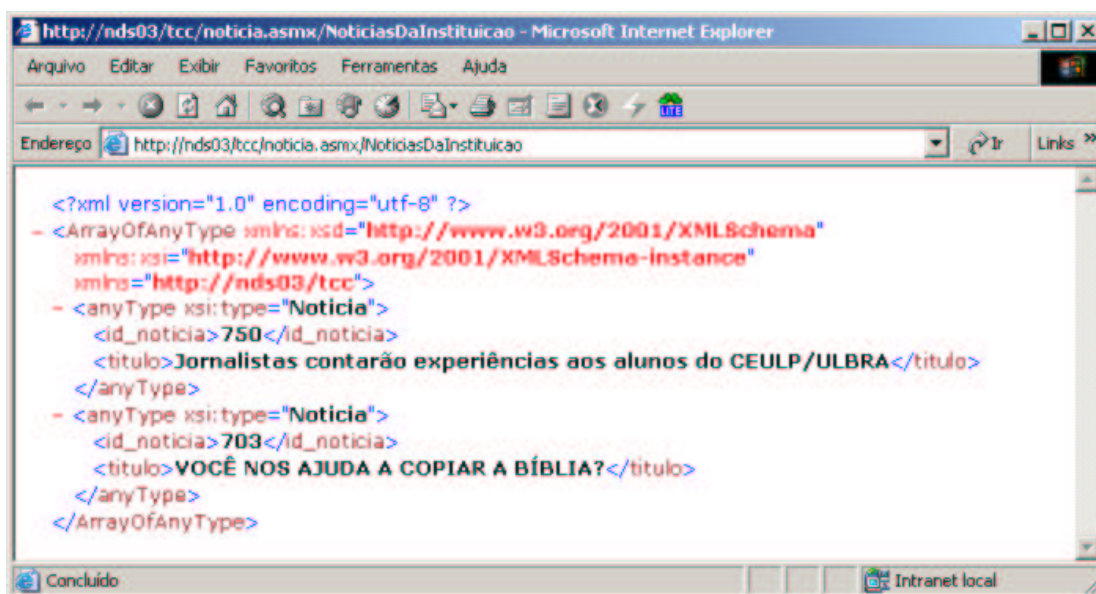


Figura 38: Resultado da execução do método *NoticiasDaInstituicao*

Apesar de vários outros dados serem apresentados na aplicação móvel, apenas dois foram retornados: *id_noticia* e *titulo*. Para os demais dados, é utilizado o método *NoticiaPorID*. Assim, de posse do valor de um campo *id_noticia*, é possível invocar este método para recuperação dos demais dados.

Esta foi a saída encontrada como forma de não carregar vários dados que talvez não sejam necessários, já que, conforme será visto na próxima seção, na aplicação móvel são apresentados, primeiramente, apenas os títulos das notícias. Assim, a partir do momento que o usuário escolher uma notícia para que seus detalhes sejam visualizados, todos os demais dados serão carregados, com a invocação do método *NoticiaPorID*. Em dispositivos com limitações de recursos, como é o caso de PDA's, esta é uma alternativa bastante usual.

Todas as classes criadas no contexto deste trabalho e vistas até aqui foram armazenadas no mesmo arquivo, *Noticia.aspx*, cujo código completo está disposto no Anexo A. A disposição de todas as classes necessárias no mesmo arquivo do *Web Service* é usual, pois elimina a necessidade de compilação manual de cada um dos arquivos que seriam gerados.

Após terem sido vistas todas as partes necessárias ao correto funcionamento do *Web Service*, a próxima seção apresentará detalhes da criação da própria aplicação criada para executar em PDA's.

4.5 A aplicação para dispositivos móveis

A aplicação criada para rodar em PDA's consiste basicamente de dois formulários principais: um para exibição das notícias da categoria escolhida e um para exibição dos detalhes da notícia selecionada na parte anterior.

O primeiro, o formulário principal, é designado pela classe *FormConsumer*, que é composta por um objeto da classe *ListView*, da *namespace* *System.Windows.Forms*, que é utilizado para apresentar os títulos das notícias recuperadas pelos métodos do *Web Service* apresentado anteriormente. Este objeto permite que um clique sobre o título de uma notícia dispare um evento que chame o segundo formulário.

Além disto, existe ainda no primeiro formulário, um menu para a escolha da categoria de notícias e um botão para atualizar as notícias. A tela do emulador que apresenta esta parte ativa é ilustrada na figura 39.



Figura 39: Tela do emulador exibindo os títulos das notícias dos cursos

O menu apresenta os tipos de notícias existentes no banco de dados. Quando o usuário seleciona algum item do menu, é invocado o método do *Web Service* que recupera as informações relativas ao respectivo item. Os títulos das notícias recuperadas são então, carregados no *ListView*.

Como o tipo de retorno dos métodos *NoticiasDaInstituicao*, *NoticiasDosCursos* e *NoticiasDaPastoral* é um *ArrayList* contendo objetos da classe *Noticia*, é necessária a utilização do método *GetEnumerator*, que retorna uma referência à interface *IEnumerator*, da *namespace* *System.Collections*. Esta necessidade deve-se ao fato da aplicação não entender automaticamente que o *array* trata-se de um *ArrayList*. Ou seja, apesar do *WebMethod* retornar um *ArrayList*, este objeto não é reconhecido pela aplicação cliente, pois o *ArrayList* contém um “tipo” customizado, *noticia*.

