

UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIAS DA COMPUTAÇÃO
(Bacharelado)

**PROTÓTIPO DE SOFTWARE PARA TROCA DE DADOS
ENTRE APLICAÇÕES DE COMÉRCIO ELETRÔNICO
UTILIZANDO O PROTOCOLO SOAP**

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE
REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA
DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA
COMPUTAÇÃO — BACHARELADO

CRISTIANO FORNARI COLPANI

BLUMENAU, JUNHO/2002

2002/1-18

PROTÓTIPO DE SOFTWARE PARA TROCA DE DADOS ENTRE APLICAÇÕES DE COMÉRCIO ELETRÔNICO UTILIZANDO O PROTOCOLO SOAP

CRISTIANO FORNARI COLPANI

ESTE TRABALHO DE CONCLUSÃO DE CURSO FOI JULGADO ADEQUADO
PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE
CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

Prof. Alexander Roberto Valdameri — Orientador na FURB

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

Prof. Alexander Roberto Valdameri

Prof. Marcel Hugo

Prof. Dalton Solano dos Reis

AGRADECIMENTO

A Deus que proporciona minha existência de forma saudável e me cerca de pessoas maravilhosas.

A meus pais, padrasto e avós que ao longo de toda a minha caminhada sempre estiveram ao meu lado incentivando e valorizando a busca pelo conhecimento.

A minha noiva e sua família, por seu carinho, compreensão e apoio nos momentos difíceis.

Ao professor, amigo e orientador Alexander Roberto Valdameri, por sua sabedoria, paciência e disponibilidade ao resolver as dificuldades das melhores maneiras.

Aos meus dois irmãos pelo incentivo e exemplo que sempre se mostraram pessoas justas.

Obrigado também a todos que me apoiaram direta e indiretamente neste caminhada pelo curso de Bacharelado em Ciências da Computação.

SUMÁRIO

AGRADECIMENTO	III
LISTA DE FIGURAS	VII
LISTA DE QUADROS	IX
LISTA DE ABREVIATURAS.....	XI
RESUMO	XII
ABSTRACT	XIII
1 INTRODUÇÃO.....	1
1.1 OBJETIVOS DO TRABALHO	3
1.2 ESTRUTURA DO TRABALHO	3
2 APLICAÇÕES DE COMÉRCIO ELETRÔNICO	4
2.1 ESTRUTURA DAS APLICAÇÕES DE E-COMMERCE.....	5
2.1.1 APLICAÇÕES B2B.....	6
2.1.2 APLICAÇÕES B2C.....	7
2.2 DIFICULDADES NO DESENVOLVIMENTO.....	7
3 EXTENSIBLE MARKUP LANGUAGE (XML)	9
3.1 ELEMENTOS	9
3.2 UNIFORM RESOURCE IDENTIFIER.....	10
3.3 NAMESPACES.....	11
3.4 ATRIBUTOS.....	12
3.5 SCHEMAS	12
4 PROTOCOLO SOAP	14
4.1 CONVENÇÕES	15
4.2 ESTRUTURA DA MENSAGEM SOAP.....	15

4.2.1 ENVELOPE	17
4.2.2 CABEÇALHO	17
4.2.3 CORPO	18
4.2.4 ERROS SOAP.....	19
4.3 SERIALIZAÇÃO	20
4.3.1 TIPOS SIMPLES	20
4.3.2 ENUMERAÇÕES.....	22
4.3.3 ARRAY DE BYTES.....	23
4.3.4 TIPOS COMPOSTOS DE DADOS	24
4.3.4.1 ESTRUTURAS.....	25
4.3.4.2 MATRIZES	25
4.3.5 REMOTE PROCEDURE CALL	26
4.4 WEB SERVICES	27
4.4.1 WSDL	27
4.4.2 UTILIZAÇÃO DE WEB SERVICES	27
5 DESENVOLVIMENTO DO PROTÓTIPO	29
5.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO.....	29
5.2 ESPECIFICAÇÃO	30
5.2.1 APLICAÇÃO PARA TROCA DE CATÁLOGOS	31
5.2.2 CASOS DE USO	33
5.2.2.1 PARAMETRIZAÇÃO DO SISTEMA.....	33
5.2.2.2 SOLICITAÇÃO E RECEBIMENTO DE CATÁLOGOS	35
5.2.2.3 ENVIO DE CATÁLOGOS	36
5.3 IMPLEMENTAÇÃO	38
5.3.1 TÉCNICAS E FERRAMENTAS UTILIZADAS.....	38

5.3.2 APLICAÇÃO PRINCIPAL	39
5.3.2.1 PARAMETRIZAÇÃO DO SISTEMA.....	45
5.3.2.2 SOLICITAÇÃO E RECEBIMENTO DE CATÁLOGOS	48
5.3.2.3 ENVIO DE CATÁLOGOS	48
5.3.3 OPERACIONALIDADE DA IMPLEMENTAÇÃO.....	51
5.3.3.1 CENÁRIO “PARAMETRIZAÇÃO DO SISTEMA”	52
5.3.3.2 CENÁRIO “SOLICITAÇÃO E RECEBIMENTO DE CATÁLOGOS”	55
5.3.3.3 CENÁRIO “ENVIO DE CATÁLOGOS”	57
6 CONSIDERAÇÕES FINAIS	60
6.1 CONCLUSÕES.....	60
6.2 DIFICULDADES ENCONTRADAS	61
6.3 EXTENSÕES	61
ANEXO 1 – CÓDIGO DO OBJETO DE ACESSO A INTERFACE ISERVICOPRODUTO	62
ANEXO 2– UNIT PARA ACESSO A INTERFACE.....	65
REFERÊNCIAS BIBLIOGRÁFICAS	67

