UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

PROTÓTIPO PARA ATUALIZAÇÃO ASSÍNCRONA DE DADOS UTILIZANDO WEB SERVICES

SÉRGIO KOCH VAN-DALL

BLUMENAU
2006

2006/1-38

SÉRGIO KOCH VAN-DALL

PROTÓTIPO PARA ATUALIZAÇÃO ASSÍNCRONA DE DADOS UTILIZANDO WEB SERVICES

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Paulo Fernando da Silva - Orientador

BLUMENAU
2006

2006/1-38

PROTÓTIPO PARA ATUALIZAÇÃO ASSÍNCRONA DE DADOS UTILIZANDO WEB SERVICES

Por

SÉRGIO KOCH VAN-DALL

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Paulo Fernando da Silva – Orientador, FURB
Membro:	Prof. Maurício Capobianco Lopes – FURB
Membro:	Prof. Francisco Adell Péricas – FURB

Blumenau, 28 de junho de 2006

Dedico este trabalho a Deus, a meus familiares e esposa, a todos os amigos, especialmente aqueles que me ajudaram diretamente na realização deste trabalho.

AGRADECIMENTOS

À Deus, por estar presente em meus pensamentos e no cotidiano.

À minha esposa Fabiane e sua família, por sempre me apoiarem e me incentivarem na realização dos meus sonhos.

À minha mãe Donatila, que sempre esteve presente em todos os momentos da minha vida, me apoiando e incentivando.

Aos meus amigos, pelos incentivos e cobranças, que me auxiliaram de alguma forma para que este trabalho se tornasse realidade.

Ao meu orientador, Prof. Paulo Fernando da Silva, pela paciência e por ter acreditado e me guiado nesta difícil caminhada que foi a conclusão deste trabalho.

Aprender é a única coisa de que a mente nunca se cansa, nunca tem medo e nunca se arrepende.

Leonardo Da Vinci

RESUMO

Este trabalho descreve a análise, especificação e implementação de um protótipo para atualização assíncrona de banco de dados utilizando serviços web. O protótipo é composto por um cenário representado por uma aplicação cliente, um servidor de serviços web e um servidor de banco de dados, que compartilham informações através de um arquivo XML. Os métodos ficam armazenados em um contêiner HTTP e são invocados através de mensagens utilizando o protocolo SOAP. Foi implementada uma aplicação cliente em ASP.NET que envia informações para o arquivo XML e um aplicativo cliente, sendo o servidor de banco de dados, que lê o arquivo XML e atualiza uma base de dados através de chamadas remotas aos métodos das classes do servidor de serviços web.

Palavras-chave: Redes de computadores. SOAP. Serviços web. XML.

ABSTRACT

This work describes the analysis, specification and implementation of an archetype for asynchronous update of data base using web services. The archetype is composed for a scene represented for an application customer, a server of services web and a server of data base, that shares information through an archive XML. The methods are stored in contêiner HTTP and are invoked through messages using protocol SOAP. An application was implemented customer in ASP.NET that sends information for archive XML and a applicatory customer, being the server of data base, that reads archive XML and brings up to date a database through remote calls to the methods of classrooms of server of web services.

Key-words: Computer networks. SOAP. Web services. XML.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

API – *Application Programming Interface*

ASP – *Active Server Page*

CLR – *Common Language Runtime*

CORBA – *Commom Object Request Broker Architecture*

COM – *Component Object Model*

DCOM – *Distributed Component Model*

DLL – *Dynamic Link Library*

DOM – *Document Object Model*

DTD – *Document Type Definition*

HTTP – *Hypertext Transfer Protocol*

IIS – *Internet Information Server*

IL – *Intermediate Language*

MSDE – *Microsoft SQL Server Desktop Engine*

MSIL – *Microsoft Intermediate Language*

SGML – *Standart Generalized Markup Language*

SOAP – *Simple Object Access Protocol*

UDDI – *Universal Description, Discover and Integration*

UML – *Unified Modeling Language*

URI – *Unique Resourse Identifier*

URL – *Uniform Resource Locator*

VB – *Visual Basic*

XML – *eXtensible Markup Language*

W3C – *World Wide Web Consortium*

SUMÁRIO

1 INTRODUÇÃO

Atualmente, com o enorme crescimento no número de redes de computadores e a complexidade de seus softwares, há a necessidade da comunicação entre eles para que possam compartilhar informações e trabalharem de forma cooperativa.

Segundo Turtschi et al (2004, p. 384), levando em consideração os variados recursos utilizados, como sistemas operacionais, linguagens de programação e modelos de objetos, as tecnologias existentes dificultaram muito a comunicação entre as aplicações de diferentes empresas, pois impõem enormes desafios aos integradores de aplicação.

De acordo com Turtschi et al (2004, p. 384), os *Web Services* foram criados para resolver a interoperabilidade das aplicações através de sistemas operacionais, linguagens de programação e modelos de objetos, onde, para tornar realidade o desenvolvimento de aplicações distribuídas, cria-se um mecanismo comum (*Web Services*), aberto e baseado em padrões, que se comunicam através do protocolo *Simple Object Access Protocol* (SOAP). Segundo Alexander e Hollis (2002, p. 184), SOAP é um padrão para enviar dados de um lado para o outro, entre um cliente e um servidor. Baseia-se em *eXtensive Markup Language* (XML) e é um protocolo simples e leve, que implementa a funcionalidade de chamada de procedimentos remotos para serviços *web*.

A idéia central de um *Web Service* consiste em permitir que as aplicações, sejam elas da web ou *desktop*, ou ainda *middleware*, comuniquem-se e troquem dados de forma simples e transparente, independente do sistema operacional ou da linguagem de programação (LIMA; REIS, 2002, p. 3).

Segundo Tarifa, Facunte e Garcia (2005, p. 10), foi criada a plataforma de desenvolvimento .NET, visando tornar real e viável a conexão de qualquer tipo de aplicação, dispositivo ou sistema em um ambiente distribuído. A plataforma .NET foi criada levando em

consideração práticas e padrões baseados na *web* e a padronização de modelos de programação com suporte a múltiplas linguagens de programação, tendo sua distribuição e gerenciamento simplificado.

Diante dos obstáculos na comunicação e troca de informações em um sistema heterogêneo, surgiu a intenção de desenvolver um protocolo independente de arquitetura, sistema operacional e linguagem de programação, para comunicar-se em um sistema distribuído, levando em consideração a troca dos dados que estão sendo compartilhados, utilizando-se de um mecanismo de comunicação assíncrona, onde os dispositivos envolvidos no processo trabalham de forma independente.

O protótipo será composto por um servidor de banco de dados, um servidor de *Web Services* e uma aplicação cliente, onde a aplicação cliente comunica-se com o servidor de *Web Services* quando deseja adicionar informações no banco de dados. Estas informações são gravadas em um arquivo XML, independente do servidor de banco de dados estar ou não ativo. O servidor de banco de dados pode acessar a qualquer momento o arquivo XML através do servidor de *Web Services* para atualizar a sua base de dados, efetuando desta forma, uma atualização assíncrona da base de dados.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é a criação de um protótipo para atualização assíncrona de dados no servidor de banco de dados utilizando *Web Services* através do protocolo SOAP.

Os objetivos específicos do trabalho são:

a) utilizar XML como uma linguagem de marcação para a confecção dos documentos que serão utilizados no intercâmbio das informações;

b) criar um *Web Service* para guardar as informações de forma estruturada no arquivo

XML, utilizando o protocolo SOAP;

c) gerar atualização assíncrona dos dados no servidor de banco de dados através da chamada do *Web Service*;

d) criar uma aplicação cliente para demonstração do funcionamento do protótipo.

1.2 ESTRUTURA DO TRABALHO

No primeiro capítulo são apresentados objetivos e a estrutura deste trabalho.

No segundo capítulo são apresentadas as tecnologias utilizadas neste trabalho, tais como plataforma .NET, XML, *Web Services* e SOAP, além de alguns trabalhos correlatos.

No terceiro capítulo são apresentados os requisitos, a especificação e a implementação do protótipo.

No quarto capítulo são apresentadas as conclusões e sugestões para o desenvolvimento de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nas próximas seções são apresentados conceitos e técnicas que foram relevantes no desenvolvimento do trabalho.

2.1 PLATAFORMA .NET

Segundo Turtschi et al (2004, p. 2), a plataforma .NET é muito mais do que uma linguagem, um *kit* de programação ou mesmo um sistema operacional. Ela oferece serviços novos e poderosos, um novo formato binário independente de processador, novas linguagens gerenciadas, extensões de linguagem gerenciada para linguagens existentes, entre os vários recursos .NET.

Segundo Turtschi et al (2004, p. 3), suas principais características são:

a) independência de sistema operacional: a plataforma compila o programa para código nativo antes da primeira execução;
b) integração entre linguagens: todas as linguagens .NET utilizam a mesma *Application Programming Interface* (API) de desenvolvimento;
c) tempo de execução compartilhado: o "*runtime*" é compartilhado de forma igual entre as diversas linguagens que a suportam;
d) facilidade de distribuição: ao serem compilados, os executáveis anotam as versões das *Dynamic Link Library* (DLLs) que foram utilizadas, verificando e utilizando sempre as mesmas DLLs nas demais distribuições;
e) confiabilidade: possui tipagem forte. Todas as classes .NET derivam da mesma classe mãe, *Object*. Mecanismo de coleta de lixo executado quando a aplicação está aparentemente fora da memória livre ou simplesmente quando ela é chamada e

não há mais memória livre para gerenciar.

2.1.1 .NET *Framework*

Segundo Alexander e Hollis (2002, p. 14), o .NET *Framework* é uma plataforma de desenvolvimento de aplicação que reúne todas as funções (serviços e aplicações Windows) normalmente associadas ao sistema operacional e necessárias ao funcionamento dos programas.

O .NET *Framework* pode ser dividido em três partes:

a) *Common Language Runtime* (CLR): um ambiente de execução gerenciado que manipula alocação de memória, detecção de erro e interação com os serviços do sistema operacional;
b) *Base Class Library*: uma extensa coleção de componentes de programação e interfaces de programas aplicativos (APIs);
c) dois destinos de desenvolvimento de alto nível: um para aplicações *web* (ASP.NET), para criação de páginas na internet e *Web Services*; e outro para aplicações Windows comuns (Windows *Forms*), para criação de aplicações modulares.

As vantagens oferecidas pelo .NET *Framework* incluem ciclos de desenvolvimento mais curtos (reutilização de código, menos problemas em ambientes distribuídos, suporte para múltiplas linguagens de programação), maior facilidade de distribuição, menos *bugs* relacionados a tipos de dados devido à segurança de tipo integral (*byte*, *short*, *int* e *long*), redução das falhas de memória graças ao coletor de lixo, mais escalabilidade e aplicações mais confiáveis (TURTSCHI et al, 2004, p. 2).

2.1.2 CLR

O maior dos componentes do .NET *Framework* é o CLR. Com o conceito semelhante ao *Java Virtual Machine*, o CLR é um ambiente de *Runtime* que executa código *Microsoft Intermediate Language* (MSIL) ou *Intermediate Language* (IL) e, diferente do ambiente Java, suporta múltiplas linguagens de programação. Como todo o código visando a plataforma .NET roda com o ambiente CLR, ele é chamado de código gerenciado. Isso significa simplesmente que a execução do código e o seu comportamento são gerenciados pelo CLR. Quando um programa é compilado para o CLR, seu resultado é chamado de Código Gerenciado (*Managed Code*), que nada mais é do que um código que usufrui das vantagens do CLR. Para o *Runtime* trabalhar com código gerenciado, este código precisa conter metadados (*metadata*), que são criados durante o processo de compilação. Estes metadados são armazenados com o código compilado e contêm as informações relativas a tipos, membros e referências no código. Entre outras coisas, o CLR utiliza os metadados para localizar e carregar classes, gerar código nativo e segurança (TURTSCHI et al, 2004, p. 10).

Segundo Turtschi et al (2004, p. 10), todas as aplicações .NET, independentes de suas linguagens de origem, compartilham um sistema de tipos comuns. Isso significa que qualquer tipo .NET possui os mesmos atributos, independente da linguagem em que é usado.

Os *assemblies* são os meios de empacotar e distribuir aplicações e componentes no .NET e contêm toda a informação necessária (metadados) para o funcionamento do programa.

Os metadados são um recurso que permite ao CLR saber os detalhes sobre um componente específico. São eles que fazem referência a *assemblies* externos com as DLLs que são usadas e suas respectivas versões. O CLR usa os metadados para verificação, reforço de segurança, *layout* da memória e para a execução da aplicação. O Carregador de Classes utiliza os metadados para achar e carregar classes .NET, já que os metadados contêm

informações de onde essas classes encontram-se.

2.2 XML

De acordo com Turtschi et al (2004, p. 290), XML emergiu com o padrão da *web* para representação e transmissão de dados pela internet. XML é uma linguagem de descrição de dados genérica e independente de plataforma e, como tal, ganhou grande popularidade na área da computação, sendo adotada por muitas das maiores empresas do setor, tais como IBM, Microsoft e Sun.