O carregamento dos títulos das notícias no *ListView* é feito por meio de um método chamado *preencherLV*, existente na classe *FormConsumer*. Ele recebe valores inteiros que indicam a categoria de notícias e retorna um valor booleano indicando se existem itens no *ListView*. Este método é apresentado na figura 40.

```

1 private bool preencherLV(int iTipo) {
2     System.Collections.IEnumerator enumerator = null;
3     ArrayList listaDeNoticias = new ArrayList();
4     bool bReturn = false;
5     lv.Items.Clear();
6     if (iTipo == 1) {
7         enumerator = wsNoticias.NoticiasDaInstituicao().GetEnumerator();
8         while ((enumerator.MoveNext() && (enumerator.Current != null))
9             listaDeNoticias.Add(enumerator.Current);
10    }
11    else if (iTipo == 2) {
12        enumerator = wsNoticias.NoticiasDosCursos().GetEnumerator();
13        while ((enumerator.MoveNext() && (enumerator.Current != null))
14            listaDeNoticias.Add(enumerator.Current);
15    }
16    else {
17        enumerator = wsNoticias.NoticiasDaPastoral().GetEnumerator();
18        while ((enumerator.MoveNext() && (enumerator.Current != null))
19            listaDeNoticias.Add(enumerator.Current);
20    }
21    if (wsNoticias != null) {
22        if (listaDeNoticias != null) {
23            for (int i=0; i<listaDeNoticias.Count; i++) {
24                Noticia = (nds03.tcc.Noticia)listaDeNoticias[i];
25                ListViewItem lvi = new
26                ListViewItem(noticia.titulo.ToString());
27                lvi.SubItems.Add(noticia.id_noticia.ToString());
28                lv.Items.Add(lvi);
29            }
30            if (lv.Items.Count > 0) {
31                bReturn = true;
32            }
33        }
34    }
35    return bReturn;
36 }

```

Figura 40: Método preencherLV, da classe FormConsumer

Nota-se, entre as linhas 5 e 19, que os métodos do *Web Service* são invocados dependendo do valor passado como parâmetro. Assim, para qualquer que seja o valor passado como parâmetro, é utilizado o método `GetEnumerator` para que o resultado da execução do `WebMethod` correspondente seja devidamente adicionado em um objeto da classe `ArrayList`. Em seguida, se o `ArrayList` contendo as notícias não for nulo, seus itens são atribuídos a um objeto da classe `Noticia`, para que possam ser adicionados ao `ListView`, o que é feito entre as linhas 22 e 27.

A classe `FormConsumer` é apresentada na íntegra no Anexo B.

O segundo formulário é representado pela classe FormDetalhes, apresentando os detalhes da notícia escolhida. Ele é composto por objetos da classe TextBox, da *namespace* System.Windows.Forms, que são utilizados para a apresentação de dados de texto. Este formulário contém ainda, um objeto da classe PictureBox, da *namespace* System.Windows.Forms, utilizado para a apresentação da imagem referente à notícia escolhida. A tela do emulador apresentando este formulário é apresentada na figura 41.



Figura 41: Tela do emulador exibindo os detalhes de uma notícia

Quando um usuário clica sobre o título de uma notícia, no formulário principal, é invocado o método da classe FormDetalhes, setNoticia, responsável por carregar os detalhes da notícia escolhida, através do uso do WebMethod NoticiaPorID. Este método é apresentado na figura 42.

```

1 public void setNoticia(string id_noticia) {
2     NoticiaDetalle noticia = wsNoticias.NoticiaPorID(id_noticia);
3     If(noticia != null) {
4         lbTitulo.Text = noticia.titulo;
5         tbResumo.Text = noticia.resumo;
6         tbTextoPlano.Text = noticia.texto_plano;
7         byte[] image = noticia.img_big;
8         System.IO.MemoryStream memStream = new

```

```

9      System.IO.MemoryStream(image);
10         If (memStream.Length != 0) {
11             try {
12                 Bitmap bm = new Bitmap(memStream);
13                 pbImagem.Image = bm;
14             }
15             catch(Exception) {
16                 MessageBox.Show("Ocorreram problemas com o
formato da imagem.", "Erro");
17             }
18         } else {
19             MessageBox.Show("Não foi possível apresentar a
imagem","Alerta");
20         }
21     }
22 }

```

Figura 42: Método setNoticia, da classe FormDetalhes

A segunda linha apresenta a execução do WebMethod NoticiaPorID, que passa o valor do campo id_noticia recebido pelo método setNoticia, da classe FormDetalhes. O resultado desta execução é armazenado em um objeto noticia, da classe NoticiaDetalhe. Em seguida, se este objeto não for nulo, as linhas de 4 a 6 atribuem aos objetos de texto, os valores de texto plano do objeto noticia.