LISTA DE FIGURAS

Figura 1 – ESQUEMA DE TRANSFERÊNCIA DE UMA PÁGINA WWW DINÂMICA.....	6
Figura 2 – AS APLICAÇÕES NÃO TEM RECURSOS PARA SE INTEGRAREM.....	8
Figura 3 – UM MODELO IDEAL	8
Figura 4 – DIAGRAMA DE BLOCO DA MENSAGEM SOAP	15
Figura 5 – DIAGRAMA DE CLASSES DA APLICAÇÃO.....	32
Figura 6 – CASO DE USO “PARAMETRIZAÇÃO DO SISTEMA”	33
Figura 7 – DIAGRAMA DE SEQUÊNCIA “PARAMETRIZAÇÃO DO SISTEMA”	34
Figura 8 – CASO DE USO “SOLICITAÇÃO E RECEBIMENTO DE CATÁLOGOS”	35
Figura 9 – DIAGRAMA DE SEQUÊNCIA “SOLICITAÇÃO E RECEBIMETO”	36
Figura 10 – CASO DE USO “ENVIO DE CATÁLOGOS”	37
Figura 11 – DIAGRAMA DE SEQUÊNCIA “ENVIA CATÁLOGO”	38
Figura 12 – INTERAÇÃO COM A APLICAÇÃO PRINCIPAL.....	39
Figura 13 – APLICAÇÃO SERVIDORA SOAP NO AMBIENTE DELPHI	40
Figura 14 – ERRO AO ACESSAR DIRETAMENTE A APLICAÇÃO	42
Figura 15 – LISTA DE INTERFACES DO PROTÓTIPO	43
Figura 16 – WSDL DA INTERFACE ISERVICOAREA	44
Figura 17 – MODELO DAS TELAS DE CADASTRO	45
Figura 18 – LAYOUT BÁSICO DA TELA DE CADASTRO.....	46
Figura 19 – WIZARD PARA GERAR OBJETO DE ACESSO.....	47
Figura 20 – WIZARD PARA GERAR INTERFACE DE ACESSO.....	49
Figura 21 – TELA PRINCIPAL DA APLICAÇÃO DE PARAMETRIZAÇÃO.....	52
Figura 22 – PARAMETRIZAÇÃO DE ÁREA.....	53
Figura 23 – PARAMETRIZAÇÃO DE SEÇÃO	54

Figura 24 – PARAMETRIZAÇÃO PRODUTO	55
Figura 25 – SOLICITAÇÃO DE CATÁLOGOS.....	56
Figura 26 – CATÁLOGOS RECEBIDOS	57
Figura 27 – APLICAÇÃO PARA ENVIO DE CATÁLOGOS	58
Figura 28 – LISTA DE SOLICITAÇÕES.....	58
Figura 29 – ESBOÇO DE ESTOQUE LOCAL	59
Figura 30 – SUCESSO NO ENVIO DE CATÁLOGO.....	59

LISTA DE QUADROS

Quadro 1 – ELEMENTO CODIFICADO EM XML	10
Quadro 2 – URI e URN	10
Quadro 3 – ELEMENTOS NÃO QUALIFICADOS	11
Quadro 4 – ELEMENTOS QUALIFICADOS	12
Quadro 5 – ATRIBUTOS	12
Quadro 6 – NAMESPACES E ATRIBUTOS	12
Quadro 7 – ENVELOPE SOAP	16
Quadro 8 – ELEMENTO ENVELOPE	17
Quadro 9 – MÉTODO SOMA	21
Quadro 10 – MÉTODO SOMA SERIALIZADO EM SOAP	21
Quadro 11 – MÉTODO VERIFICAPRESSAO	21
Quadro 12 – MÉTODO VERIFICAPRESSAO SERIALIZADO EM SOAP	22
Quadro 13 – RESPOSTA PARA O MÉTODO VERIFICA PRESSAO	22
Quadro 14 – ENUMERAÇÃO	23
Quadro 15 – BODY DA MENSAGEM SOAP GERADO	23
Quadro 16 – CARACTERES RESERVADOS QUE DEVEM SER CONVERTIDOS	23
Quadro 17 – CODIFICAÇÃO BASE64	24
Quadro 18 – MÉTODO PARA TRANSMITIR DADOS BINÁRIOS	24
Quadro 19 – ESTRUTURA TALUNO	25
Quadro 20 – MATRIZ SIMPLES	25
Quadro 21 – MATRIZ PARCIAL	26
Quadro 22 – MATRIZ ESPARSA	26
Quadro 23 – DEFINIÇÃO DA CLASSE TSERVICOAREA	41

Quadro 24 – DEFINIÇÃO DA INTERFACE ISERVICOAREA	41
Quadro 25 – REGISTRO DAS INTERFACES E CLASSES.....	42
Quadro 26 – CODIGO DO OBJETO DE ACESSO A INTERFACE ISERVICOPRODUTO	47
Quadro 27 – EVENTO DE EXCLUSÃO.....	48
Quadro 28 – FUNÇÃO PARA RETORNAR A INTERFACE.....	50
Quadro 29 – EVENTO QUE FAZ ACESSO A APLICAÇÃO PRINCIPAL.....	50

LISTA DE ABREVIATURAS

- B2B - *Business to Business*
- B2C - *Business to Consumer*
- DTD - *Document Type Definition*
- EDI - *Electronic Data Interchange*
- FTP - *File Transfer Protocol*
- HTTP - *Hypertext Transfer Protocol*
- J2EE - *Java 2 Enterprise Edition*
- JSP - *Java Server Pages*
- RFC - *Request for Comment*
- RPC - *Remote Procedure Call*
- SGML - *Standard Generalized Markup Language*
- SMTP - *Simple Mail Transport Protocol*
- SOAP - *Simple Object Access Protocol*
- UML - *Unified Modeling Language*
- URI - *Universal Resource Identifier*
- URL - *Universal Resource Locator*
- URN - *Universal Resource Name*
- XML - *Extensible Markup Language*
- WSDL - *Web Service Description Language*
- W3C - *World Wide Web Consortium*

RESUMO

O presente trabalho descreve 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 que funcionam como exemplo do uso da aplicação principal.

ABSTRACT

The present work describes the implementation of an application prototype that is capable to let a catalog interchange between e-commerce applications, using the Internet. This application provides resources for it in a HTTP server using Web services based on SOAP protocol and XML language. Also is implemented two other e-commerce applications to make an example of interchange application use.

1 INTRODUÇÃO

De acordo com Hahn (1995) no início dos anos 90, foi concretizada a possibilidade da Internet tornar-se disponível à grande parte da população mundial, e com isso, concretizar a criação de uma rede global que conecta milhões de computadores de forma que possam comunicar-se entre si. Com base na disponibilidade desses recursos surgiram diversas aplicações que disponibilizam serviços, sejam eles para o consumidor final ou entre diversas organizações. Essas aplicações são chamadas de *e-commerce* ou comércio eletrônico, sendo diferenciadas principalmente em dois grupos: *Business to Consumer* (B2C) ou *Business to Business* (B2B).

Para Albertin (2000) o *e-commerce* é a realização de toda a cadeia de valor dos processos de negócio num ambiente eletrônico, por meio da aplicação intensa das tecnologias de comunicação e de informação. As tecnologias que tornam possíveis os processos de *e-commerce* não são apenas a Internet e as páginas *Web*, mas sim todo o conjunto de aplicações e recursos que possibilitam que a informação possa estar disponível e ser manipulada através de vários meios eletrônicos. Evidentemente este modelo de aplicações interfuncionais pode se tornar bastante complexo e para que sua implementação torne-se viável é necessário que essas aplicações sejam baseadas em camadas independentes.