Segundo Bento (2005, p. 19), XML foi desenvolvida pelo XML *Working Group* que pertence ao *World Wide Web Consortium* (W3C) e é uma tecnologia aberta para troca de dados. É uma linguagem de marcação descendente da *Standart Generalized Markup Language* (SGML) que possui as seguintes características:

a) poder de armazenamento e organização das informações em um formato adequado;
b) possui *unicode* como conjunto de caracteres padrão;
c) oferece várias maneiras de validação de documentos;
d) fácil entendimento para seres humanos e programas;
e) formato puro dos dados não dificulta as conversões de formato;
f) tem a capacidade de ter seu vocabulário extensível.

Em essência, XML é uma maneira de comunicar dados estruturados entre aplicações e sistemas de computadores, mesmo quando estão envolvidas múltiplas arquiteturas, linguagens e padrões (ALEXANDER; HOLLIS, 2002, p. 179).

Um documento XML consiste de elementos e atributos que implementam uma hierarquia em árvore. Um modelo de documento XML é mostrado no Quadro 1.

```
<?xml version= “1.0” ?>
- <BookOnCars xmins=”http://www.publishing.org”
   xmls:xsi=”http://www.w3.org/2001/XMLSchema-instance”
   xsi:schemaLocation=”http://www.publishing.org
ScopeTest.xsd”>
    -<Chapter>
      <Title>My Car</Title>
     -<Section>
      <Title>Thunderbird</Title>
      </Section>
     </Chapter>
     <Title>A car</Title>
  </BookOnCar>
```

Fonte: Bento (2005, p. 21).

Quadro 1 – Modelo de documento XML

De acordo com Skonnard e Gudgin (2002, p. 5), os documentos XML são constituídos na sua maioria de elementos, sendo que todo documento XML tem um elemento de nível mais alto conhecido como elemento raiz que serve de contêiner para todos os outros elementos. Os elementos têm nomes e podem ter outros elementos filhos.

Para processar arquivos XML é necessário um analisador sintático para os documentos. Este analisador pode ser baseado em *Document Type Definition* (DTD) ou XML *Schemas*, para modelagem dos documentos. Ao utilizar um analisador sintático é garantida a boa formatação dos documentos (BENTO, 2005, p. 20).

2.2.1 Elementos e Atributos

Segundo Alexander e Hollis (2002, p. 182), um elemento consiste em um *tag* de abertura, qualquer informação que ele contenha, e o *tag* de fechamento. Os elementos podem ter elementos filhos. Um documento XML adequadamente formatado pode conter apenas um único elemento-raiz. Depois disso, pode-se utilizar qualquer grau de aninhamento, podendo-se colocar diferentes tipos de elementos no mesmo nível. Um elemento pode conter dados entre seus *tags* de início e final.

Segundo Bento (2005, p. 21), XML não define qualquer nome de elemento, ficando a critério do desenvolvedor escolher os nomes da maneira que desejar, lembrando que XML é

case-sensitive e os nomes de elementos devem iniciar com uma letra ou um *underscore*.

Os atributos são pares/valor que aparecem com o *tag* de início de um elemento, permitindo-se o detalhamento dos elementos. Podem ser usados para descrever propriedade de um elemento ou algum aspecto de comportamento do elemento. Um elemento pode conter inúmeros atributos que são separados por espaços em branco. Os atributos aparecem após o nome do elemento em qualquer ordem e seus valores devem estar entre apóstrofos (‘) ou aspas (“).

O Quadro 2 mostra o elemento *Customer* que contém atributos ID, *FirstName* e *LastName*. O elemento *Loan* contém atributos de *CurrentDue* e *Balance*. O elemento *Notes* não contém atributos.

```
<Customer ID=’9853651 ’ FirstName=’Lydia ’ LastName=’Smith ’>
  <Loan CurrentDue=’295.00’ Balance=’4899.00’ />
  <Notes>Uma de nossas melhores clientes, Ms. Smith faz negócios
conosco há anos, e sempre cumprimenta nossos caixas com um
sorriso. </Notes>
</Customer>
```

Fonte: Alexander e Hollis (2002, p. 182).

Quadro 2 –Elementos e atributos

2.3 WEB SERVICES

De acordo com Turtschi et al (2004, p. 384), os *Web Services* podem ser definidos como aplicações modulares, baseadas na internet, que realizam tarefas comerciais específicas e de acordo com um formato técnico específico.

Segundo Bento (2005, p. 25), *Web Services* têm como principal característica a integração entre sistemas heterogêneos, utilizando padrões de protocolos independentes da plataforma e linguagem de programação em que foram desenvolvidos. Os *Web Services* são usados para disponibilizar serviços interativos na *web*, podendo ser acessados por outras aplicações.

Segundo Alexander e Hollis (2002, p. 177), os *Web Services* podem ser pensados como componentes *Component Object Model* (COM) da internet, sendo que um componente COM foi projetado para fornecer um modelo baseado em interface de comunicação de informações entre componentes em uma única máquina.

Em .NET, os *Web Services* permitem que se encapsule código, publique interfaces, descubra serviços e se comunique entre quem publica e quem consome os serviços utilizando tecnologias padrão independentes de fornecedor, conforme arquitetura mostrada na Figura 1.

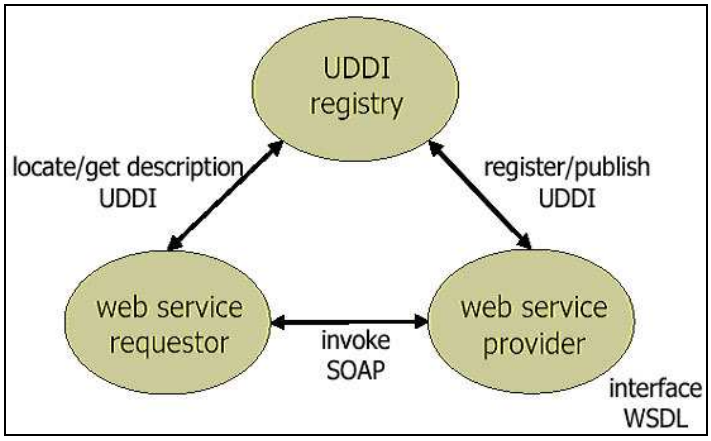


Figura 1 – Arquitetura dos *Web Services*

2.3.1 Camadas

Segundo Bento (2005, p. 27), os *Web Services* utilizam uma pilha de serviços dividida em camadas conforme Figura 2, onde os principais componentes são:

a) *Web Service Description Language* (WSDL): utilizado na camada de descrição de serviços *web*. O WSDL é o responsável por fornecer o documento que contém a interface do serviço para que possa ser implementada pelos clientes;

b) *Universal Description, Discover and Integration* (UDDI): atua na camada de descoberta (descrição do objetivo do *Web Service*, de que forma ele pode ser utilizado, quem o construiu, entre outros), publica dados sobre as empresas fornecedoras e descreve os serviços para implementação da interface de

comunicação por parte de seus clientes;

c) XML: atua na camada de dados, responsável pelo formato das mensagens;

d) SOAP: protocolo padrão para vínculo dos serviços *web* utilizado na camada de mensagens;

e) *Hypertext Transfer Protocol* (HTTP): atua na camada de transporte do ponto de vista de *Web Services*. Utilizado para encapsular as mensagens SOAP em requisições do tipo POST.

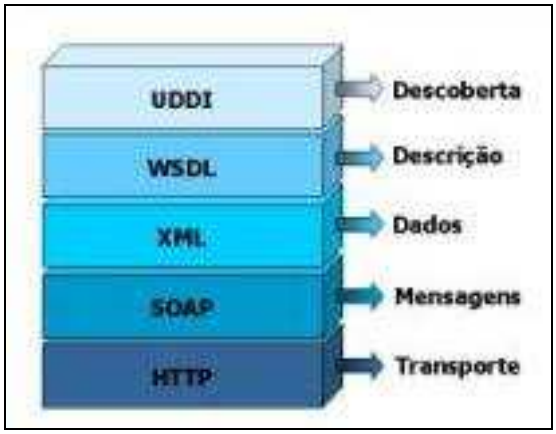


Figura 2 – Pilha de camadas dos *Web Services*

Os *Web Services* permitem que se exponha códigos que implementam a lógica empresarial que podem ser reutilizados em múltiplas aplicações, sendo baseadas em tecnologias de internet e protocolos independentes de fabricantes (ALEXANDER; HOLLIS, 2002, p. 6).

2.3.2 Vantagens

A recente ênfase nos *Web Services* denota uma significante mudança no desenvolvimento de aplicação: de soluções insulares e monolíticas para aplicações verdadeiramente distribuídas, modulares, abertas, interempresas e baseadas na internet. A idéia certamente é que os *Web Services* façam pelas aplicações corporativas o que a *World Wide Web* fez pelas aplicações interativas de usuário final (TURTSCHI et al, 2004, p. 385).

Segundo Alexander e Hollis (2002, p. 177), no modelo de *Web Services*, cada sistema da organização atua como um componente independente na arquitetura de integração. Todas as interfaces, transformações de dados e comunicações entre componentes são baseados em padrões abertos e vastamente adotados, independentes de fornecedores e plataforma.

As vantagens de se utilizar essa abordagem são:

a) simplicidade: é mais simples de se implementar que as soluções tradicionais que utilizam CORBA ou DCOM;
b) padrões abertos: utilizam padrões abertos como HTTP, SOAP, UDDI, ao invés de tecnologias proprietárias;
c) flexibilidade: alterações nos componentes são muito mais simples para o sistema como um todo do que alterações nos adaptadores tradicionais;
d) custo: as soluções tradicionais são muito mais caras;
e) escopo: cada sistema pode ser tratado de maneira individual, já que para "componentizá-lo" basta implementar uma camada que o encapsule. Na abordagem tradicional, todos os sistemas devem ser tratados ao mesmo tempo, já que farão parte da mesma solução monolítica de integração.

2.4 SOAP

De acordo com Turtschi et al (2004, p. 384), o SOAP é um padrão de internet aberto. Ele foi projetado originalmente pela IBM, Ariba e Microsoft, e o W3C tomou a iniciativa de desenvolvê-lo. Foi elaborado com os três seguintes objetivos:

a) ser otimizado para rodar na internet;
b) ser simples e fácil de implementar;
c) ser baseado em XML.

O SOAP é um protocolo que permite a dois sistemas – um cliente e um servidor – trocarem dados, onde, embora tenha sido especificado de modo a ser implementado em uma variedade de protocolos de transporte da internet, ele é mais usado sobre o HTTP.

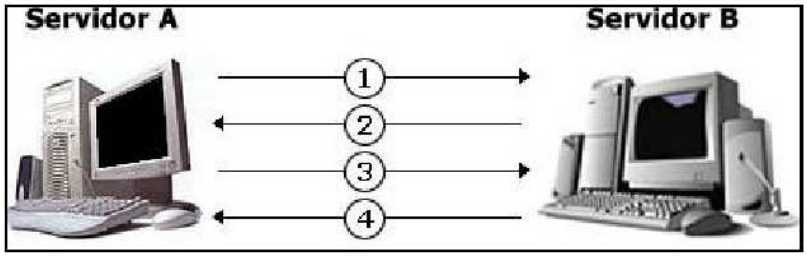
Segundo Alexander e Hollis (2002, p. 186), SOAP é utilizado para implementar chamadas de procedimentos remotos e trocar dados entre o código que utiliza um serviço *web* (o cliente) e o serviço *web* propriamente dito (o servidor).

"O SOAP é um protocolo padrão na tecnologia dos *Web Services* para a troca de informações em um ambiente distribuído." (SNELL; TIDWELL; KULCHENKO, 2002, p. 11, tradução nossa).

De acordo com Turtschi et al (2004, p. 385), o protocolo SOAP apresenta muitas vantagens em comparação com outras aplicações de processamento distribuído. Algumas destas vantagens são:

a) capacidade de transpor *firewalls* facilmente ao ser utilizado sobre o HTTP;
b) utiliza a estruturação de dados em XML;
c) é satisfatoriamente mapeado no padrão solicitação/resposta do HTTP;
d) superficial como um protocolo, pois contém menos recursos do que outros protocolos de computação distribuídos, permitindo que sistemas distribuídos se comuniquem diretamente, sem nenhum intermediário;
e) existe suporte para o protocolo SOAP por muitos fornecedores como Microsoft, IBM e a Sun.

A Figura 3 ilustra os procedimentos realizados para troca de mensagens, utilizando SOAP, entre dois servidores (A e B).



Fonte: Souza (2003, p. 2).

Figura 3 – Modelo de estrutura da chamada de métodos entre servidores

Os elementos do mecanismo de troca de mensagens apresentado pela Figura 3 podem ser explicados da seguinte maneira:

1) servidor A envia solicitação SOAP ao servidor B para receber informações da biblioteca de tipos;

2) servidor B envia de volta a mensagem SOAP com a biblioteca de tipo;

3) servidor A envia envelope SOAP com o método que será chamado;

4) servidor B executa o método e envia de volta o resultado em um envelope SOAP.