Na sétima linha, é atribuído a um *array* de *bytes*, os dados referentes à imagem que representa a notícia em questão. Para que possa realmente ser apresentado como imagem, é necessário o procedimento localizado entre as linhas 8 e 20. O *array* de *bytes* é transformado em um objeto memStream, da classe MemoryStream, contido pela *namespace* System.IO.

Caso o tamanho do objeto memStream não seja vazio, ele é convertido para um objeto da classe Bitmap, para ser, logo em seguida, atribuído ao objeto que irá apresentar a imagem no formulário. Caso contrário, é exibida uma mensagem informando que não foi possível apresentar a imagem.

Após ter sido apresentada a arquitetura desenvolvida, bem como o embasamento teórico para sua realização, o próximo capítulo apresentará algumas conclusões e considerações obtidos a partir do trabalho.

5 CONSIDERAÇÕES FINAIS

A realização deste trabalho permitiu demonstrar as principais características de tecnologias que estão começando a ser utilizadas em larga escala: *Web Services* e dispositivos móveis. Ainda mais importante que apenas demonstrar seus usos, foi combiná-los, visto que a aplicação desenvolvida apóia-se em um modelo que permite seu funcionamento. Ou seja, além da aplicação desenvolvida ser realmente utilizável, ela utilizou-se de um modelo que pode ser facilmente modificado a fim de produzir aplicações com outras finalidades. Isto pode ser afirmado pelo fato desta aplicação ser composta de três componentes básicos para soluções desta natureza, que, neste caso, encontram-se em lugares diferentes em uma rede: banco de dados, *Web Service*, e aplicação final.

Assim, pode-se dizer que a concretização de conceitos encontrados separadamente na literatura permitiu a verificação da viabilidade da utilização de *Web Services*, de aplicações para PDA's que utilizam a plataforma operacional *Pocket PC*, quanto de ambos interagindo entre si. Alguns itens importantes para esta conclusão são facilidade de implementação e desempenho.

O desenvolvimento do código de *Web Services* não apresenta grandes dificuldades, sendo bastante semelhante ao desenvolvimento de classes tradicionais. Sobre o desempenho de seu funcionamento, pode-se dizer que não seja o mesmo de comunicações entre uma aplicação e um banco de dados, já que os dados trocados são baseados em XML, sendo necessário um processo de *parsing* dos dados.

Sobre a implementação de aplicativos para PDA's, ela pode ser considerada simples, já que a plataforma .NET Compact é, na verdade, um subconjunto da plataforma .NET, além de que ferramentas como Visual Studio .NET 2003 simplificam sua construção. Quanto à questão de desempenho, é necessário um cuidado na construção destes aplicativos, dada a escassez de recursos dos dispositivos móveis.

A questão de comunicação em redes sem fio dos dispositivos móveis, não foi abordada neste trabalho, visto que o trabalho foi focado principalmente no desenvolvimento das aplicações.

Tendo este trabalho apresentado as principais características das tecnologias de *Web Services* e de aplicativos móveis, ambos trabalhando sob a plataforma .NET, possíveis trabalhos futuros podem basear-se na construção de *Web Services* de diferentes plataformas para serem acessados a partir de aplicações móveis construídas em várias plataformas. Desta forma, tem-se um exemplo prático de uma das principais características da teoria de *Web Services*: a interoperabilidade entre plataformas diferentes. Além disto, outros domínios poderiam ser atendidos em trabalhos futuros, tal como *Web Services* para inserção de informações provenientes de outros tipos de dispositivos (sensores, leitores de digitais), em servidores remotos, por exemplo.

REFERÊNCIAS BIBLIOGRÁFICAS

- (AMUNDSEN, 2002) AMUNDSEN, Michael; LITWIN, Paul. **ASP.NET para desenvolvedores de Web sites**. Rio de Janeiro: Editora Ciência Moderna Ltda., 2002.
- (ANDERSON, 2001) ANDERSON, R. Et al. **Professional XML**. Rio de Janeiro: Ciência Moderna LTDA., 2001.
- (BALLINGER, 2003) BALLINGER, Keith. **.NET Web Services: Architecture and Implementation**. Boston: Addison-Wesley, 2003.
- (JORGENSEN, 2002) JORGENSEN, David. **Developing .Net Web Services with XML**. Rockland: Syngress Publishing, Inc., 2002.
- (MSDN, 2003B) **SOAP Specification Index Page**. MSDN Library. Disponível em <<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsoapspec/html/soapspecindex.asp>>. Acesso em: 15/11/2003.
- (MSDN, 2003D) **Introducing XML Serialization**. MSDN Library. Disponível em <<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconintroducingxmlserialization.asp>>. Acesso em: 15/11/2003.
- (MSDN, 2003E) **WSDL Specification Index Page**. MSDN Library. Disponível em <<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwsdl/html/wsdlspecindex.asp>>. Acesso em: 23/11/2003.
- (MSDN, 2003G) **.NET Compact Framework Overview**. MSDN Library. Disponível em <<http://msdn.microsoft.com/vstudio/device/compactfx.aspx>>. Acesso em: 29/11/2003.