Kalakota (2002) mostra que a tendência dos negócios eletrônicos está em direção ao suporte de processos habilitado por *software*, que é realizado pela disponibilização de aplicações de negócios que fundem múltiplas funções em um conjunto de estruturas bem ajustadas. Para participar ativamente do mercado as organizações estão agilizando a integração entre seus processos em níveis altíssimos, conseguindo assim vencer as batalhas de preço, participação no mercado, qualidade e ainda atenderem com mais eficácias as necessidades dos seus clientes.

As aplicações de *e-commerce* só tornam-se possíveis se implementarem grande parte de suas funcionalidades integradas as outras aplicações empresarias. Segundo W3C (2000b), o protocolo *Simple Object Access Protocol* (SOAP) é designado para troca de informações e processamento remoto entre ambientes descentralizados ou distribuídos. É baseado em linguagem *Extensible Markup Language* (XML) e consiste em três partes: um envelope que define a estrutura para descrever o conteúdo da mensagem e como processá-lo, um conjunto

de regras de codificação para especificar instâncias de tipos de dados pertencentes às aplicações, e um mecanismo para processamento remoto.

Segundo Marchal (2000), XML é uma linguagem usada para descrever e manipular dados organizados de forma estruturada e uma de suas principais aplicações é a troca de dados entre organizações. Durante anos, isso foi resolvido com tecnologias de *Electronic Data Interchange* (EDI) – intercâmbio eletrônico de dados, porém todos os esforços para sua padronização não chegaram a um consenso, pois as organizações que o utilizavam precisavam de aplicações específicas e que não são fornecidas com recursos de uma forma padrão.

Como a linguagem XML especifica apenas a estrutura do documento e suas regras de formação, é necessário um protocolo para controlar a transmissão dos dados entre as aplicações. No protocolo SOAP essa transmissão é feita com uma associação com algum protocolo utilizado na Internet, como o protocolo *Hipertext Transfer Protocol* (HTTP). Para compreender melhor essa associação, Snell (2001) relata em sua obra que essa arquitetura provê uma nova maneira de ver e implementar a integração e interoperabilidade entre aplicações fazendo com que a plataforma de desenvolvimento seja irrelevante. Duas aplicações, sem levar em consideração o sistema operacional, linguagem de programação e qualquer outro detalhe de implementação, comunicam-se utilizando mensagens XML sobre protocolos de Internet como HTTP. O protocolo SOAP é a especificação que detalha como a informação deve ser organizada para prover mecanismos de troca de mensagens e processamento remoto.

Neste contexto o presente trabalho visa utilizar a tecnologia SOAP para criação de uma aplicação para comércio eletrônico objetivando a troca de catálogos entre organizações. Essa aplicação é executada nos servidores *Web* das organizações que farão a integração entre os catálogos. Para realização deste trabalho utilizam-se recursos de programação para Internet e aplicações que disponibilizam serviços na forma de *Web Services*. Além da aplicação para troca de catálogos também foram desenvolvidas duas aplicações de comércio eletrônico para ilustrar o funcionamento do protótipo.

1.1 OBJETIVOS DO TRABALHO

O objetivo desta proposta é o desenvolvimento de uma aplicação que controle a troca de catálogos entre organizações via Internet.

Os objetivos específicos do trabalho são:

- a) criar uma aplicação para ser executada em um servidor *Web* que disponibilize recursos para troca de catálogos eletrônicos;
- b) criar uma interface para parametrização dos catálogos a serem trocados;
- c) criar duas aplicações de comércio eletrônico para ilustrar a interoperabilidade da troca de catálogos.

1.2 ESTRUTURA DO TRABALHO

A fundamentação teórica, descrita nos capítulos 2, 3 e 4 fornece visões sobre a construção de aplicações de comércio eletrônico e as tecnologias empregadas para isto. Esses capítulos também destacam as tecnologias XML, SOAP e *Web Services* como recursos bem definidos para construir aplicações integradas em ambientes heterogêneos e distribuídos.

O capítulo 5 trata os tópicos relativos à especificação e implementação do protótipo, iniciando com o estudo dos requisitos principais do problema a ser tratado, seguindo pela especificação e implementação das aplicações que compõe o protótipo, além de apresentar um estudo de caso para ilustrar a funcionalidade do protótipo.

Finalmente a conclusão, dificuldades encontradas e as extensões sugeridas para futuras continuações baseadas neste trabalho estão descritas no capítulo 6.

2 APLICAÇÕES DE COMÉRCIO ELETRÔNICO

Nova economia, novas ferramentas, novas regras. Poucos conceitos revolucionaram tão profundamente os negócios como o *e-commerce*. A simplicidade de implantação, a agilização das interações, os produtos e os pagamentos dos clientes às empresas e das empresas aos fornecedores estão causando verdadeiros terremotos na forma de administrar as corporações.

Kalakota (2002) explica que para competir no mundo do *e-commerce*, uma empresa precisa transformar seus fundamentos. Essa mudança estrutural requer o desenvolvimento de uma estratégia de negócio inovadora, concentrando-se na comercialização e na execução, exigindo ainda mudanças de processo. Ao mesmo tempo, as empresas precisam desenvolver uma infra-estrutura robusta de sistemas de informação para dar suporte ao seu novo conceito.

A integração entre os meios de comércio tradicionais e o *e-commerce* está acontecendo diante dos olhos de todos. A sabedoria convencional diz que o comércio eletrônico é um solvente econômico. Ele dissolve os velhos modelos de negócios, muda a estrutura de custo e formula as conexões entre compradores, vendedores e intermediários. Apesar disto, somente agora está se tornando claro que o comércio eletrônico é também um solvente de relações, dissolvendo fronteiras tradicionais entre clientes e parceiros. Kalakota (2002) separa a evolução do comércio eletrônico em três fases, descritas a seguir.

A primeira fase, de 1994 a 1997, relata a presença das empresas na Internet. Todas as empresas de grande e médio porte queriam assegurar sua marca através de um *site* na *Web*. De fato a maioria não poderia prever o principal motivo de por que faziam isto, mas sabiam que precisariam estar *on-line*.

A segunda fase, de 1997 a 2000 enfatiza as transações eletrônicas, ou seja, comprar e vender utilizando um meio digital. O foco nessa fase estava no fluxo de pedidos e receita. Em alguns casos, era a competição de compradores e vendedores que nunca teriam se encontrado no passado. Em outros, tratava-se simplesmente de realizar transações que teriam sido feitas através de formulário de papel de pedidos de compra e dizer que este negócio tinha sido feito na Internet, embora o significado daquela mudança fosse quase insignificante. Mas, nessa fase, as recomendações eram todas as respeito do fluxo de pedidos a qualquer custo. Como

resultado desses modelos, muitas empresas fecharam ou estão dando seus últimos suspiros, além de estarem afundadas em dívidas.