Para que os servidores consigam se comunicar, é necessário que eles entendam o mesmo “vocabulário”. Os métodos (e seus argumentos) e os tipos precisam ser entendidos por ambas as partes. Para isso existe a WSDL, que disponibiliza as informações necessárias para compor as solicitações e as respostas SOAP.

2.4.1 WSDL

Segundo Alexander e Hollis (2002, p. 187), é um novo padrão independente de representação para a definição de interface de serviços *web*. Um documento WSDL define todos os métodos expostos pelos serviços *web*; os nomes, os tipos de dados e ordem dos parâmetros; e os tipos de dados retornados. Foi descrito como o contrato entre um serviço *web* e os clientes desse serviço.

A Figura 4 apresenta um exemplo de mensagem WSDL para o serviço “Calculadora” onde são descritos os tipos e a forma de comunicação a ser estabelecida com um serviço que

tem um método (“Soma”) que realiza a soma de dois números reais.

```
<?xml version="1.0" encoding="utf-8"?>
<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">
 <types>
   <s:schema elementFormDefault="qualified"
targetNamespace="http://tempuri.org/">
     <s:element name="Soma">
       <s:complexType>
         <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="a" type="s:float" />
            <s:element minOccurs="1" maxOccurs="1" name="b" type="s:float" />
         </s:sequence>
       </s:complexType>
     </s:element>
     <s:element name="SomaResponse">
       <s:complexType>
         <s:sequence>
           <s:element minOccurs="1" maxOccurs="1" name="SomaResult"
type="s:float"/>
         </s:sequence>
       </s:complexType>
     </s:element>
   </s:schema>
</types>
<message name="SomaSoapIn">
  <part name="parameters" element="s0:Soma" />
</message>
<message name="SomaSoapOut">
  <part name="parameters" element="s0:SomaResponse" />
</message>
<service name="Calculadora">
  <port name="CalculadoraSoap" binding="s0:CalculadoraSoap">
    <soap:address location="http://localhost/webservices/calculadora.asmx" />
  </port>
</service>
</definitions>
```

Fonte: Souza (2003, p. 3).

Figura 4 – Exemplo de trecho de mensagem WSDL para o serviço “Calculadora”

Na Figura 4, o elemento *<s:element name="Soma">* contém uma série de elementos *<s:element>* que definem os tipos de dados dos parâmetros do método “Soma”. Os elementos *<message name="SomaSoapIn">* e *<messagename="SomaSoapOut">* representam, respectivamente, as mensagens utilizadas para solicitação e envio de resposta.

2.4.2 Mensagens SOAP

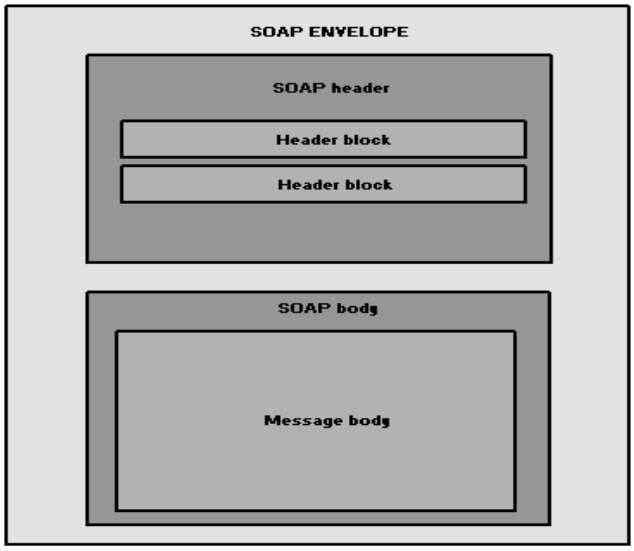
A troca de mensagens é feita através da sintaxe de envelope para envio e recebimento de mensagens XML com *Web Services*. Isto é, o SOAP é o envelope que empacota as

mensagens XML que são enviadas sobre HTTP entre clientes e *Web Services*.

Segundo Alexander e Hollis (2002, p. 185), as mensagens SOAP são representadas através de documentos XML e contêm os seguintes elementos:

a) envelope SOAP: elemento raiz obrigatório de uma mensagem SOAP, que define um documento XML como uma mensagem SOAP;
b) cabeçalho SOAP: elemento opcional, utilizado para adição de recursos de autenticação, gerenciamento de transações e serviços de pagamento;
c) corpo SOAP: elemento obrigatório, que contém a mensagem SOAP endereçada ao destino final da mensagem.

A Figura 5 mostra as partes que compõem a estrutura de uma mensagem SOAP.



Fonte: Snell, Tidwell e Kulchenko (2002, p. 11).
Figura 5 – Estrutura de uma mensagem SOAP

2.4.2.1 A mensagem SOAP de solicitação

Assim que a máquina solicitante recebe a mensagem WSDL da descrição do serviço, já é possível compor uma mensagem SOAP de solicitação, pedindo ao servidor para executar um determinado método e retornar algum resultado. A Figura 6 apresenta um exemplo de

mensagem de solicitação que poderia ser utilizada para chamar o método “Soma”, passando-se os parâmetros “3.5” e “6.5”, do serviço “Calculadora”.

```
<?xml version="1.0" encoding="utf-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/">
  <soap:Body>
    <Soma xmlns="http://tempuri.org/">
      <a>3,5</a>
      <b>6,5</b>
    </Soma>
  </soap:Body>
</soap:Envelope>
```

Fonte: Souza (2003, p. 4).

Figura 6 – Exemplo de mensagem SOAP de solicitação do método “Soma” do serviço “Calculadora”

Na Figura 6, a mensagem inclui um elemento <*soap:envelope*>, que define um esquema para informações que estarão presentes no corpo da solicitação e um elemento <*soap:body*>, que é o corpo da mensagem de solicitação (representada pelo elemento <*Soma*>, descrito pela WSDL (Figura 4) pelo elemento <*s:element name="Soma"*>.

2.4.2.2 A mensagem SOAP de resposta

Após uma mensagem de solicitação, devidamente formatada, ser recebida pelo servidor, a mensagem de resposta é devolvida, contendo o resultado da execução do método. A Figura 7 apresenta um exemplo de mensagem de resposta a uma solicitação de execução do método “Soma” do serviço “Calculadora”.

```
<?xml version="1.0" encoding="utf-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/">
  <soap:Body>
    <SomaResponse xmlns="http://tempuri.org/">
      <SomaResult>10</SomaResult>
    </SomaResponse>
  </soap:Body>
</soap:Envelope>
```

Fonte: Souza (2003, p. 4).

Figura 7 – Exemplo de mensagem SOAP de resposta do método “Soma” do serviço “Calculadora”

Na Figura 7, no corpo da mensagem, está contido o elemento <*SomaResponse*> que

contém a resposta à chamada do método “Soma” e está descrito na WSDL (Figura 4) pelo elemento *<s:element name=”SomaResponse”>*, neste exemplo, contém o elemento *<SomaResult>* com valor “10”.

2.5 TRABALHOS CORRELATOS

Em Bento (2005) é demonstrada a implementação de um protótipo de protocolo de aplicação para realizar a troca de documentos entre sistemas de cartórios distintos, utilizando serviços *web*. Este protocolo disponibiliza métodos em uma classe Java armazenada em um *contêiner* HTTP e também implementa um aplicativo cliente que acessa dados em uma base de dados SQL remota através de métodos de uma classe servidora de serviços *web*. Os métodos são invocados através de mensagens SOAP, o qual é responsável pela troca de documentos entre os cartórios, independente da heterogeneidade dos sistemas que estão se comunicando.

Em Becker, Claro e Sobral (2004) é enfatizado o ambiente heterogêneo da internet, onde são encontrados equipamentos e sistemas de vários fabricantes diferentes em aplicações diversas. Isto torna a integração desses sistemas e aplicações uma tarefa complicada e dispendiosa. Aborda duas novas tecnologias, a linguagem XML como um meio de compartilhar a informação de forma neutra e *Web Services*, que representa um novo paradigma da programação distribuída e promove a integração de sistemas e aplicações de uma maneira fracamente acoplada, dinâmica e programática. Além disso, são abordadas tecnologias relacionadas com *Web Services*, como UDDI, WSDL e SOAP. Para exemplificar a integração destas tecnologias foi implementada uma aplicação relativa ao planejamento de uma viagem a um determinado lugar, que executa operações utilizando *Web Services* e XML.

Em Colpani (2002) é demonstrada a implementação de um protótipo de aplicação

capaz de possibilitar a troca de catálogos entre aplicações de comércio eletrônico, utilizando a internet. Esta aplicação disponibiliza funcionalidades em um servidor HTTP através de *Web Services* baseados no protocolo SOAP e na linguagem XML. Também são implementadas duas aplicações de comércio eletrônico como exemplo do uso da aplicação principal.

Em Wieczorek (2004), foi desenvolvido um aplicativo para gerenciamento de documentos jurídicos, utilizando XML como a estrutura de armazenamento das informações dos documentos. Utilizando a plataforma .NET e API DOM para o processamento dos dados dos formulários e armazenamento destes em arquivos XML. Efetuando ainda a validação do conteúdo de cada documento jurídico de acordo com uma definição de tipo de documento (DTD), criada para descrever a estrutura (elementos e relação entre eles) de cada documento.

3 DESENVOLVIMENTO DO CENÁRIO PROPOSTO

O presente trabalho resultou na construção de um protótipo composto por uma aplicação cliente, um servidor de *Web Services* e um servidor de banco de dados, onde a comunicação entre os mesmos ocorre em um ambiente *web*, efetuando a troca das informações através de um arquivo no formato XML, utilizando o protocolo SOAP via HTTP, conforme cenário mostrado na Figura 8.

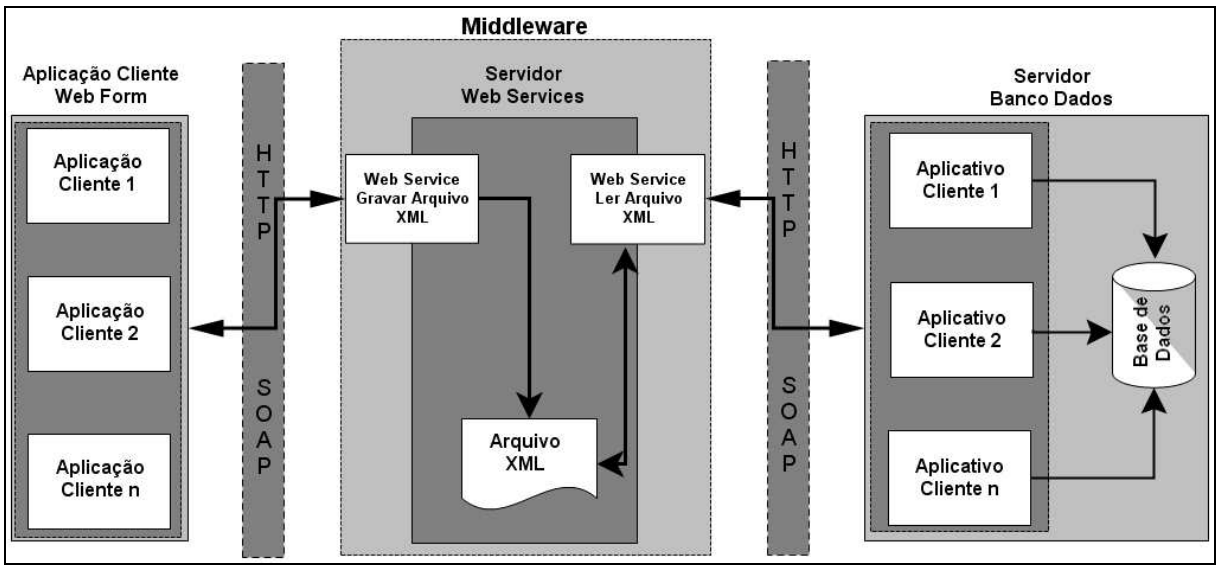


Figura 8 – Demonstração do cenário utilizado no desenvolvimento do protótipo

Na seqüência serão apresentados os principais requisitos especificados no desenvolvimento do trabalho.

3.1 PRINCIPAIS REQUISITOS ESPECIFICADOS NO DESENVOLVIMENTO DO TRABALHO

Desenvolveu-se este trabalho levando em consideração os seguintes requisitos funcionais (RF) e não funcionais (RNF):

a) realizar o acesso aos *Web Services* através de uma aplicação cliente (RF);

b) realizar o acesso aos *Web Services* através de um cliente servidor de banco de

dados (RF);

c) realizar atualização assíncrona no servidor de banco de dados através de *Web Services* utilizando o protocolo SOAP (RF);

d) utilizar um arquivo XML como uma linguagem de marcação para a confecção das informações que serão compartilhadas (RNF).

As próximas seções descrevem a especificação e implementação do protótipo proposto, tendo como cenário uma aplicação cliente, um servidor de *Web Services* e um servidor de banco de dados, necessários para a demonstração do funcionamento do protótipo.

3.2 ESPECIFICAÇÃO

Nesta seção são apresentadas especificações do protótipo para atualização assíncrona no banco de dados em um cenário composto por uma aplicação cliente, um servidor de *Web Services* e um servidor de banco de dados.