- (NAMES, 2003) **Namespaces in XML 1.1.** World Wide Web Consortium, nov. 2003. Disponível em <<http://www.w3.org/TR/xml-names11/>>. Acesso em 15/11/2003.
- (PAYNE, 2001) PAYNE, Chris. **Aprenda em 21 dias ASP.NET.** Rio de Janeiro: Campus, 2001.
- (PINTO, 2003) PINTO, Marcus Barbosa; SACCOL, Deise de Brum. **Um estudo sobre esquemas para documentos XML.** In: V Encontro de Estudantes de Informática do Tocantins. Palmas: ANAIS ENCOINFO/EIN, 2003. p. 211 – 220.
- (SIDDIQUI, 2001) SIDDIQUI, Bilal. **Deploying Web Services with WSDL: Part 1.** IBM developerWorks, nov. 2001. Disponível em <<http://www-106.ibm.com/developerworks/library/ws-intwsdl/>>. Acesso em: 23/11/2003.
- (SOAP, 2000) **SimpleObject Acces Protocol (SOAP) 1.1.** World Wide Web Consortium, maio 2000. Disponível em <<http://www.w3.org/TR/soap/>>. Acesso em 15/11/2003.
- (SOAP, 2003) **SOAP Version 1.2 Part 0: Primer.** World Wide Web Consortium, jun. 2003. Disponível em <<http://www.w3.org/TR/2003/REC-soap12-part0-20030624/>>. Acesso em 15/11/2003.
- (SOARES, 1995) SOARES, Luiz Fernando Gomes. **Redes de computadores: das LANs, MANs e WANs às redes ATM.** Rio de Janeiro: Campus, 1995.
- (TECHNET, 2003) **Mobile Enterprise Solutions: What Is the Appropriate Pocket-Size Platform?.** Microsoft TechNet. Disponível em <<http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/mobile/evaluate/mobilwhy.asp?frame=true>>. Acesso em 29/11/2003.
- (UDDI, 2002) **UDDI Programmer's API 1.0.** UDDI.org, jun. 2002. Disponível em <<http://uddi.org/pubs/ProgrammersAPI-V1.01-Published-20020628.pdf>>. Acesso em 26/11/2003.

- (UDDI, 2003) **UDDI.org.** UDDI.org. Disponível em <<http://www.uddi.org>>. Acesso em 26/11/2003.
- (WSA, 2003) **Web Services Architecture.** World Wide Web Consortium, ago. 2003. Disponível em <<http://www.w3.org/TR/2003/WD-ws-arch-20030808/>>. Acesso em 09/11/2003.
- (WSDL, 2001) **Web Services Description Language (WSDL) Version 1.1.** World Wide Web Consortium, mar. 2001. Disponível em <<http://www.w3.org/TR/wsdl>>. Acesso em 15/11/2003.
- (WSDL, 2003) **Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language.** World Wide Web Consortium, nov. 2003. Disponível em <<http://www.w3.org/TR/wsdl20/>>. Acesso em 15/11/2003.
- (XML, 2003) **Extensible Markup Language (XML).** World Wide Web Consortium. Disponível em <<http://www.w3.org/XML/>>. Acesso em 10/04/2003.
- (XSD, 2001A) **XML Schema Part 0: Primer.** World Wide Web Consortium, maio 2001. Disponível em <<http://www.w3.org/TR/xmlschema-0/>>. Acesso em 15/11/2003.
- (XSD, 2001B) **XML Schema Part 2: Datatypes.** World Wide Web Consortium, maio 2001. Disponível em <<http://www.w3.org/TR/xmlschema-2/>>. Acesso em 15/11/2003.

ANEXO A

Código do Web Service criado – Noticia.asmx

```

<%@ WebService Language="c#" Class="Noticias" %>

using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Diagnostics;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml.Serialization;

[WebService(Namespace="http://nds03/tcc")]
[SoapInclude(typeof(Noticia))]
[XmlInclude(typeof(Noticia))]
    /// <summary>
    /// Web Service para acesso a notícias do Portal CEULP/ULBRA.
    /// </summary>
    public class Noticias : System.Web.Services.WebService
    {

        private ArrayList _lista;
        private DateTime _dt;
        private Banco _banco;
        private Noticia _noticia;
        private SqlDataReader _reader;

        public ArrayList lista {
            get {
                return _lista;
            }
            set {
                _lista = value;
            }
        }

        public DateTime dt {
            get {
                return _dt;
            }
            set {
                _dt = value;
            }
        }

        public Banco banco {
            get {
                return _banco;
            }
            set {
                _banco = value;
            }
        }

        public Noticia noticia {