Hoje, o comércio eletrônico está na terceira fase, que começou em 2000 e é baseada em como a Internet pode influenciar a lucratividade. Lucratividade não significa aumentar somente a receita bruta, mas aumentar as margens de lucro. Essa fase pode ser chamada essencialmente de *e-business*, e inclui todas as aplicações e processos que permitem a uma empresa realizar transações de negócios. Além de englobar o comércio eletrônico, o *e-business* inclui atividades de contato e retaguarda que formam o mecanismo principal do negócio moderno. Assim, o *e-business* não trata apenas das transações de comércio eletrônico ou de compras e vendas pela Internet. É uma estratégia global de redefinição dos antigos modelos de negócios, com o auxílio de tecnologia, para maximizar o valor do cliente e conseqüentemente dos lucros obtidos.

2.1 ESTRUTURA DAS APLICAÇÕES DE E-COMMERCE

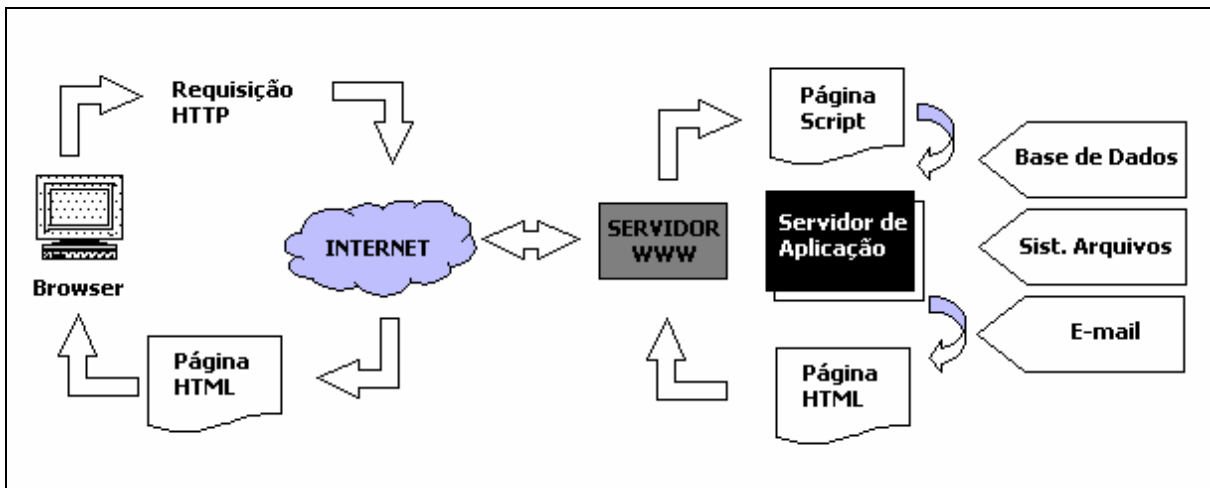
Segundo Brose (2001), a maioria das aplicações para Internet é baseada em softwares baseados no conceito de *Common Gateway Interface* (CGI). Este recurso disponibiliza a execução de programas do lado do servidor e na maioria das vezes é desenvolvido utilizando um padrão diferente para cada aplicação. Os servidores de aplicações para Internet são baseados em sua maioria no protocolo HTTP.

Para Tomaz (2001), basicamente um servidor HTTP permite que sejam acessados apenas conteúdos estáticos na WWW. Para que um *site* tenha funcionalidades não disponíveis apenas com o servidor HTTP, como é o caso de acesso a banco de dados, envio de *e-mail* ou comandos de linguagem de *script* é necessário desenvolver um aplicativo externo ao servidor HTTP. Este aplicativo quando utilizado em conjunto com um servidor HTTP é chamado de servidor de aplicação para Internet. Na figura 1 é demonstrada a seqüência de etapas percorrida para transformação de um *arquivo script* em uma página dinâmica através de um servidor de aplicação. Para escrever um servidor de aplicação para Internet é preciso seguir algumas regras que são definidas pelas diferentes especificações HTTP.

Porém o fato de gerar uma página dinâmica resolve apenas a questão de apresentar ao usuário uma interface para visualização e interação com os dados. O servidor não pode estar

restrito a acessar apenas simples bancos de dados, arquivos ou aplicações de *e-mail*. Nas aplicações para Internet voltadas ao comércio eletrônico é necessária a integração com os processos de negócio disponíveis em vários sistemas, estes talvez implementados em diferentes plataformas.

Figura 1 – ESQUEMA DE TRANSFERÊNCIA DE UMA PÁGINA WWW DINÂMICA



Fonte: Tomaz (2001)

A maioria dos ambientes de desenvolvimento e servidores de aplicação provê recursos para acessar objetos de outras aplicações, mas na maioria das vezes esse acesso é restrito a aplicações desenvolvidas com a mesma tecnologia. Por exemplo, Fields (2000) explica que na plataforma *Java 2 Enterprise Edition* (J2EE) podem ser desenvolvidas aplicações para Internet baseadas em *JSP*, acessando servidores baseados em *Servlets* integrados aos processos de negócio dos sistemas empresariais através de *Enterprise Java Beans*. Como *Servlets* é uma tecnologia disponível apenas para a plataforma *J2EE*, a solução só poderá tornar-se disponível a outros ambientes e aplicações se implementar um mecanismo de acesso padronizado que possa ser implementado em outros ambientes.

2.1.1 APLICAÇÕES B2B

Albertin (2000) define as aplicações B2B como aplicações que servem para fazer transações eletrônicas entre empresas parceiras e fornecedores. A sigla B2B provém do termo em inglês *Business to Business* que significa “negócios para negócios”. Abiteboul (2000) explica que normalmente as aplicações B2B integram-se aos sistemas corporativos das empresas para que possam obter dessas informações sobre os estoques, produção,

necessidades de compra de matéria prima e assim disponibilizar recursos eletrônicos para que os fornecedores dessas empresas possam oferecer seus produtos. Uma grande vantagem deste tipo de recurso é que fornecedores podem estabelecer relações comerciais não importando os limites de horário e distância. Muitos fornecedores que não podiam vender seus produtos utilizando outros meios podem utilizar plataformas B2B para efetuar os seus negócios.

Alguns exemplos práticos de aplicações B2B são empresas de transformação que necessitam comprar materiais diversos e de vários lugares do mundo e que podem centralizar todo seu processo de compra nesta plataforma.

2.1.2 APLICAÇÕES B2C

Kalakota (2002) explica que a relação entre consumidores finais de produtos e serviços e as empresas fornecedoras desses produtos e serviços em meio eletrônico é chamada de B2C. A sigla B2C provém do termo em inglês *Business to Consumer* que significa “negócios para consumidor”.