O protótipo desenvolvido neste trabalho visa atender a necessidade da integração de sistemas distintos através da troca de informações entre os mesmos, levando em consideração os dados que estão sendo compartilhados. O protótipo utiliza um servidor de *Web Services* que disponibiliza métodos para gravação e leitura de um arquivo no formato XML para atualizar de forma assíncrona dados em um servidor de banco de dados, através de chamadas remotas efetuadas pela aplicação cliente e pelo servidor de banco de dados.

Para especificação dos requisitos foram utilizadas as técnicas da *Unified Modeling Language* (UML) através da ferramenta *Enterprise Architect* 6.1, para descrição dos casos de uso, diagramas de classes e diagramas de seqüências.

3.2.1 Diagramas de classes do protótipo

O protótipo consiste em classes que fazem referência às respostas e tratamentos efetuados no servidor de *Web Services* aos métodos invocados remotamente pela aplicação cliente e pelo servidor de banco de dados. As classes serão apresentadas através de diagramas UML e foram divididas em duas partes, uma referente a chamada remota dos métodos do servidor de *Web Services* pela aplicação cliente e outra pelo servidor de banco de dados.

3.2.1.1 Diagrama de classes: aplicação cliente

O diagrama de classes da Figura 9 mostra as classes principais utilizadas pelo protótipo na formatação e envio dos dados para gravação no arquivo XML através da chamada remota dos métodos no servidor de *Web Services* pela aplicação cliente.

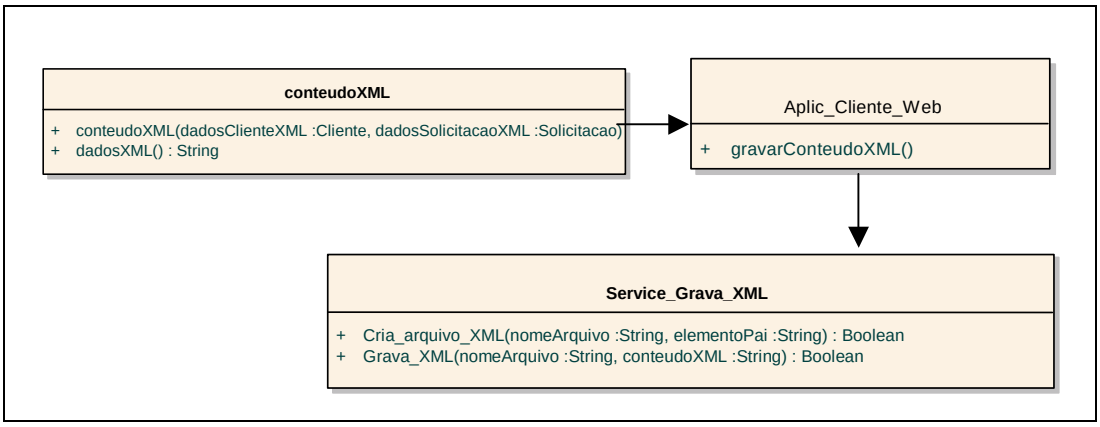


Figura 9 – Diagrama de classes da chamada remota dos métodos no servidor de *Web Services* pela aplicação cliente

As principais classes mostradas na Figura 9 são apresentadas em detalhes nas próximas seções.

3.2.1.1.1 Classe *conteudoXML*

A classe *conteudoXML* possui dois métodos que são *conteudoXML*, sendo o método construtor desta classe e o método *dadosXML*. Estes métodos representam respectivamente a formatação e o conteúdo das informações que são enviadas para serem gravadas no arquivo XML.

O método *conteudoXML* representa as informações formatadas de acordo com as tabelas e campos já existentes no servidor de banco de dados. Possui dois atributos: um referente à estrutura *Cliente*, e outro referente à estrutura *Solicitacao*, onde ambos são os nomes das tabelas que estão criadas no servidor de banco de dados.

Tanto o atributo *Cliente*, quanto *Solicitacao*, são constituídos por vários atributos, que são respectivamente os campos já existentes nas tabelas no servidor de banco de dados.

O método *dadosXML* representa o conteúdo das informações de uma forma concatenada, separando os identificadores e nomes das tabelas, nome dos campos e seus respectivos valores por ponto e vírgula (;).

3.2.1.1.2 Classe *Aplic_Cliente_Web*

A classe *Aplic_Cliente_Web* possui apenas o método *gravarConteudoXML*, que é o responsável por enviar as informações que serão gravadas no arquivo XML.

O método *gravarConteudoXML* instancia o objeto responsável por gravar as informações no servidor de *Web Services* e envia os parâmetros responsáveis por localizar o arquivo XML.

3.2.1.1.3 Classe *Service_Grava_XML*

A classe *Service_Grava_XML* é o serviço *web* que encontra-se no servidor de *Web Services* e responsável pela criação do arquivo XML e gravação das informações neste arquivo. Ela possui dois métodos que são *Cria_arquivo_XML* e *Grava_XML*.

O método *Cria_arquivo_XML* representa a criação do arquivo no formato XML, caso o mesmo não exista. Possui os seguintes atributos:

a) *nomeArquivo*: o nome do arquivo no formato XML;
b) *elementoPaiXML*: o elemento de maior nível, que conterá todas as informações passadas como seus elementos filhos.

O método *Grava_XML* representa a gravação das informações no arquivo XML. Possui os seguintes atributos:

a) *nomeArquivo*: o nome do arquivo no formato XML;
b) *dadosArquivoXML*: informações que serão gravadas no arquivo XML.

3.2.1.2 Diagrama de classes: servidor de banco de dados

O diagrama de classes da Figura 10 mostra as classes principais utilizadas pelo protótipo na leitura das informações no arquivo XML através da chamada remota dos métodos no servidor de *Web Services* pelo servidor de banco de dados.

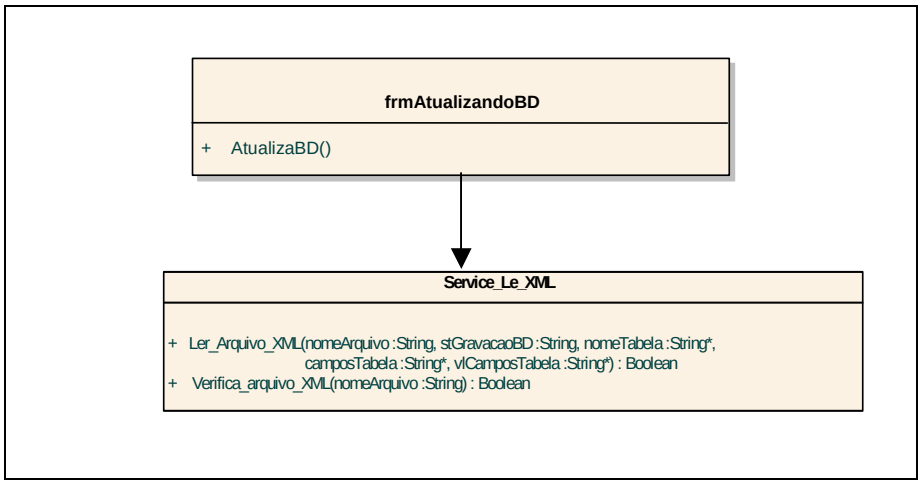


Figura 10 – Diagrama de classes da chamada remota dos métodos no servidor de *Web Services* pelo servidor de banco de dados

As principais classes mostradas na Figura 10 são apresentadas em detalhes nas próximas seções.

3.2.1.2.1 Classe *frmAtualizandoBD*

A classe *frmAtualizandoBD* possui apenas o método *atualizaBD*, que é o responsável por ler as informações no arquivo XML e atualizar a base de dados no servidor de banco de dados.

O método *atualizaBD* instancia o objeto responsável por ler as informações do arquivo XML e envia os parâmetros responsáveis por localizar este arquivo no servidor de *Web Services*.

3.2.1.2.2 Classe *Service_Le_XML*

A classe *Service_Le_XML* é o serviço *web* que encontra-se no servidor de *Web Services* é responsável pela leitura das informações do arquivo XML. Ela possui dois métodos que são *Verifica_arquivo_XML* e *Ler_Arquivo_XML*.

O método *Verifica_arquivo_XML* representa a verificação da existência do arquivo XML no local informado. Possui o seguinte atributo:

a) *nomeArquivo*: o nome do arquivo no formato XML.

O método *Ler_Arquivo_XML* representa a leitura das informações no arquivo XML. Possui os seguintes atributos:

a) *nomeArquivo*: o nome do arquivo no formato XML;
b) *stGravacaoBD*: status da gravação das informações provindas do arquivo XML no banco de dados;
c) *nomeTabela*: nome da tabela onde será gravado o novo registro conforme descrito no arquivo XML;
d) *camposTabela*: nome dos campos da referida tabela que serão atualizados com as informações provindas do arquivo XML;
e) *vlCamposTabela*: valores dos respectivos campos dos parâmetros anteriores que serão atualizados com as informações do arquivo XML.

3.2.2 Casos de uso e diagramas de seqüência do protótipo

A especificação do protótipo pode ser demonstrada através de casos de uso e diagramas de seqüência. Estes representam as ações realizadas nas respostas e tratamentos efetuados no servidor de *Web Services* aos métodos invocados remotamente pela aplicação cliente e pelo servidor de banco de dados.

Os casos de uso do protótipo estão divididos em duas partes, sendo que a aplicação cliente representa a parte inicial do protótipo e é responsável por enviar as informações que serão gravadas no arquivo XML. Por fim, tem-se o servidor de banco de dados, onde as informações gravadas no arquivo XML serão então atualizadas no banco de dados e excluídas

do arquivo XML.

3.2.2.1 Casos de uso: aplicação cliente

Os casos de uso mostrados na Figura 11 demonstram a especificação do protótipo através da aplicação cliente na geração das informações que serão compartilhadas entre as aplicações.

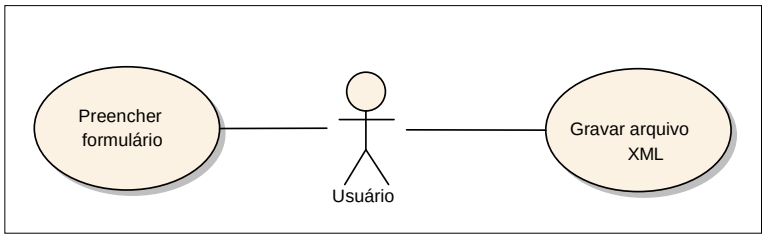


Figura 11 – Casos de uso: aplicação cliente

O caso de uso *Preencher Formulário* da aplicação cliente representa o envio das informações que serão gravadas no arquivo XML. É através deste caso de uso que são preenchidas as informações que são gravadas no arquivo XML, identificando as referidas tabelas e campos a serem atualizados no servidor de banco de dados. A Figura 12 mostra o diagrama de seqüência *Preencher Formulário*.

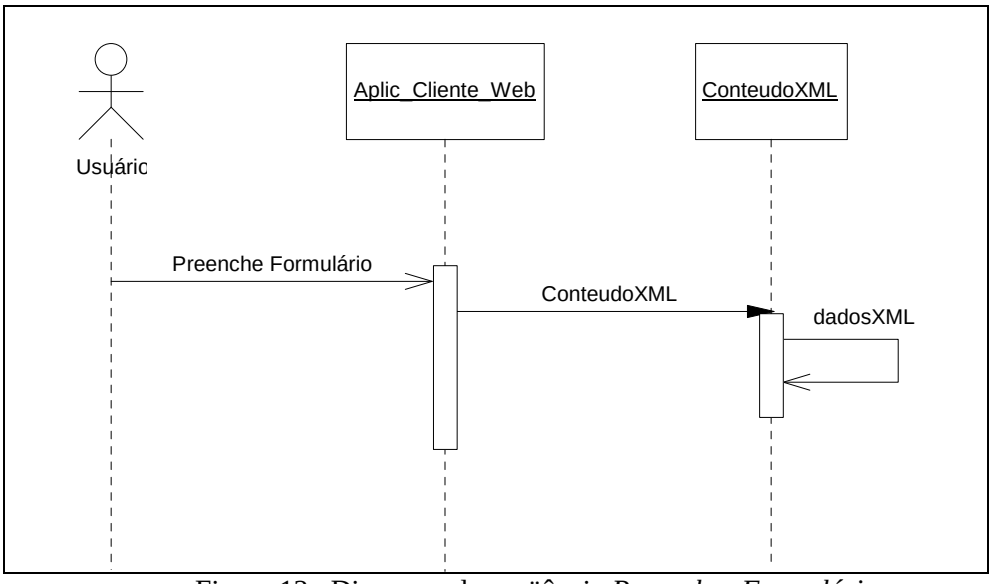


Figura 12– Diagrama de seqüência *Preencher Formulário*

A gravação das informações no arquivo XML é demonstrada no caso de uso *Gravar*

arquivo XML do servidor de *Web Services*, onde as informações são gravadas de forma estruturada no arquivo XML. A Figura 13 demonstra o diagrama de seqüência do caso de uso *Gravar arquivo XML*.