```

```

        get {
            return _noticia;
        }
        set {
            _noticia = value;
        }
    }

    public SqlDataReader reader {
        get {
            return _reader;
        }
        set {
            _reader = value;
        }
    }

    [WebMethod(Description="Retorna a lista de notícias dos cursos")]
    [SoapRpcMethod]
    public ArrayList NoticiasDosCursos(){
        lista = new ArrayList();
        dt = DateTime.Now;
        banco = new Banco();
        reader = banco.selectSQL("SELECT n.id_noticia, n.titulo,
n.resumo, n.texto_plano, nim.img_big FROM tb_noticia_curso nc INNER JOIN
tb_noticia n ON nc.id_noticia = n.id_noticia INNER JOIN tb_noticia_imagem nim ON
n.id_noticia = nim.id_noticia WHERE (n.data_inicial <= '" + dt.Month + "/" + dt.Day +
"/" + dt.Year + "') AND (n.data_final >= '" + dt.Month + "/" + dt.Day + "/" + dt.Year
+ "') ORDER BY n.id_noticia DESC");
        banco.fecharConexao();
        if(reader != null){
            while(reader.Read()){
                noticia = new Noticia(reader.GetInt32(0),
reader["titulo"].ToString());
                lista.Add(noticia);
            }
        }
        return lista;
    }

    [WebMethod(Description="Retorna a lista de notícias da Pastoral")]
    [SoapRpcMethod]
    public ArrayList NoticiasDaPastoral(){
        lista = new ArrayList();
        dt = DateTime.Now;
        banco = new Banco();
        reader = banco.selectSQL("SELECT n.id_noticia, n.titulo,
n.resumo, n.texto_plano, nim.img_big FROM tb_noticia_pastoral np INNER JOIN
tb_noticia n ON np.id_noticia = n.id_noticia INNER JOIN tb_noticia_imagem nim ON
n.id_noticia = nim.id_noticia WHERE (n.data_inicial <= '" + dt.Month + "/" + dt.Day +
"/" + dt.Year + "') AND (n.data_final >= '" + dt.Month + "/" + dt.Day + "/" + dt.Year
+ "') ORDER BY n.id_noticia DESC");
        banco.fecharConexao();
        if(reader != null){
            while(reader.Read()){
                noticia = new Noticia(reader.GetInt32(0),
reader["titulo"].ToString());

```

```

        lista.Add(noticia);
    }
}
return lista;
}

[WebMethod(Description="Retorna a lista de notícias da Instituição")]
[SoapRpcMethod]
public ArrayList NoticiasDaInstituicao(){
    lista = new ArrayList();
    dt = DateTime.Now;
    banco = new Banco();
    reader = banco.selectSQL("SELECT n.id_noticia, n.titulo,
n.resumo, n.texto_plano, nim.img_big FROM tb_noticia_instituicao ni INNER JOIN
tb_noticia n ON ni.id_noticia = n.id_noticia INNER JOIN tb_noticia_imagem nim ON
n.id_noticia = nim.id_noticia WHERE (n.data_inicial <= '' + dt.Month + '/' + dt.Day +
 '/' + dt.Year + '') AND (n.data_final >= '' + dt.Month + '/' + dt.Day + '/' + dt.Year
+ '') ORDER BY n.id_noticia DESC");
    banco.fecharConexao();
    if(reader != null){
        while(reader.Read()){
            noticia = new Noticia(reader.GetInt32(0),
reader["titulo"].ToString());
            lista.Add(noticia);
        }
    }
    return lista;
}

[WebMethod(Description="Retorna um objeto da Classe
NoticiaDetalhe")]
public NoticiaDetalhe NoticiaPorID(string id_noticia){
    Banco banco = new Banco();
    NoticiaDetalhe noticia = null;
    reader = banco.selectSQL("SELECT n.id_noticia, n.titulo,
n.resumo, n.texto_plano, nim.img_big FROM tb_noticia n INNER JOIN
tb_noticia_imagem nim ON n.id_noticia = nim.id_noticia WHERE n.id_noticia ='' +
id_noticia +''");
    banco.fecharConexao();
    if (reader!=null){
        while(reader.Read()){
            noticia = new NoticiaDetalhe(reader.GetInt32(0),
reader["titulo"].ToString(), reader["resumo"].ToString(),
reader["texto_plano"].ToString(), (byte[])reader["img_big"]);
        }
        return noticia;
    } else {
        return null;
    }
}

}

public class Noticia {

    private int _id_noticia;
    private string _titulo;

```

```

public Noticia(){ }

public Noticia(int Id_noticia, string Titulo){
    this._id_noticia=Id_noticia;
    this._titulo=Titulo;
}

public int id_noticia {
    get {
        return _id_noticia;
    }
    set {
        _id_noticia = value;
    }
}

public string titulo {
    get {
        return _titulo;
    }
    set {
        _titulo = value;
    }
}
}

public class NoticiaDetalhe {

    private int _id_noticia;
    private string _titulo;
    private String _resumo;
    private string _texto_plano;
    private byte[] _img_big;

    public NoticiaDetalhe(){ }

    public NoticiaDetalhe(int Id_noticia, string Titulo, string Resumo, string
Texto_plano, byte[] Img_big){
        this._id_noticia=Id_noticia;
        this._titulo=Titulo;
        this._resumo=Resumo;
        this._texto_plano=Texto_plano;
        this._img_big=Img_big;
    }

    public int id_noticia {
        get {
            return _id_noticia;
        }
        set {
            _id_noticia = value;
        }
    }

    public string titulo {
        get {
            return _titulo;
        }
        set {