As aplicações B2C também se integram com os sistemas corporativos, mas com o objetivo de obter informações sobre os produtos a serem vendidos, verifica a disponibilidade e possibilidade de entrega e efetivar as transações eletrônicas de pagamento entre o consumidor final e as empresas fornecedoras.

2.2 DIFICULDADES NO DESENVOLVIMENTO

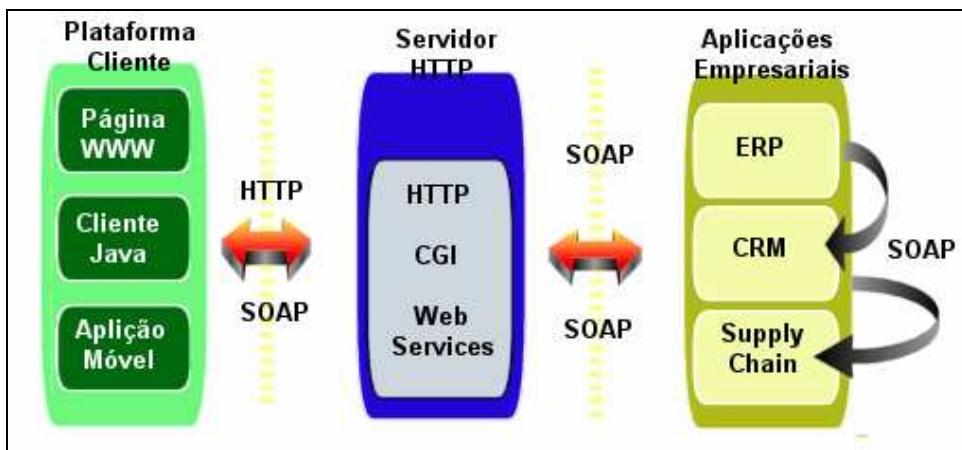
Para disponibilizar uma solução de *e-commerce*, seja ela no modelo B2B ou B2C, os sistemas de informação que subsidiam dados e recursos precisam integrar-se de forma automática e segura. Um dos principais desafios no desenvolvimento de soluções integradas envolvendo sistemas heterogêneos é manter o máximo da funcionalidade de cada sistema, obtendo proveito dos seus dados para alimentar outros sistemas, tendo que desenvolver o mínimo de funcionalidade redundante entre eles. Soluções obtidas neste modelo trazem evoluções tecnológicas e melhores soluções, podendo sempre agregar novas funcionalidades e principalmente aproveitar da melhor forma o que já existe.

Figura 2 – AS APLICAÇÕES NÃO TEM RECURSOS PARA SE INTEGRAREM



Obter este modelo ideal só é possível se todas as aplicações envolvidas implementarem conceitos de comunicação padronizados, pois estes podem ser do conhecimento de todos os fornecedores. As figuras 2 e 3 procuram comparar duas aplicações de comércio eletrônico. A primeira utiliza vários sistemas, porém cada um se integra de uma forma. A segunda demonstra um modelo ideal onde todas as aplicações se comunicam por um protocolo comum.

Figura 3 – UM MODELO IDEAL



O estudo das tecnologias XML, SOAP e *Web services* presente nos capítulos seguintes possibilita implementar recursos para comunicação e distribuição de processamento entre sistemas baseados em qualquer plataforma ou ambiente.

3 EXTENSIBLE MARKUP LANGUAGE (XML)

Light (1999) define a linguagem *Extensible Markup Language* (XML) como um sistema de codificação que o permite a representação de qualquer tipo de informação baseado nos conceitos da linguagem *Standart Generalized Markup Language* (SGML). A especificação completa da linguagem XML pode ser encontrada em (W3C, 2000a).

Segundo Moulitis (2000), o SGML é uma linguagem de marcação genérica amplamente utilizada em diversas áreas de publicação de informação de alto grau de especificação, que foi utilizando extensivamente por muitos anos. Apesar disto, é considerada uma linguagem altamente complexa pois sua especificação tem a extensão de mais de 500 páginas. Esse foi um dos motivos que fizeram com que os desenvolvedores de aplicação não utilizassem amplamente os padrões SGML como um recurso incluído na especificação de seus aplicativos.

O objetivo principal das linguagens de formatação genéricas é fornecer uma descrição detalhada de como o documento deve ser estruturado. A primeira qualidade que distingue os documentos formatados com uma linguagem de marcação genérica é a sua organização visivelmente lógica e sua ordem altamente estruturada. Para Leventhal (1998) a maneira mais comum de separar os elementos em um documento é através de marcas, mais comumente chamadas de *tags*. As *tags* são formadas por um nome envolvido pelos símbolos “<” e “>”, sendo que normalmente são auto-explicativas, pois seu nome significa quase que objetivamente seu significado.

3.1 ELEMENTOS

Conforme explica Skonnard (2002), os elementos fazem parte da maioria do conteúdo de um documento XML. Cada documento XML tem somente um elemento de nível mais alto, conhecido como *elemento documento*. Possuem um nome que os identifica e podem conter descendentes. Estes descendentes podem ser outros elementos, instruções de processamento, comentários ou seções CDATA. Os elementos podem conter ainda atributos e estarem qualificados por um *namespace*.

Elementos são serializados com um par de *tags*: um de abertura e um de fechamento. A sintaxe para o *tag* de abertura é o caractere menor (<), imediatamente seguido pelo nome do

elemento, também conhecido como *tagname*, seguido pelo caractere maior (>). A sintaxe para o *tag* de fechamento é a seguinte seqüência de caracteres (</), imediatamente seguida pelo *tagname*, seguida pelo caractere maior. Os descendentes do elemento são serializados entre os *tags* de abertura e fechamento de seus pais. O exemplo do quadro 1 exemplifica o formato de um elemento com o *tagname* Pessoa. Este elemento possui descendentes com os *tagnames* Nome e Idade.

A linguagem XML não especifica o nome de nenhum elemento, permitindo ao desenvolvedor de um documento XML que escolha que nomes serão usados. Para a criação desses nomes devem ser seguidas algumas regras: devem começar com uma letra ou *underscore* e o caractere inicial deve ser seguido por letras, dígitos, ponto (.), hífen (-) ou *underscores*.

Quadro 1 – ELEMENTO CODIFICADO EM XML

```
<Pessoa>
  <Nome>Luiz Felipe</Nome>
  <Idade>33</Idade>
</Pessoa>
```

3.2 UNIFORM RESOURCE IDENTIFIER

Skonnard (2002) mostra que para acessar um item único na Internet é necessária uma forma de identificação única. *Uniform Resource Identifiers* (URI) provê uma forma de criar identificadores únicos. Descrito em detalhes pelo *Request for Comments* (RFC) 1630, esta especificação provê regras para o uso de muitos protocolos. A sintaxe de uma URI é mostrada no quadro 2. Quando a parte específica contém barras (/), é representada uma estrutura hierárquica, como diretórios.