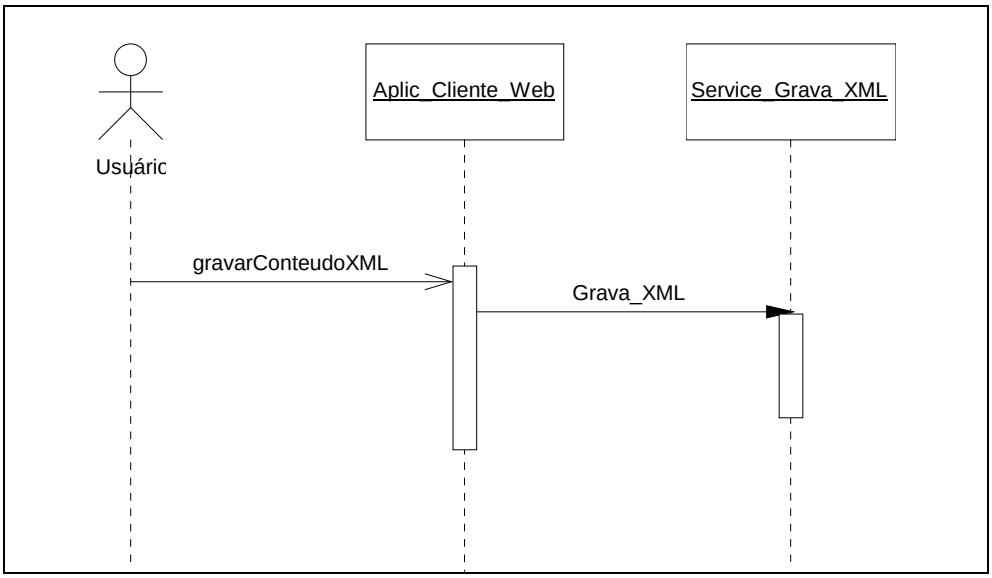


Figura 13 – Diagrama de seqüência *Gravar arquivo XML*

3.2.2.2 Casos de uso: servidor de banco de dados

Os casos de uso mostrados na Figura 14 demonstram a especificação do protótipo através do servidor de banco de dados na realização da leitura das informações no arquivo XML e atualização da base de dados no servidor de banco de dados.

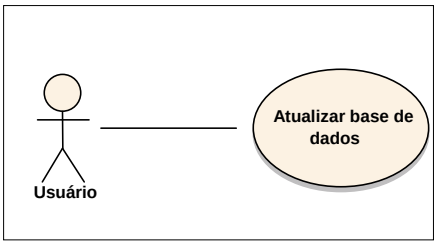


Figura 14 – Casos de uso: servidor de banco de dados

O caso de uso *Atualizar base de dados* do servidor de banco de dados, onde as informações são passadas do arquivo XML e atualizadas no banco de dados de acordo com as referidas tabelas e campos conforme gravados no arquivo XML. A Figura 15 mostra o diagrama de seqüência *Atualizar base de dados*.

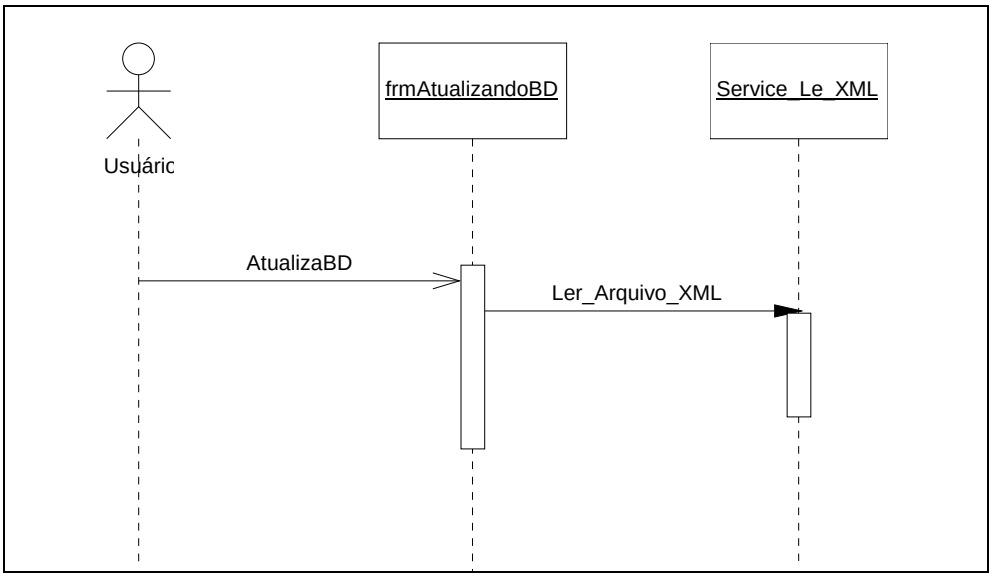


Figura 15 – Diagrama de seqüência *Atualizar base de dados*

3.3 IMPLEMENTAÇÃO

Nesta seção são discutidos todos os aspectos relacionados ao desenvolvimento do protótipo. São demonstrados todos os aspectos técnicos referentes à aplicação cliente, ao servidor de *Web Services* e ao servidor de banco de dados.

3.3.1 Técnicas e ferramentas utilizadas

Para o desenvolvimento do protótipo, tanto para a aplicação cliente, quanto para o servidor de *Web Services* e o servidor de banco de dados, foi utilizada a linguagem de programação *Visual Basic .Net* (VB .Net), sobre a plataforma .NET.

A aplicação cliente consiste em uma aplicação *Web Form* desenvolvida em VB .Net no formato *Active Server Page* (ASP), sendo disponibilizada e acessada em uma rede de computadores via *web*. A interface da aplicação consiste em uma tela para informações dos dados que são descritos abaixo:

a) *Provedor de Serviços Web*: representa o local onde está localizado o servidor dos

serviços *web*;

b) *demais campos da tela*: informações que serão preenchidas e enviadas para serem gravadas de forma estruturada no arquivo XML.

O item *a* descrito acima é apresentado em detalhes na seção 3.3.1.1.

Na implementação do servidor de *Web Services* foi utilizado o servidor *Internet Information Server* (IIS), responsável por disponibilizar as classes que fornecem os serviços *web* e gerenciar a troca de mensagens. Os *Web Services* consistem em classes em VB .Net que contém métodos que são invocados através de mensagens SOAP enviadas pela aplicação cliente e pelo servidor de banco de dados. Os métodos do servidor de *Web Services* de criação, gravação, verificação e leitura do arquivo XML são apresentados em detalhes nas seções 3.3.1.2 à 3.3.1.5 respectivamente.

Ao receber uma requisição SOAP, seja da aplicação cliente ao gravar as informações no arquivo XML (Apêndice A) ou do servidor de banco de dados ao ler as informações do arquivo XML (Apêndice B), o servidor de *Web Services* libera o acesso e executa o método que está implícito no corpo da mensagem.

O servidor de banco de dados é uma aplicação cliente baseada em *Windows Form*, e utiliza o banco de dados MSDE para armazenamento e consulta das informações. Na Figura 16 é mostrado o modelo físico de dados que contém as tabelas utilizadas pelo protótipo.

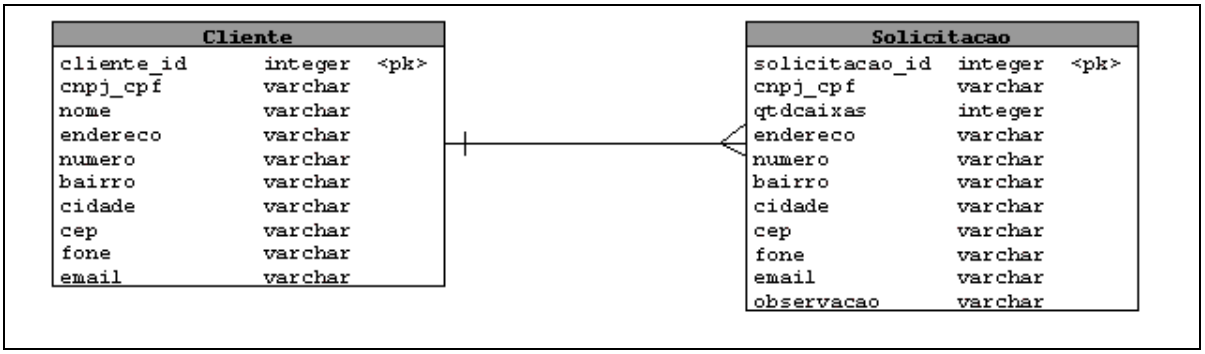


Figura 16 – Modelo físico de dados utilizado pelo protótipo

A Figura 16 representa as tabelas *Cliente* e *Solicitacao* com os respectivos campos que estão criados no servidor de banco de dados e serão atualizados a partir do arquivo XML após

o preenchimento destas informações pela aplicação cliente.

O servidor de banco de dados permite alterar o local do servidor de *Web Services* e realiza atualizações na base de dados através da invocação dos métodos do servidor de *Web Services*. Os métodos do servidor de banco de dados para configurar o local do servidor de *Web Services* e atualizar a base de dados de forma assíncrona são apresentados em detalhes nas seções 3.3.1.6 e 3.3.1.7 respectivamente.

3.3.1.1 Provedor de Serviços *Web*

Ao criar um *Web Service* e o referenciar em uma aplicação cliente para *web* (ASP.NET), é definido um local, ou seja, uma *Uniform Resource Locator* (URL) no arquivo de configurações (*Web.config*), onde fica descrito o endereço do serviço *web*, conforme Quadro 3.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
   <appSettings>
      <add key="Web_Aplicacao_Cliente.localhost.Service_Grava_XML"

value="http://localhost/WebServiceGravaXML/Service_Grava_XML.asmx"/>
   </appSettings>
</configuration>
```

Quadro 3 – Arquivo de configurações *Web.Config*

Quando se faz a referência ao serviço *web* pela aplicação cliente, uma classe *proxy* herdada de *SoapHttpClientProtocol* é criada. Esta classe fornece uma propriedade chamada *URL*, que armazena a URL do *Web Service* que o cliente está invocando. No momento desta referência, o *proxy* gerado define a URL baseando-se no que está definido no arquivo de configurações.

No *Web Service* tem-se uma propriedade chamada *URLBehavior* que, ao defini-la como *Dynamic*, automaticamente é criada uma chave dentro do arquivo *Web.config* contendo a URL do *Web Service* conforme mostrado no Quadro 3. O construtor da classe *proxy* que

herda de *SoapHttpClientProtocol* é alterado para buscar a URL *Web Service* dentro do arquivo de configurações da aplicação cliente.

Ao acessar a aplicação cliente, aparece um *label* com a descrição *Provedor de Serviços Web* na tela principal e um *textBox* ao lado com o endereço do serviço *web*. O Quadro 4 mostra o código que carrega este endereço na tela ao ser acessado a aplicação cliente.

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
  'Cria e instancia um objeto da classe Web Service
   Dim wsAtual As New Service_Grava_XML
  'Carrega o campo com o endereço (URL) do serviço web caso ainda não foi
carregado
   If txtProvedorWS.Text = "" Then
      txtProvedorWS.Text = wsAtual.Url
   End If
End Sub
```

Quadro 4 – URL atual do provedor de serviços *web* na tela da aplicação cliente

3.3.1.2 Método remoto *Cria_arquivo_XML*

O método *Cria_arquivo_XML* é invocado remotamente pela aplicação cliente e é responsável por criar o arquivo no formato XML, caso o mesmo ainda não esteja criado. Recebe como parâmetros de entrada o nome do arquivo e o elemento pai, necessários para a criação e formatação da estrutura principal do arquivo XML conforme mostrado no Quadro 5.

```
<WebMethod()> _
Public Function Cria_arquivo_XML(ByVal nomeArquivo As String, ByVal
elementoPai As String) As Boolean
  Dim arquivo As String = nomeArquivo
  Dim elementPai As String
  elementPai = elementoPai
  If Not System.IO.File.Exists(Server.MapPath("/" & arquivo)) Then
     Try
         Dim xmlArq As New XmlTextWriter(Server.MapPath("/" &
arquivo), System.Text.Encoding.ASCII)
         xmlArq.Formatting = Formatting.Indented

         'Escrever o início do documento XML
         xmlArq.WriteStartDocument(True)

         'Escrever o elemento raiz
         xmlArq.WriteStartElement(elementPai)

         'Escrever o Fim do documento XML
         xmlArq.WriteEndDocument()
         xmlArq.Close()
     Catch ex2 As Exception
         Return False
     End Try
  End If
  Return True
End Function
```

Quadro 5 – Método remoto *Cria_arquivo_XML*

Além dos parâmetros enviados pela aplicação cliente, deve-se criar o novo arquivo considerando atributos referentes ao tipo de texto, formato, entre outros. O Quadro 6 mostra o conteúdo do arquivo XML ao ser criado.

```
<?xml version="1.0" encoding="us-ascii" standalone="yes"?>
<Solicitacoes />
```