```

```

        _titulo = value;
    }
}

public string resumo {
    get {
        return _resumo;
    }
    set {
        _resumo = value;
    }
}

public string texto_plano {
    get {
        return _texto_plano;
    }
    set {
        _texto_plano = value;
    }
}

public byte[] img_big {
    get {
        return _img_big;
    }
    set {
        _img_big = value;
    }
}
}

public class Banco {

    private String strConexao =
"Server=LABIN501;DataBase=Noticias;uid=michael;pwd=tcc";
    private SqlConnection conexao;
    private SqlCommand comando;

    public SqlDataReader selectSQL(String strSelect) {
        try {
            conexao = new SqlConnection(strConexao);
            comando = new SqlCommand(strSelect, conexao);
            comando.Connection.Open();
            return comando.ExecuteReader();
        }
        catch (SQLException) {
            return null;
        }
    }

    public void fecharConexao() {
        comando.Connection.Close();
    }
}
}

```

ANEXO B

Código da aplicação para PDA's com a plataforma *Pocket PC*

FormConsumer.cs e FormDetalhes.cs

FormConsumer.cs:

```

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.IO;

namespace NamespaceTeste
{
    /// <summary>
    /// Summary description for FormConsumer
    /// </summary>
    public class FormConsumer : System.Windows.Forms.Form
    {
        private System.Windows.Forms.ColumnHeader columnHeader1;
        private System.Windows.Forms.ListView lv;
        private System.Windows.Forms.MainMenu mainMenu;
        private System.Windows.Forms.Label tbQdNoticias;
        private System.Windows.Forms.ColumnHeader columnHeader2;
        private System.Windows.Forms.ColumnHeader columnHeader3;
        private System.Windows.Forms.MenuItem mTipos;
        private System.Windows.Forms.MenuItem miInstituicao;
        private System.Windows.Forms.MenuItem miCursos;
        private System.Windows.Forms.MenuItem miPastoral;
        private System.Windows.Forms.Label lbTipo;
        private System.Windows.Forms.ToolBar toolBar1;
        private System.Windows.Forms.Button btAtualizar;
        private nds03.tcc.Noticias wsNoticias = null;
        private nds03.tcc.Noticia noticia;
        private ArrayList listaDeNoticias;

        public FormConsumer()
        {
            //
            // Required for Windows Form Designer support
            //
            InitializeComponent();
            wsNoticias = new nds03.tcc.Noticias();
            listaDeNoticias = new ArrayList();
            preencherLV(1);
            lbTipo.Text = "Notícias da Instituição";

            //
            // TODO: Add any constructor code after InitializeComponent call
            //
        }
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            base.Dispose( disposing );
        }
        #region Windows Form Designer generated code
        /// <summary>

```

```

/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.lv = new System.Windows.Forms.ListView();
    this.columnHeader1 = new
System.Windows.Forms.ColumnHeader();
    this.columnHeader2 = new
System.Windows.Forms.ColumnHeader();
    this.columnHeader3 = new
System.Windows.Forms.ColumnHeader();
    this.mainMenu = new System.Windows.Forms.MainMenu();
    this.mTipos = new System.Windows.Forms.MenuItem();
    this.miInstituicao = new System.Windows.Forms.MenuItem();
    this.miPastoral = new System.Windows.Forms.MenuItem();
    this.miCursos = new System.Windows.Forms.MenuItem();
    this.tbQdNoticias = new System.Windows.Forms.Label();
    this.lbTipo = new System.Windows.Forms.Label();
    this.toolBar1 = new System.Windows.Forms.ToolBar();
    this.btAtualizar = new System.Windows.Forms.Button();
    //
    // lv
    //
    this.lv.Columns.Add(this.columnHeader1);
    this.lv.Columns.Add(this.columnHeader2);
    this.lv.Columns.Add(this.columnHeader3);
    this.lv.FullRowSelect = true;
    this.lv.Location = new System.Drawing.Point(8, 40);
    this.lv.Size = new System.Drawing.Size(224, 192);
    this.lv.View = System.Windows.Forms.View.List;
    this.lv.GotFocus += new
System.EventHandler(this.lv_GotFocus);
    //
    // columnHeader1
    //
    this.columnHeader1.Text = "Noticia";
    this.columnHeader1.Width = 222;
    //
    // columnHeader2
    //
    this.columnHeader2.Text = "Resumo";
    this.columnHeader2.Width = 0;
    //
    // columnHeader3
    //
    this.columnHeader3.Text = "texto_plano";
    this.columnHeader3.Width = 0;
    //
    // mainMenu
    //
    this.mainMenu.MenuItems.Add(this.mTipos);
    //
    // mTipos
    //
    this.mTipos.MenuItems.Add(this.miInstituicao);
    this.mTipos.MenuItems.Add(this.miPastoral);