O tipo mais conhecido de URI é o *Uniform Resource Locator* (URL), utilizado para identificar os documentos e servidores na Internet. Menos comum que a URL, existe um padrão chamado *Universal Resource Name* (URN), que serve como um identificador de recursos persistentes. Sua sintaxe pode ser observada no quadro 2.

Quadro 2 – URI e URN

```
URI: <esquema>:<esquema-especifico>

URN: "urn:" <NID> ":" <NSS>
```

A URN usa a *string* "urn:" para identificar o esquema. NID especifica o identificador do *namespace* e NSS a *string* específica. O uso mais comum do URN é dado em *namespaces* XML.

3.3 NAMESPACES

Como o XML permite aos desenvolvedores que escolham seus próprios *tagnames*, torna-se possível a hipótese que dois desenvolvedores escolham os mesmos *tagnames* para um ou até todos os elementos de um documento XML. Os *namespaces* provêm uma maneira de distinguir deterministicamente dois elementos que tem o mesmo nome, mas de fato, pertencem a lugares e contextos diferentes. Isto é feito pela associação de um elemento com um *namespace*. Um *namespace* atua como um escopo para todos os elementos associados a ele. O nome de um *namespace* é definido por um *Universal Resource Identifier* (URI). O nome de um *namespace* e o nome do elemento associado a ele formam um nome único conhecido como *qualified name*.

Ceponkus (1999) mostra que as declarações do *namespace* aparecem dentro do *tag* de abertura e são usadas para mapear o nome do *namespace* para outra palavra tipicamente menor, conhecida como *namespace prefix*. A sintaxe para declaração de *namespaces* é `xmlns:prefix='URI'`. Também é possível designar um *namespace* sem usar o *prefix*, usando a seguinte declaração: `xmlns='URI'`.

A qualificação de um *namespace* aplica-se ao elemento que o está declarando e a todos os seus descendentes, porém, para que os descendentes sejam considerados elementos qualificados por este *namespace*, algumas regras devem ser seguidas. Os quadros 3 e 4 mostram respectivamente elementos qualificados e não qualificados.

Quadro 3 – ELEMENTOS NÃO QUALIFICADOS

```
<Pessoa xmlns:pre'http://colpani.com/ex01'>
  <Nome>Luiz Felipe</Nome>
  <Idade>33</Idade>
</Pessoa>
```


Quadro 4 – ELEMENTOS QUALIFICADOS

```
<pre:Pessoa xmlns:pre='http://colpani.com/ex01'>
  <pre:Nome>Luiz Felipe</pre:Nome>
  <pre:Idade>33</pre:Idade>
</pre:Pessoa>
```

3.4 ATRIBUTOS

Em Skonnard (2002) os atributos estão descritos como uma forma de prover informações sobre um elemento. Os atributos fazem parte de um elemento e são serializados dentro do *tag* de abertura. Aparecem na forma de pares, no estilo nome/valor, separados por um sinal de igual (=). A regra para definição de seus nomes baseia-se no mesmo conceito dos elementos e seu valor deve aparecer entre aspas simples ou duplas. Um elemento pode conter qualquer número de atributos, desde que todos tenham nomes diferentes.

O quadro 5 mostra exemplos do uso de atributos para representação de dados.

Quadro 5 – ATRIBUTOS

```
<Pessoa Nome='Luiz Felipe' Idade='33' />
<idade mascara='AA' unidade='anos'>20</idade>
<idade mascara='AA' unidade='anos'>36</idade>
```

Os atributos ainda podem conter *namespaces* específicos que devem ser declarados junto com o *namespace* do qual o elemento pertence. O quadro 6 mostra um atributo de nome mascara pertencente ao *namespace* urn:exemplo:Pessoa:Mascara e um atributo chamado unidade pertencente ao *namespace* urn:exemplo:Pessoa:Unidade.

Quadro 6 – NAMESPACES E ATRIBUTOS

```
<Pessoa xmlns='urn:exemplo:Pessoa'
  xmlns:b='urn:exemplo:Pessoa:Mascara'
  xmlns:c='urn:exemplo:Pessoa:Unidade'>
  <Nome>Luiz Felipe</Nome>
  <Idade b:mascara='MM' c:unidade='anos'>33</Idade>
</Pessoa>
```

3.5 SCHEMAS

Para Seely (2002) o *schema* XML provê um método para especificar a estrutura de um elemento XML. Os *schemas* herdam muitas características dos *Document Type Definition*

(DTD), mas são os únicos recursos que permitem especificar informações sobre tipos. As informações sobre os elementos são uma forma de especificar dados sobre os dados, também comumente chamados de metadados. O protocolo *Simple Object Access Protocol* (SOAP) utiliza *schemas* para especificar e obter informações sobre seus elementos.

4 PROTOCOLO SOAP

Segundo Seely (2002), *Simple Object Access Protocol* (SOAP) é um mecanismo para interconexão de aplicações formado por três diferentes partes:

- a) mecanismo de serialização, formado por regras de codificação;
- b) modelo de empacotamento chamado envelope SOAP;
- c) mecanismo *Remote Procedure Call* (RPC);

O protocolo SOAP foi especificado pelo Word Wide Web Consortium (W3C) que desenvolveu essas três partes para serem usadas juntas, porém não impedindo o desenvolvedor de usar cada parte separadamente. Por exemplo, um programa pode utilizar o mecanismo de serialização para gravar um arquivo de configuração ou um sistema de troca de mensagens instantâneas pode utilizar o envelope SOAP para transmitir os dados entre as partes envolvidas. Por outro lado, com o mecanismo RPC pode-se usar envelope SOAP e as regras de codificação para fazer chamadas de função. Essa independência entre as três partes faz com que implementações baseadas em SOAP sejam modulares. Seely (2002) conclui que implementações modulares funcionam bem porque as várias partes desenvolvidas não dependem umas das outras.

Para Scribner (2001), o protocolo SOAP, quando destinado para propósitos de RPC, foi desenhado para traduzir os parâmetros dos métodos a partir de uma forma binária nativa de qualquer linguagem para um conjunto de informações em XML e transportá-los até um servidor remoto. Quando a informação for transportada, um processador SOAP correspondente irá então traduzir as informações XML novamente para o estado binário para processamento.

Este capítulo procura mostrar os principais aspectos envolvidos na especificação e utilização do protocolo SOAP versão 1.2 que apesar de ser uma especificação recente, já é adotado por grande parte dos ambientes de desenvolvimento e servidores de aplicação disponíveis no mercado.

A especificação do protocolo SOAP versão 1.1 pode ser encontrada em W3C (2000b) e mostra como utilizá-lo em conjunto com o protocolo HTTP, ou seja, padroniza a sua implementação sobre um mecanismo de transporte.