Quadro 6 – Conteúdo do arquivo XML ao ser criado

3.3.1.3 Método remoto *Grava_XML*

O método *Grava_XML* é invocado remotamente pela aplicação cliente e é responsável por gravar as informações no arquivo XML. Recebe como parâmetros de entrada o nome do arquivo e o conteúdo, conforme mostrado no Quadro 7.

```
<WebMethod()> _
Public Function Grava_XML(ByVal nomeArquivo As String, ByVal conteudoXML As
String) As Boolean

  Dim arquivoXML As String = nomeArquivo
  Dim conteudo As String = ""
  Dim i As Integer
  Dim x As Integer
  Dim ultPosicao As Integer
  Dim nroElementos As Integer = 0
  Dim auxString As String
  Dim elementoPaiXML As String
  Dim elementoFilhoXML As String
  Dim elementoXMLinner As String

  'Retira os espaços no início e fim do conteudo passado
  conteudo = conteudoXML.Trim()

  'Busca tamanho do conteúdo passado e cria array com este tamanho
  Dim tamanho As Integer = Len(conteudo)
  Dim dadosXML() As String = New String(tamanho) {}

  'Carrega array com o conteúdo passado
  For i = 0 To (tamanho - 1)
      dadosXML(i) = conteudo.Substring(i, 1)
  Next
  Try
      'Cria um novo documento XML
      Dim xmlDoc As New XmlDocument

      'Carrega arquivo XML
      xmlDoc.Load(Server.MapPath("/" & arquivoXML))

      'Cria elemento e item
      Dim xmlElement As XmlElement
      Dim xmlItem As XmlElement

      'Posiciona no último elemento
      xmlElement = xmlDoc.LastChild

      'Fica em loop separando dados(elementos/atributos) até fim do conteúdo
enviado
      For i = 0 To (tamanho - 1)
          elementoPaiXML = ""
          elementoFilhoXML = ""
          elementoXMLinner = ""
          If UCase(dadosXML(i)) = "T" Then
             auxString = ""
             ultPosicao = i + 9
             For x = i To ultPosicao
                 auxString = auxString & dadosXML(x)
             Next
             If auxString <> "" Then
                i = x
             End If
             If UCase(auxString) = "TABELA_BD " Then
                'Se existir mais de uma tabela(considerado como um elemento),
                'antes de criar novo elemento Pai(nova tabela), insere elemento
                nroElementos += 1
                If nroElementos > 1 Then
                  'Insere novo elemento
                  xmlDoc.DocumentElement.AppendChild(xmlElement)
                End If
                Do While dadosXML(i) <> ";"
                   elementoPaiXML = elementoPaiXML & dadosXML(i)
                   i += 1
                   If i = tamanho Then
                      Exit Do
```

```
                  End If
               Loop
               'Cria novo elemento
               xmlElement = xmlDoc.CreateElement(elementoPaiXML)
               i += 1
               If i >= tamanho Then
                  Exit For
               End If
            End If
         End If
         Do While dadosXML(i) <> ";"
            elementoFilhoXML = elementoFilhoXML & dadosXML(i)
            i += 1
            If i = tamanho Then
               Exit Do
            End If
         Loop
         If dadosXML(i) = ";" Then
            i += 1
         End If
         Do While dadosXML(i) <> ";"
            elementoXMLinner = elementoXMLinner & dadosXML(i)
            i += 1
            If i = tamanho Then
               Exit Do
            End If
         Loop
         'Cria novos elementos filhos e seus atributos
         xmlItem = xmlDoc.CreateElement(elementoFilhoXML)
         xmlItem.InnerText = elementoXMLinner
         xmlElement.AppendChild(xmlItem)
      Next
      'Insere novo elemento
      xmlDoc.DocumentElement.AppendChild(xmlElement)

      'salva o arquivo XML
      xmlDoc.Save(Server.MapPath("/" & arquivoXML))
   Catch ex As Exception
         Return False
   End Try
   Return True
End Function
```

Quadro 7 – Método remoto *Grava_XML*

O método *Grava_XML* é responsável por receber o conteúdo, que está concatenado e separado por ponto e vírgula (;), abrir o arquivo XML, posicionar no último elemento e criar novos elementos filhos de acordo com o conteúdo passado. No Quadro 8 é mostrado um exemplo do conteúdo do arquivo XML após gravar uma solicitação através da aplicação cliente.

```
<?xml version="1.0" encoding="us-ascii" standalone="yes"?>
<Solicitacoes>
  <Cliente>
    <CNPJ_CPF>90187334900</CNPJ_CPF>
    <Nome>Sergio Koch Van-Dall</Nome>
    <Endereco>Rua Pinheiro Machado</Endereco>
    <Numero>72</Numero>
    <Bairro>Vila Nova</Bairro>
    <Cidade>Blumenau</Cidade>
    <CEP>89032272</CEP>
    <Fone>3323-0867</Fone>
    <Email>sergiod@inf.furb.br</Email>
  </Cliente>
  <Solicitacao>
    <CNPJ_CPF>90187334900</CNPJ_CPF>
    <QtdCaixas>2</QtdCaixas>
    <Endereco>Rua Pinheiro Machado</Endereco>
    <Numero>78</Numero>
    <Bairro>Vila Nova</Bairro>
    <Cidade>Blumenau</Cidade>
    <CEP>89035272</CEP>
    <Fone>3323-0999</Fone>
    <Email>svandall@gmail.com</Email>
    <Observacao>Colocar em cima da calcada, no lado esquerdo do portao da
garagem.</Observacao>
  </Solicitacao>
</Solicitacoes>
```

Quadro 8 – Conteúdo do arquivo XML após gravação da solicitação

O arquivo XML mostrado no Quadro 8 é dividido em dois elementos filhos e seus respectivos atributos e valores, que são respectivamente as tabelas e os campos já existentes no servidor de banco de dados.

3.3.1.4 Método remoto *Verifica_arquivo_XML*

O método *Verifica_arquivo_XML* é invocado remotamente pelo servidor de banco de dados e é responsável por verificar a existência do arquivo XML. Recebe como parâmetro de o nome do arquivo, conforme mostrado no Quadro 9.

```
<WebMethod()> _
Public Function Verifica_arquivo_XML(ByVal nomeArquivo As String) As
Boolean
  Dim arquivo As String = nomeArquivo
  If Not System.IO.File.Exists(Server.MapPath("/" & arquivo)) Then
     Return False
  End If
  Return True
End Function
```

Quadro 9 – Método remoto *Verifica_arquivo_XML*

3.3.1.5 Método remoto *Ler_Arquivo_XML*

O método *Ler_Arquivo_XML* é invocado remotamente pelo servidor de banco de dados e é responsável por ler as informações no arquivo XML. Recebe como parâmetros de entrada o nome do arquivo e o *status* da gravação no banco de dados, recebendo também como parâmetros de entrada e saída (por referência) o nome, os campos e os valores dos campos da tabela a ser atualizada, conforme mostrado no Quadro 10.

```
<WebMethod()> _
Public Function Ler_Arquivo_XML(ByVal nomeArquivo As String, ByVal stGravacaoBD As
String, ByRef nomeTabela As String, ByRef camposTabela As String, ByRef
vlCamposTabela As String) As Boolean

  Dim arquivoXML As String = nomeArquivo
  'Cria variável para receber o arquivo XML
  Dim xmlDoc As New XmlDocument
  'Carrega o arquivo XML conforme caminho e nome informado
  xmlDoc.Load(Server.MapPath("/" & arquivoXML))
  'Cria elemento que os dados da tabela
  Dim xmlElement As XmlElement
  'Cria uma instância XmlElement na qual atribuímos a raiz do documento
  xmlElement = xmlDoc.DocumentElement

  If stGravacaoBD = "OK" Then
     xmlElement.RemoveChild(xmlElement.ChildNodes.Item(0))
     xmlDoc.Save(Server.MapPath("/" & arquivoXML))
  Else
     If stGravacaoBD = "NOK" Then
        Return False
     End If
  End If
  'Percorre todos os elementos filhos(tabelas) que existem no elemento raiz
  Dim i As Integer = 0
    Do While i < xmlElement.ChildNodes.Count
       nomeTabela = xmlElement.ChildNodes.Item(i).Name
       'Percorre todos os elementos(campos) dentro do elemento filho atual
       Dim a As Integer = 0
       Do While a < xmlElement.ChildNodes.Item(i).ChildNodes.Count
          If Not camposTabela.ToString.Equals(String.Empty) Then
             camposTabela = camposTabela & ","
          End If
          If Not vlCamposTabela.ToString.Equals(String.Empty) Then
             vlCamposTabela = vlCamposTabela & ","
          End If
          camposTabela = camposTabela &
                         xmlElement.ChildNodes.Item(i).ChildNodes.Item(a).Name
          'Busca os valores dos elementos(campos) dentro do filho atual
          vlCamposTabela = vlCamposTabela & "'" &
          xmlElement.ChildNodes.Item(i).ChildNodes.Item(a).InnerText & "'"
          a += 1
       Loop
       Return True
   Loop
   Return False
End Function
```

Quadro 10 – Método remoto *Ler_Arquivo_XML*

O método *Ler_Arquivo_XML* é responsável por carregar o arquivo XML e percorrer todo o conteúdo do mesmo, separando os elementos correspondentes ao nome da tabela, campos da tabela e seus respectivos valores. Enviá-los para o servidor de banco de dados e caso os dados tenham sido gravados com sucesso, eliminar o elemento correspondente do arquivo XML e continuar processando desta forma os demais elementos, até o fim do arquivo.

3.3.1.6 Método *ConfigurarWS*

Ao criar um *Web Service* e o referenciar em uma aplicação cliente, é definido um local, ou seja, uma *Uniform Resource Locator* (URL) no arquivo de configurações (*App.config*), onde fica descrito o endereço do serviço *web*, conforme Quadro 11.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
   <appSettings><add key="AplicacaoClienteBD.localhost.Service_Le_XML"
value="http://localhost/WebServiceLeXML/Service_Le_XML.asmx"/>
   </appSettings>
</configuration>
```

Quadro 11 – Arquivo de configurações *App.Config*

Ao distribuir uma aplicação, onde se tinha o arquivo de configuração *App.config*, o mesmo é regravado com o mesmo nome do arquivo executável com a extensão adicional *.config*.

O método *ConfigurarWS* é chamado pelo servidor de banco de dados e é responsável por ler o arquivo de configurações e trocar o local do servidor de serviços *web* em tempo de execução, a partir do momento que é informado o novo local e clicado no botão *Alterar* da aplicação cliente, conforme Quadro 12.

```
Private Sub ConfigurarWS()
  Try
    'Cria e instancia documento XML
    Dim xmlDoc As New XmlDocument
    'Cria Nodo XML
    Dim xmlNodo As XmlNode

    'Cria e atribui nome do arquivo referente às configurações
    Dim nomeArquivo As String =
        System.Reflection.Assembly.GetExecutingAssembly().GetName().CodeBase +
".config"
    'Carrega arquivo
    xmlDoc.Load(nomeArquivo)

    'Busca/altera caminho do provedor de WS conforme informado em tela
    For Each xmlNodo In xmlDoc("configuration")("appSettings")
      If (xmlNodo.Name = "add") Then
         If (xmlNodo.Attributes.GetNamedItem("key").Value =
            "AplicacaoClienteBD.localhost.Service_Le_XML") Then
            xmlNodo.Attributes.GetNamedItem("value").Value =
            txtNovoProvedorWS.Text
         End If
      End If
    Next xmlNodo

    'Salva o arquivo com o conteúdo novo
    xmlDoc.Save([Assembly].GetExecutingAssembly.Location + ".config")
    Catch ex As Exception
        MsgBox("Verifique - Problemas ao atualizar Local do Provedor de Serviços
Web !!!", MsgBoxStyle.Information)
        txtNovoProvedorWS.Focus()
    End Try
End Sub
```

Quadro 12 – Método *ConfigurarWS*

O Quadro 12 apresenta o código responsável por alterar as informações em tempo de execução do arquivo de configurações referentes à URL onde encontra-se o serviço *web* que se deseja referenciar e consumir seus serviços. O arquivo de configurações é um arquivo XML que possui parâmetros necessários para a execução da aplicação cliente, sendo que para efetuar a alteração deste, deve-se criar uma instância para o documento XML e carregar suas informações, sendo que o mesmo é percorrido até encontrar o parâmetro que contém o valor da URL atual, alterá-lo para o novo endereço, e, por fim, gravar novamente o arquivo.

3.3.1.7 Método *AtualizaBD*

O método *AtualizaBD* é chamado pelo servidor de banco de dados e é responsável por atualizar de forma assíncrona a base de dados com as informações contidas no arquivo XML,

conforme mostrado no Quadro 13.