```

```

        this.mTipos.MenuItems.Add(this.miCursos);
        this.mTipos.Text = "Tipos";
        //
        // miInstituicao
        //
        this.miInstituicao.Text = "Instituição";
        this.miInstituicao.Click += new
System.EventHandler(this.carregar);
        //
        // miPastoral
        //
        this.miPastoral.Text = "Pastoral";
        this.miPastoral.Click += new
System.EventHandler(this.carregar);
        //
        // miCursos
        //
        this.miCursos.Text = "Cursos";
        this.miCursos.Click += new System.EventHandler(this.carregar);
        //
        // tbQdNoticias
        //
        this.tbQdNoticias.ForeColor =
System.Drawing.SystemColors.ControlText;
        this.tbQdNoticias.Location = new System.Drawing.Point(8, 24);
        this.tbQdNoticias.Size = new System.Drawing.Size(112, 16);
        this.tbQdNoticias.Text = "Clique na manchete";
        //
        // lbTipo
        //
        this.lbTipo.Font = new System.Drawing.Font("Microsoft Sans
Serif", 9F, System.Drawing.FontStyle.Bold);
        this.lbTipo.ForeColor = System.Drawing.Color.MidnightBlue;
        this.lbTipo.Location = new System.Drawing.Point(8, 4);
        this.lbTipo.Size = new System.Drawing.Size(160, 20);
        this.lbTipo.Text = "Notícias";
        //
        // btAtualizar
        //
        this.btAtualizar.Location = new System.Drawing.Point(84, 240);
        this.btAtualizar.Text = "Atualizar";
        this.btAtualizar.Click += new
System.EventHandler(this.atualizar);
        //
        // FormConsumer
        //
        this.BackColor = System.Drawing.Color.WhiteSmoke;
        this.Controls.Add(this.btAtualizar);
        this.Controls.Add(this.lbTipo);
        this.Controls.Add(this.tbQdNoticias);
        this.Controls.Add(this.lv);
        this.Controls.Add(this.toolBar1);
        this.Menu = this.mainMenu;
        this.Text = "CEULP Notícias";
    }
#endregion

```

```

/// <summary>
/// The main entry point for the application.
/// </summary>

static void Main()
{
    Application.Run(new FormConsumer());
}

private bool preencherLV(int iTipo)
{
    System.Collections.IEnumerator enumerator = null;
    bool bReturn = false;

    lv.Items.Clear();

    if (iTipo == 1)
    {
        enumerator =
wsNoticias.NoticiasDaInstituicao().GetEnumerator();
        while (( enumerator.MoveNext() ) && (
enumerator.Current != null ))
            listaDeNoticias.Add(enumerator.Current);
    }
    else if(iTipo == 2)
    {
        enumerator =
wsNoticias.NoticiasDosCursos().GetEnumerator();
        while (( enumerator.MoveNext() ) && (
enumerator.Current != null ))
            listaDeNoticias.Add(enumerator.Current);
    }
    else
    {
        enumerator =
wsNoticias.NoticiasDaPastoral().GetEnumerator();
        while (( enumerator.MoveNext() ) && (
enumerator.Current != null ))
            listaDeNoticias.Add(enumerator.Current);
    }

    if (wsNoticias != null)
    {
        if (listaDeNoticias != null)
        {
            for (int i=0; i<listaDeNoticias.Count; i++)
            {
                noticia =
(nds03.tcc.Noticia)listaDeNoticias[i];
                ListViewItem lvi = new
ListViewItem(noticia.titulo.ToString());
                lvi.SubItems.Add(noticia.id_noticia.ToString());
                lv.Items.Add(lvi);
            }
        }
    }
}

```

```

        if (lv.Items.Count > 0)
        {
            bReturn = true;
        }
    }
    return bReturn;
}

private void atualizar(object sender, System.EventArgs e)
{
    if (lbTipo.Text == "Notícias da Instituição")
        preencherLV(1);
    else if (lbTipo.Text == "Notícias dos Cursos")
        preencherLV(2);
    else
        preencherLV(3);
}

private void carregar(object sender, System.EventArgs e)
{
    if (sender.Equals(miInstituicao))
    {
        preencherLV(1);
        lbTipo.Text = "Notícias da Instituição";
    }
    else if (sender.Equals(miCursos))
    {
        preencherLV(2);
        lbTipo.Text = "Notícias dos Cursos";
    }
    else
    {
        preencherLV(3);
        lbTipo.Text = "Notícias da Pastoral";
    }
}

private void lv_GotFocus(object sender, System.EventArgs e)
{
    FormDetalhes formDetalhes = new FormDetalhes();
    formDetalhes.Text = "Detalhes";

    formDetalhes.setNoticia(lv.Items[lv.FocusedItem.Index].SubItems[1].Text);
    formDetalhes.ShowDialog();
}
}
}
}

```

FormDetalhes.cs:

```

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.IO;

namespace NamespaceTeste
{
    /// <summary>
    /// Summary description for FormDetalhes.
    /// </summary>
    public class FormDetalhes : System.Windows.Forms.Form
    {
        private System.Windows.Forms.Label lbTitulo;
        private System.Windows.Forms.PictureBox pbImagem;
        private System.Windows.Forms.TextBox tbTextoPlano;
        private nds03.tcc.Noticias wsNoticias;
        private System.Windows.Forms.TextBox tbResumo;
        private nds03.tcc.NoticiaDetalhe noticia;

        public FormDetalhes()
        {
            InitializeComponent();
            wsNoticias = new nds03.tcc.Noticias();
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            base.Dispose( disposing );
        }

        public void setNoticia(string id_noticia)
        {
            noticia = wsNoticias.NoticiaPorID(id_noticia);
            if(noticia != null)
            {
                lbTitulo.Text = noticia.titulo;
                tbResumo.Text = noticia.resumo;
                tbTextoPlano.Text = noticia.texto_plano;

                byte[] image = noticia.img_big;
                System.IO.MemoryStream memStream = new
System.IO.MemoryStream(image);
                if (memStream.Length != 0)
                {
                    try
                    {
                        Bitmap bm = new Bitmap(memStream);
                        pbImagem.Image = bm;
                    }
                    catch(Exception)