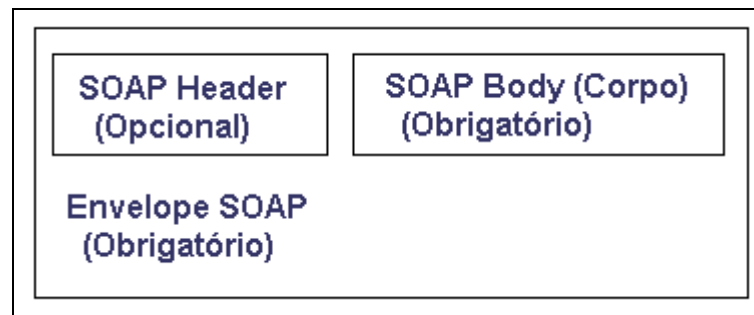
4.1 CONVENÇÕES

Os prefixos de *namespace* “env”, “enc”, “xs” e “xsi” que serão utilizados nos exemplos deste capítulo são associados aos *namespaces* SOAP correspondentes a “<http://www.w3.org/2001/12/soap-envelope>”, “<http://www.w3.org/2001/12/soap-encoding>”, “<http://www.w3.org/2001/XMLSchema>” e “<http://www.w3.org/2001/XMLSchema-instance>” respectivamente.

4.2 ESTRUTURA DA MENSAGEM SOAP

Como explica Seely (2002), todas as mensagens SOAP são documentos XML utilizados para transmitir informações entre dois pontos, definidos normalmente por um cliente e um servidor. Essas mensagens possuem obrigatoriamente um envelope SOAP e um corpo (*body*). Ainda podem conter opcionalmente um cabeçalho chamado SOAP *header*. A figura 4 mostra esse conjunto em um diagrama de bloco. Cada um desses elementos possui um conjunto especial de regras que serão discutidas neste capítulo.

Figura 4 – DIAGRAMA DE BLOCO DA MENSAGEM SOAP



Fonte: Seely (2002)

Scribner (2001) conceitua cada um desses três elementos como um objeto visto que cada um deles tem um propósito distinto. O conjunto desses três objetos é então chamado de “Modelo de objeto SOAP”.

O envelope SOAP forma o elemento raiz do documento XML da mensagem SOAP. Dessa maneira, uma de suas principais tarefas é transportar as informações do *namespace* XML usadas quando a mensagem foi serializada. Como o envelope é a raiz do documento, os outros dois objetos SOAP, também em XML, devem ser serializados dentro do envelope.

O cabeçalho SOAP é uma parte opcional da mensagem que carrega as informações necessárias para processar a solicitação e que não estão contidas nas propriedades do método. Normalmente o cabeçalho SOAP carrega informações sobre chaves criptográficas, identificadores de seqüências transacionais ou outras informações que o processador SOAP irá requerer para gerenciar e processar a chamada remota.

No corpo SOAP são encontradas todas as informações do método e seus parâmetros armazenados em XML. O processador SOAP lê o corpo e converte as informações sobre os parâmetros que estão no formato XML para o formato nativo com o objetivo de processá-las. O quadro 7 exemplifica uma mensagem SOAP utilizada em um sistema imaginário de reserva de viagens.

Quadro 7 – ENVELOPE SOAP

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
  <env:Header>
    <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
      env:actor="http://www.w3.org/2001/12/soap-envelope/actor/next"
      env:mustUnderstand="true">
      <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</reference>
      <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
    </m:reservation>
    <n:passenger xmlns:n="http://mycompany.example.com/employees"
      env:actor="http://www.w3.org/2001/12/soap-envelope/actor/next"
      env:mustUnderstand="true">
      <n:name>John Q. Public</n:name>
    </n:passenger>
  </env:Header>
  <env:Body>
    <p:itinerary xmlns:p="http://travelcompany.example.org/reservation/travel">
      <p:departure>
        <p:departing>New York</p:departing>
        <p:arriving>Los Angeles</p:arriving>
        <p:departureDate>2001-12-14</p:departureDate>
        <p:departureTime>late afternoon</p:departureTime>
        <p:seatPreference>aisle</p:seatPreference>
      </p:departure>
      <p:return>
        <p:departing>Los Angeles</p:departing>
        <p:arriving>New York</p:arriving>
        <p:departureDate>2001-12-20</p:departureDate>
        <p:departureTime>mid morning</p:departureTime>
        <p:seatPreference/>
      </p:return>
    </p:itinerary>
    <q:lodging xmlns:q="http://travelcompany.example.org/reservation/hotels">
      <q:preference>none</q:preference>
    </q:lodging>
  </env:Body>
</env:Envelope>
```

Fonte: W3C (2000b)

4.2.1 ENVELOPE

O envelope serve literalmente como um invólucro para as informações que são realmente importantes. Pode ser imaginado como um envelope postal tradicional que carrega em seu interior uma carta. Porém a analogia deve parar neste ponto, pois ao contrário do envelope postal tradicional o envelope SOAP não especifica um destinatário. O mecanismo de transporte do envelope SOAP pode ser implementado com vários protocolos, entre eles o HTTP. Essa relação será discutida com detalhes no tópico 4.4.

A formação do envelope deve obrigatoriamente seguir as seguintes regras:

- a) deve ser obrigatoriamente definido por um elemento de nome `Envelope`;
- b) toda mensagem SOAP deve contê-lo como o primeiro elemento XML;
- c) este elemento pode conter declarações no *namespace*, atributos adicionais e sub-elementos, porém todos devem ser qualificados pelo *namespace*.

O quadro 8 mostra o documento XML de uma mensagem SOAP apenas com a notação do elemento `Envelope`.

Quadro 8 – ELEMENTO ENVELOPE

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope">
</SOAP-ENV:Envelope>
```

4.2.2 CABEÇALHO

Uma mensagem SOAP pode ou não incluir um cabeçalho (*header*). Para Seely (2002) esta parte opcional da mensagem provê um mecanismo para prover informações adicionais. Quando incluído deve ser o primeiro elemento filho do elemento `Envelope`. Normalmente, as informações contidas no elemento `Header` são usadas para autenticar informações da requisição ou para prover mecanismos para gerenciamento de transações.

Scribner (2001) nos mostra que o *header* SOAP possui atributos para indicar se certo comportamento é desejado ou requerido. O atributo `mustUnderstand` pode conter o valor 0, significando que o elemento pode ser ignorado caso não faça sentido, ou 1, significando que deverá executar uma ação caso o elemento não faça sentido. Se os elementos filhos do `Header` não possuírem este elemento as aplicações que processarem a mensagem irão entender que o

seu valor é 0. Este mecanismo permite a servidores que não sabem o que fazer com o *header* que o ignorem e ainda provejam uma resposta da requisição.