```
Private Sub AtualizaBD()
  'Declara variável que verifica status da gravação no BD
  Dim stGravacaoBD As String = ""
  'Declara variáveis necessárias para buscar dados ao arquivo XML
  Dim nomeArquivo As String
  nomeArquivo = "Solicitacoes.xml"
  'Instancia objeto Web Service
  Dim objWebService As New Service_Le_XML

  If objWebService.Verifica_arquivo_XML(nomeArquivo) Then
     Dim nomeTabela As String = ""
     Dim camposTabela As String = ""
     Dim vlCamposTabela As String = ""

     Do While objWebService.Ler_Arquivo_XML(nomeArquivo, stGravacaoBD, nomeTabela,
camposTabela, vlCamposTabela)
        SetStatus(statusAtualizaBD.statusAtualizando)
        'Cria string do Comando Insert com os valores retornados da leitura do
         arquivo XML
        Dim sSQL As New StringBuilder
        'Cria string para verificar se existem informações para atualizar base de
dados
        Dim semVirgulas As String = vlCamposTabela.ToString.Replace(",", Nothing)
        Dim valoresTabela As String = semVirgulas.ToString.Replace("'", Nothing)
        'Verifica se tem valores nos dados retornados para atualizar base de dados
        If Not valoresTabela.Equals(String.Empty) Then
           sSQL.Append(" INSERT INTO ")
           sSQL.Append(nomeTabela)
           sSQL.Append(" ( ")
           sSQL.Append(camposTabela)
           sSQL.Append(" ) ")
           sSQL.Append("VALUES ( ")
           sSQL.Append(vlCamposTabela)
           sSQL.Append(" );")
           'Declara Classe do Banco de dados
           Dim cldBancoDados As New cldBancoDados
           Try
               cldBancoDados.ExecutaComando(sSQL.ToString)
               stGravacaoBD = "OK"
               nomeTabela = ""
               camposTabela = ""
               vlCamposTabela = ""
           Catch ex As Exception
               stGravacaoBD = "NOK"
               nomeTabela = ""
               camposTabela = ""
               vlCamposTabela = ""
           End Try
        Else
           stGravacaoBD = "OK"
           nomeTabela = ""
           camposTabela = ""
           vlCamposTabela = ""
        End If
     Loop
     SetStatus(statusAtualizaBD.statusOK)
  End If
  If stGravacaoBD.ToString = "" Then
     MsgBox("Atualização da Base de Dados efetuada com Sucesso !!!",
MsgBoxStyle.Information)
  Else
      If stGravacaoBD.ToString = "OK" Then
         MsgBox("Atualização da Base de Dados efetuada com Sucesso !!!",
MsgBoxStyle.Information)
      Else
```

```
         MsgBox("Atualização da Base de Dados efetuada com Erro !!!",
MsgBoxStyle.Information)
      End If
   End If
End Sub
```

Quadro 13 – Método *AtualizaBD*

O Quadro 13 apresenta o código responsável pela leitura das informações no arquivo XML e a atualização da base de dados no servidor de banco de dados. Localiza o arquivo XML e processa todo o conteúdo do mesmo, lendo uma informação de cada vez, referente ao elemento e seus atributos, que são o nome da tabela com os respectivos campos e valores, montando o comando *INSERT* para a inserção no banco de dados MSDE, sendo que, ao efetuar a inclusão do registro com sucesso, exclui o elemento com os seus respectivos atributos do arquivo XML.

3.3.2 Operacionalidade do protótipo

O processo de atualização assíncrona de dados utilizando *Web Services*, implementado neste trabalho, é dividido em quatro etapas distintas:

a) preencher formulário: onde a aplicação cliente permite o preenchimento de todas as informações que serão gravadas no arquivo XML;
b) configurar servidor de banco de dados: onde o aplicativo servidor de banco de dados disponibiliza e permite alterar o endereço do servidor de *Web Services*;
c) atualizar base de dados: onde o servidor de banco de dados atualiza seus dados de forma assíncrona com as informações do arquivo XML;
d) consultar base de dados: onde o usuário irá consultar o servidor de banco de dados e verificar as solicitações armazenadas em sua base.

Para exemplificar a funcionalidade das partes integrantes do sistema, o usuário acessará a aplicação cliente e realizará a solicitação de caixas para recolher entulhos para um

cliente ainda não cadastrado, efetuando o envio desta solicitação para gravação no arquivo XML. O servidor de banco de dados irá efetuar a atualização da sua base com as informações do arquivo XML e então, consultar as solicitações e verificar que a mesma está armazenada em sua base de dados.

3.3.2.1 Etapa “Preencher formulário”

O usuário ao acessar a aplicação cliente deverá preencher as informações para a solicitação de caixas para recolher entulhos, de acordo com a opção de ser ou não cliente, sendo este parâmetro necessário para o preenchimento ou não dos dados para o seu cadastro. Os demais campos são obrigatórios e serão consistidos ao clicar no botão *Enviar Solicitação*. A Figura 17 mostra a solicitação para colocação de 3 caixas por um usuário que ainda não é cliente.

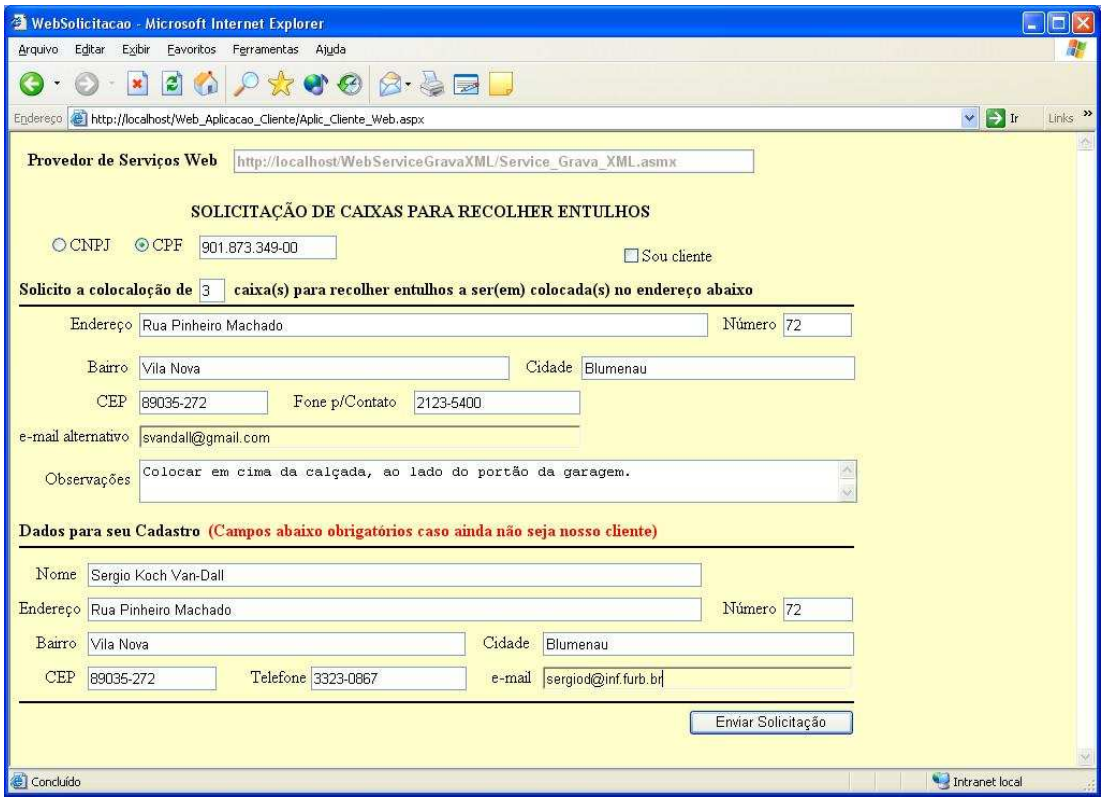


Figura 17 – Etapa *Preencher formulário*

Após clicar no botão *Enviar Solicitação* serão efetuadas as respectivas validações dos

campos e caso não haja inconsistência, serão inicializados todos os campos do formulário, aguardando para efetuar uma nova solicitação.

3.3.2.2 Etapa “Configurar servidor de banco de dados”

Para configurar o local do servidor de serviços *web*, o usuário deve acessar o módulo principal *Configurações* e então o módulo *Local Servidor Web*, conforme Figura 18.

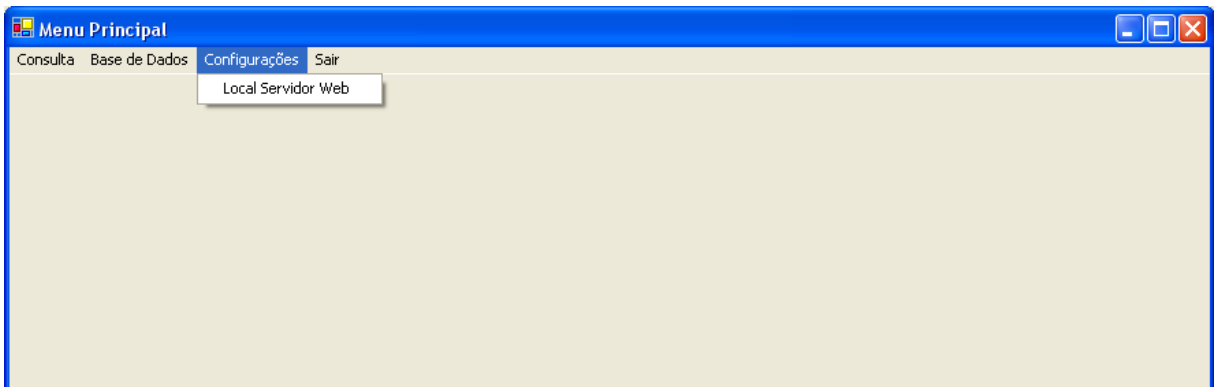


Figura 18 – Etapa *Configurar servidor de banco de dados*

A tela para configuração do local do servidor de serviços *web* possui s seguintes parâmetros:

a) *Provedor de Serviços Web Atual*: endereço onde está localizado o servidor padrão de *Web Services*. Por exemplo o endereço *http://localhost/WebServiceLeXML/Service_Le_XML.asmx* representa o local onde se encontra o serviço *web* responsável por verificar a existência do arquivo e ler as informações no arquivo XML;

b) *Novo Provedor de Serviços Web*: novo local do servidor de serviços web.

A Figura 19 mostra o parâmetro do servidor de serviços *web* atual e a eventual troca deste endereço pelo usuário nesta etapa.

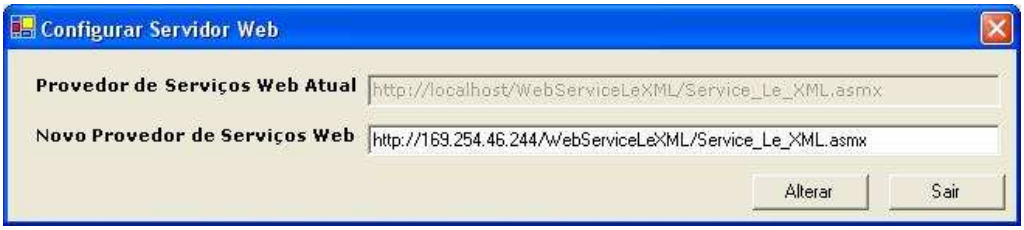


Figura 19 – Troca do endereço do servidor de serviços *web*

3.3.2.3 Etapa “Atualizar banco de dados”

Para atualizar a banco de dados com as informações contidas no arquivo XML, deve-se acessar o aplicativo cliente servidor de banco de dados, acessar o módulo principal *Base de Dados* e acessar então o módulo *Atualizar Dados*. São iniciados assim, o processo de leitura das informações do arquivo XML e a atualização da base de dados. A Figura 20 mostra a tela do aplicativo servidor de banco de dados e o acesso ao módulo de atualização dos dados.

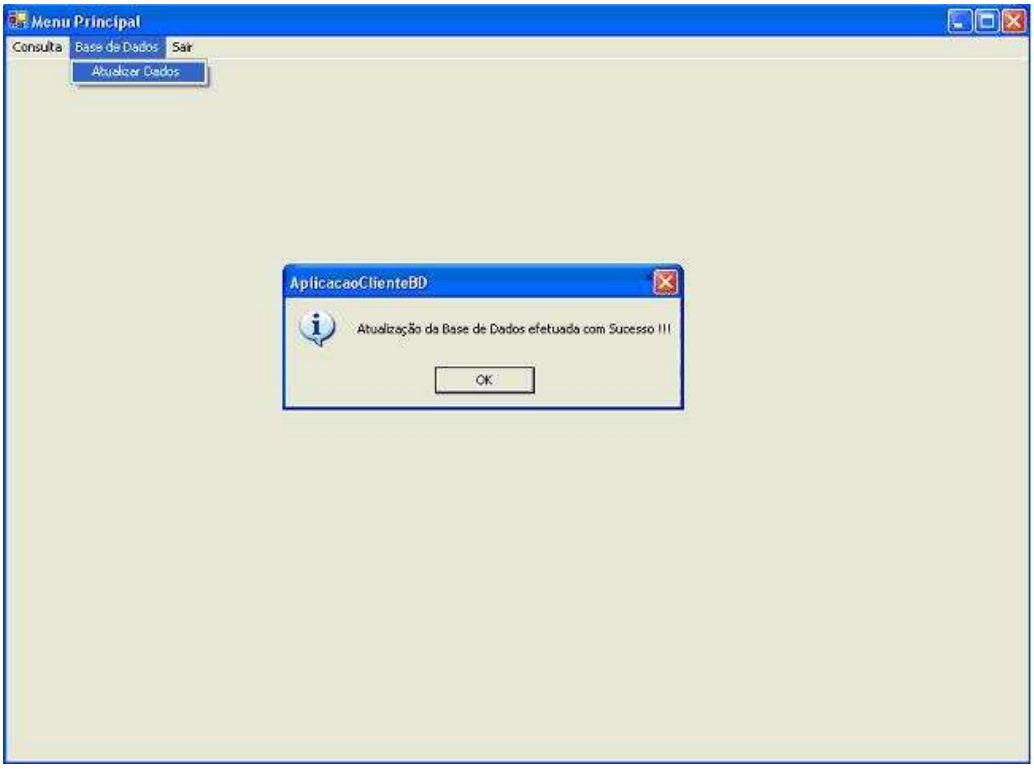


Figura 20 – Etapa *Atualizar Base de Dados*

3.3.2.4 Etapa “Consultar base de dados”

As consultas às informações das solicitações efetuadas pelos usuários na aplicação cliente estão disponíveis no servidor de banco de dados no módulo principal de *Consulta*, divididos em três módulos:

a) *Clientes*: mostra a lista de todos os clientes cadastrados no banco de dados;

b) *Solicitações*: mostra a lista de todas as solicitações cadastradas no banco de dados;

c) *Cliente X Solicitações*: mostra o cliente de acordo com o CNPJ_CPF informado e todas as solicitações efetuadas pelo mesmo que estão cadastradas no banco de dados.

A Figura 21 mostra o acesso ao módulo principal *Consultar* e os seus respectivos módulos.

Figura 21 – Etapa *Consultar base de dados*

A Figura 22 mostra o resultado de uma consulta no módulo *Cliente X Solicitações* através do CNPJ_CPF do cliente informado.

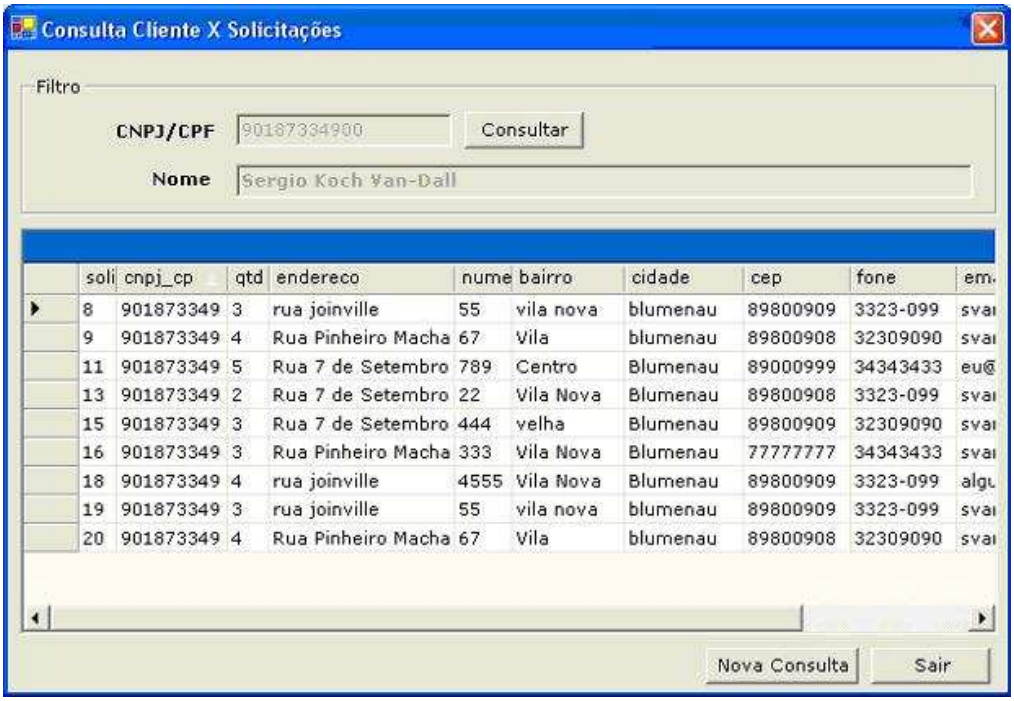


Figura 22 – Resultado de uma consulta no módulo *Cliente X Solicitações*

3.4 RESULTADOS E DISCUSSÃO

O protótipo implementado é similar aos trabalhos correlatos apresentados, onde utiliza a linguagem XML como meio de compartilhar e realizar a troca de informações entre aplicações distribuídas, disponibilizando métodos remotos em um *contêiner* HTTP, através de um servidor de serviços *web*, onde estes métodos são invocados utilizando o protocolo SOAP.

O protótipo tem a capacidade de realizar o envio de informações de forma estruturada para um arquivo no formato XML através de uma aplicação *web*, disponível em um local remoto comum entre as aplicações, permitindo a leitura destas informações e atualização da base de dados de forma assíncrona, através de um aplicativo servidor de banco de dados.

A linguagem XML em conjunto com o protocolo SOAP pode integrar sistemas distintos mais facilmente se comparado com outros padrões de integração existentes atualmente, que se baseiam em tecnologias com maior nível de complexidade de implementação.

Este protótipo forneceu suporte adequado à atualização das informações contidas no arquivo XML em um banco de dados SQL *Server* (MSDE), podendo ser facilmente substituído por outros, tais como, *MySQL*, *Oracle*, *Interbase* entre outros.

Este trabalho pode ser visto como um *middleware* responsável por disponibilizar um arquivo no formato XML, que pode ser utilizado tanto por uma aplicação que deseja gravar, quanto por uma aplicação que deseja ler informações, podendo assim, efetuar a integração de sistemas distintos através de um ambiente distribuído, utilizando um formato padrão na comunicação e troca de informações entre os mesmos.

O que mais dificultou a realização deste trabalho foi a falta de experiência em trabalhar com XML e o não conhecimento da linguagem de programação VB .NET e das funcionalidades da plataforma .NET, que tiveram que serem vistas no decorrer do trabalho.

Algumas limitações existentes no protótipo:

a) requer o .NET Framework SDK;

b) rodar somente no servidor IIS.

4 CONCLUSÕES

Este trabalho alcançou na íntegra seu objetivo principal, que é a implementação de um protótipo para atualização assíncrona de dados utilizando *Web Services*, tendo como cenário uma aplicação cliente, um servidor de *Web Services* e um servidor de banco de dados.

A troca de informações com o uso de um arquivo no formato XML e a comunicação entre as aplicações distribuídas no cenário composto através do protocolo SOAP utilizando o servidor IIS, responsável por disponibilizar as classes que fornecem os serviços *web* e gerenciar a troca de mensagens, demonstrou uma funcionalidade satisfatória.

O desenvolvimento de um protótipo baseado em XML e .NET é sem dúvida muito interessante, pois além de conhecer funcionalidades do XML, também foi possível descobrir e utilizar uma plataforma de desenvolvimento que oferece suporte a múltiplas linguagens de programação, através da utilização do ASP.NET e do VB .NET.

4.1 EXTENSÕES

Segue algumas sugestões para trabalhos futuros:

a) possibilitar a criação de novas tabelas no servidor de banco de dados a partir do arquivo XML, incluindo assim, além do nome, campos e valores das tabelas, o tipo de dados;
b) efetuar a alteração dos dados já inseridos na tabela no servidor de banco de dados a partir do arquivo XML;
c) utilizar outras linguagens de programação para a aplicação *web*, como PHP ou JSP, possibilitando assim, o uso de outros servidores, como o *Apache* ou o *Tomcat*.

REFERÊNCIAS BIBLIOGRÁFICAS

ALEXANDER, John; HOLLIS, Billy S. **Desenvolvendo aplicações web com Visual Basic.NET e ASP.NET.** Tradução Bázan Tecnologia e Lingüística. São Paulo: Berkeley Brasil, 2002. 342 p.

BECKER, Aleksader K.; CLARO, Daniela B.; SOBRAL, João B. **Web Services e XML:** um novo paradigma da computação distribuída. Florianópolis, 2004. Disponível em: < http://www.inf.ufsc.br/~danclaro/download/ArtigoWebServices.pdf>. Acesso em: 27 fev. 2006.

BENTO, Fabrício. **Protótipo de protocolo de aplicação para troca de documentos da área extrajudicial.** 2005. 88 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

COLPANI, Cristiano F. **Prototipo de software para troca de dados entre aplicações de comércio eletrônico utilizando o protocolo SOAP.** 2002. 82 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

LIMA, Edwin; REIS, Eugênio. **C# e .NET:** guia do desenvolvedor. Rio de Janeiro: Campus, 2002. 358 p.

SKONNARD, Aaron.; GUDGIN, Martin. **Essential XML quick reference.** Indianápolis: Pearson-Education, 2002. 403 p.

SNELL, James.; TIDWELL, Doug.; KULCHENKO, Pavel. **Programming web services with SOAP.** Beijing: O´Reilly, 2002. 244 p.

SOUZA, Jackson G. Web services na plataforma .NET. In: Encoinfo, 5., 2003, Palmas. Mini cursos. **Anais eletrônicos...**. Palmas: CEULP/ULBRA, 2003. Disponível em: <http://www.Ulbra-to.br/ensino/43020/artigos/anais2003/anais/MiniCursos/NETXMLWeb%20Services.pdf>. Acesso em: 9 abr. 2006.

TARIFA, Alexandre; FACUNTE, Emerson; GARCIA, Marcus. **Visual Basic .NET:** desenvolvendo uma aplicação comercial. Rio de Janeiro: BRASPORT, 2005. 174p.

TURTSCHI, Adrian et al. **C# .NET:** guia do desenvolvedor Web, curso completo. 2. ed. Rio de Janeiro : Alta Books, 2004. 517 p.

WIECZOREK, Emilio M. **Sistema de gerenciamento de documentos jurídicos utilizando XML, DOM e plataforma .NET.** 2004. 100 f. Monografia (Prática em Sistemas de Informações II) – Centro de Ciências Exatas e Naturais, Centro Universitário Luterano de Palmas, Palmas.

APÊNDICE A – Requisição SOAP para o método Gravar_XML

Ao efetuar uma chamada remota ao método *Gravar_XML* no servidor de serviços *web*, é gerada uma requisição SOAP conforme Quadro 14, com as seguintes informações:

a) as primeiras 4 linhas do exemplo definem o cabeçalho HTTP, incluindo dados como o tamanho do pacote, o tipo dos dados transmitidos, etc;

b) um campo de cabeçalho obrigatório chamado *SOAPAction*, usado para informar o propósito da requisição HTTP SOAP. Seu valor é uma URI do objeto alvo que identifica o *namespace* utilizado por esta requisição. A URI http://schemas.xmlsoap.org/soap/envelope/ é o *namespace* padrão para todas as mensagens SOAP;

c) uma chamada ao método *Grava_XML* e seu respectivo *namespace*;

d) os elementos *nomeArquivo* e *conteudoArquivo* e seus respectivos tipos.