```

```

        {
            MessageBox.Show("Ocorreram problemas
com o formato da imagem.", "Erro");
        }
    }
    else
    {
        MessageBox.Show("Não foi possível apresentar a
imagem","Alerta");
    }
}
}

```

```

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.tbTextoPlano = new System.Windows.Forms.TextBox();
    this.lbTitulo = new System.Windows.Forms.Label();
    this.pbImagem = new System.Windows.Forms.PictureBox();
    this.tbResumo = new System.Windows.Forms.TextBox();
    //
    // tbTextoPlano
    //
    this.tbTextoPlano.BackColor =
System.Drawing.SystemColors.ActiveCaptionText;
    this.tbTextoPlano.Location = new System.Drawing.Point(8, 144);
    this.tbTextoPlano.Multiline = true;
    this.tbTextoPlano.ScrollBars =
System.Windows.Forms.ScrollBars.Vertical;
    this.tbTextoPlano.Size = new System.Drawing.Size(224, 144);
    this.tbTextoPlano.Text = "";
    //
    // lbTitulo
    //
    this.lbTitulo.Font = new System.Drawing.Font("Microsoft Sans
Serif", 8F, System.Drawing.FontStyle.Bold);
    this.lbTitulo.ForeColor = System.Drawing.Color.MidnightBlue;
    this.lbTitulo.Location = new System.Drawing.Point(8, 0);
    this.lbTitulo.Size = new System.Drawing.Size(224, 40);
    this.lbTitulo.TextAlign =
System.Drawing.ContentAlignment.TopCenter;
    //
    // pbImagem
    //
    this.pbImagem.Location = new System.Drawing.Point(136, 56);
    this.pbImagem.Size = new System.Drawing.Size(100, 72);
    this.pbImagem.SizeMode =
System.Windows.Forms.PictureBoxSizeMode.StretchImage;
    //
    // tbResumo
    //

```

```
        this.tbResumo.Font = new System.Drawing.Font("Microsoft Sans
Serif", 7.25F, System.Drawing.FontStyle.Regular);
        this.tbResumo.Location = new System.Drawing.Point(8, 48);
        this.tbResumo.Multiline = true;
        this.tbResumo.ScrollBars =
System.Windows.Forms.ScrollBars.Vertical;
        this.tbResumo.Size = new System.Drawing.Size(120, 88);
        this.tbResumo.Text = "";
        //
        // FormDetalhes
        //
        this.Controls.Add(this.tbResumo);
        this.Controls.Add(this.pbImagem);
        this.Controls.Add(this.lbTitulo);
        this.Controls.Add(this.tbTextoPlano);
        this.Text = "FormDetalhes";
    }
}
#endregion
}
```

ANEXO C

Classe *proxy* gerada a partir do *Web Service* definido na figura 24
FormConsumer.cs e FormDetalhes.cs

```

//-----
// <autogenerated>
//     This code was generated by a tool.
//     Runtime Version: 1.1.4322.573
//
//     Changes to this file may cause incorrect behavior and will be lost
//     if the code is regenerated.
// </autogenerated>
//-----

//
// This source code was auto-generated by wsdl, Version=1.1.4322.573.
//
using System.Diagnostics;
using System.Xml.Serialization;
using System;
using System.Web.Services.Protocols;
using System.ComponentModel;
using System.Web.Services;

/// <remarks/>
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Web.Services.WebServiceBindingAttribute(Name="SomaSoap",
Namespace="http://nds03/michael")]
public class Soma : System.Web.Services.Protocols.SoapHttpClientProtocol
{

    /// <remarks/>
    public Soma() {
        this.Url = "http://nds03/tcc/soma.asmx";
    }

    /// <remarks/>
[System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://nds03/
michael/somar", RequestNamespace="http://nds03/michael",
ResponseNamespace="http://nds03/michael",
Use=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
    public System.Single somar(System.Single valor1, System.Single
valor2) {
        object[] results = this.Invoke("somar", new object[] {
            valor1,
            valor2});
        return ((System.Single)(results[0]));
    }

    /// <remarks/>
    public System.IAsyncResult Beginsomar(System.Single valor1,
System.Single valor2, System.AsyncCallback callback, object asyncState) {
        return this.BeginInvoke("somar", new object[] {
            valor1,
            valor2}, callback, asyncState);
    }

    /// <remarks/>
    public System.Single Endsomar(System.IAsyncResult asyncResult) {
        object[] results = this.EndInvoke(asyncResult);
    }
}

```

```
        return ((System.Single)(results[0]));  
    }  
}
```