```
POST /WebServiceGravaXML/Service_Grava_XML.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/WebServiceGravaXML/Service1/Grava_XML"

<?xml version="1.0" encoding="utf-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/">
  <soap:Body>
    <Grava_XML xmlns="http://tempuri.org/WebServiceGravaXML/Service1">
      <nomeArquivo>string</nomeArquivo>
      <conteudoXML>string</conteudoXML>
    </Grava_XML>
  </soap:Body>
</soap:Envelope>
```

Quadro 14 – Requisição SOAP para o método *Gravar_XML*

APÊNDICE B – Requisição SOAP para o método Ler_Arquivo_XML

Ao efetuar uma chamada remota ao método *Ler_Arquivo_XML* no servidor de serviços *web*, é gerada uma requisição SOAP conforme Quadro 15, com as seguintes informações:

a) as primeiras 4 linhas do exemplo definem o cabeçalho HTTP, incluindo dados como o tamanho do pacote, o tipo dos dados transmitidos, etc;

b) um campo de cabeçalho obrigatório chamado *SOAPAction*, usado para informar o propósito da requisição HTTP SOAP. Seu valor é uma URI do objeto alvo que identifica o *namespace* utilizado por esta requisição. A URI http://schemas.xmlsoap.org/soap/envelope/ é o *namespace* padrão para todas as mensagens SOAP;

c) uma chamada ao método Ler_Arquivo_XML e seu respectivo namespace;

d) os elementos nomeArquivo, stHravacaoBD, nomeTabela, camposTabela e *vlCamposTabela* e seus respectivos tipos.

```
POST /WebServiceLeXML/Service_Le_XML.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction:"http://tempuri.org/WebServiceLeXML/Service1/Ler_Arquivo_XML"

<?xml version="1.0" encoding="utf-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/">
  <soap:Body>
    <Ler_Arquivo_XML
xmlns="http://tempuri.org/WebServiceLeXML/Service1">
      <nomeArquivo>string</nomeArquivo>
      <stGravacaoBD>string</stGravacaoBD>
      <nomeTabela>string</nomeTabela>
      <camposTabela>string</camposTabela>
      <vlCamposTabela>string</vlCamposTabela>
    </Ler_Arquivo_XML>
  </soap:Body>
</soap:Envelope>
```

Quadro 15 – Requisição SOAP para o método *Ler_Arquivo_XML*