Outro atributo que pode estar contido no *header* é o *actor*. Em Scribner (2001) o atributo *actor* é explicado como o recurso para identificar a passagem entre um ponto e outro no trânsito da mensagem até seu destino. Uma mensagem SOAP pode precisar passar por várias aplicações intermediárias até chegar ao seu destino final. Uma aplicação intermediária é nada mais que uma aplicação SOAP adicional ao longo do caminho da mensagem. A cada parada, a aplicação que recebe a mensagem faz uma leitura no *header* para ver que partes são destinadas a ela, bem como para ver quem será a próxima aplicação a receber a mensagem. Tanto as aplicações intermediárias como a final são definidas por uma URI. Seely (2002) explica que a passagem do *header* entre um *actor* e outro deve seguir algumas regras.

Primeiramente, enquanto a mensagem é passada entre as aplicações intermediárias, cada uma delas precisa remover suas informações de processamento antes de encaminhar a mensagem ao próximo ponto. O receptor da mensagem pode dizer qual elemento do *header* necessita para seu processamento consultando o valor do atributo *actor*, que para esta característica deve conter a URI especial <http://schemas.xmlsoap.org/soap/actor/next>. Este mecanismo pode ser usado para processar uma mensagem que necessite da interação de várias aplicações.

4.2.3 CORPO

O principal uso de um protocolo de rede é provavelmente enviar informações de um endereço na rede até outro. A principal tarefa do elemento corpo (*body*) é carregar os dados para o protocolo SOAP. No *body* estão contidos os dados que de fato estão sendo enviados ao receptor (Seely, 2002).

Esta parte foi designada inicialmente para organizar os mecanismos de RPC e retornar informações de erro, conhecidas como *SOAP Faults*. Porém, se o protocolo SOAP for usado para propósitos de transmissão de dados, este elemento poderá ser usado para transmitir qualquer informação que possa ser serializada segundo o conjunto de regras definidos na especificação do protocolo.

4.2.4 ERROS SOAP

Dos três elementos principais do envelope, apenas o *body* tem um elemento filho predefinido, o `Fault`. Uma aplicação SOAP pode utilizar este elemento para carregar informações sobre erro na mensagem, por isto é utilizado apenas em mensagens de resposta.

O elemento `Fault` define quatro elementos filhos:

- **faultcode:** utilizado para prover uma maneira de identificar o que aconteceu. Este elemento é obrigatório em todos os elementos `Fault`;
- **faultstring:** este elemento expressa o erro de uma maneira textual e também é obrigatório em todos os elementos `Fault`;
- **faultactor:** identifica qual das aplicações ao longo do processamento da mensagem escreveu o elemento `Fault`;
- **detail:** este elemento leva os erros específicos relatados sobre o conteúdo do elemento `body`.

Para Seely (2002), mensagens SOAP podem falhar por uma série de motivos. A especificação tenta qualificar as falhas mais comuns de modo que todas as implementações baseadas em SOAP possam entender porque uma chamada não obteve sucesso. Pode-se usar como exemplo um dos erros mais comuns em sistemas distribuídos que é a falta de permissão para que uma chamada acesse um método, objeto ou base de dados. Esses erros podem acontecer em qualquer plataforma, seja ela Windows ou Unix. Tipicamente, esses erros são qualificados por códigos numéricos, com 101 ou 99999. Ao invés de usar números inteiros para definir os erros, o protocolo SOAP utiliza nomes XML qualificados. Estes usam o caracter ponto (“.”) para separar o erro mais genérico do mais específico. Então, se houver um problema de autenticação do cliente será retornado um código com conteúdo `Client.Authentication`.

A especificação define quatro *faultcodes* genéricos:

- **versionMismatch:** se a aplicação encontrou um *namespace* inválido para o envelope;
- **mustUnderstand:** a aplicação não entendeu o elemento *header* que tem o atributo `mustUnderstand` setado para 1;

- **client:** ocorre em mensagens incorretamente formadas, como erros de autenticação e dados perdidos em transações. Quando o cliente recebe este tipo de informação de erro como retorno, somente solicitará retransmissão depois que o conteúdo for modificado;
- **server:** são erros relativos ao processamento da mensagem, e não à mensagem propriamente dita. Erros deste tipo são comuns quando o servidor tem problemas de sobrecarga, falta de espaço em disco ou qualquer outro problema que o impeça de processar a mensagem naquele momento. Clientes que recebem este tipo de erro como retorno provavelmente obterão sucesso se solicitarem o processamento depois de algum tempo.

4.3 SERIALIZAÇÃO

Para transmitir informações sobre os parâmetros de um método que está armazenado na memória local e codificado de maneira nativa relativa ao ambiente em que a aplicação foi gerada, são necessárias regras de codificação que descrevam o processo de serialização, ou seja, que definam regras para representar os valores desses parâmetros.

Segundo Scribner (2001), como o protocolo SOAP pode ser usado para propósitos de transmissão de dados, é possível representar as informações em quase qualquer tipo de arranjo, desde que codificadas em XML. Porém quando o protocolo SOAP é utilizado para mecanismos de RPC, sua especificação fornece um mecanismo preciso para serializar informações destinadas ao sistema remoto.

4.3.1 TIPOS SIMPLES

Como o SOAP é baseado em XML, todas as características relacionadas à representação de tipos simples são herdadas da especificação *XML Schema Part 2*, que pode ser encontrada em Biron (2001). Essa especificação descreve como representar tipos de dados simples utilizando *schemas* e é utilizada pelo protocolo SOAP, que se refere a este grupo de definições como “tipos simples”.

Como regra, valores comuns são serializados como elementos XML simples. Os *tagnames* são compatíveis com o nome do parâmetro especificado no método. O quadro 9 mostra um método chamado Soma, que é serializado conforme o quadro 10.

Quadro 9 – MÉTODO SOMA

```
function Soma(A, B: Integer): Integer;
```

Se os parâmetros A e B receberem respectivamente os valores 10 e 12, o pacote SOAP correspondente será muito parecido com o quadro 10.

Quadro 10 – MÉTODO SOMA SERIALIZADO EM SOAP

```
<soap:body>
  <m:Soma xmlns:m="http://tempuri.org">
    <A>10</A>
    <B>12</B>
  </m:Soma>
</soap:body>
```

Scribner (2001) explica que A e B são elementos embutidos (descendentes do método) serializados na ordem especificada no método, da esquerda para a direita. Neste exemplo os parâmetros são valores inteiros. É possível representar da mesma maneira tipos de dados simples como *date*, *datetime*, *float* e *string*.

Quando os parâmetros são passados por referência, ou seja, podem servir como recurso para entrada e saída, a serialização do método descrito no quadro 11 deve ser feita conforme o exemplo no quadro 12.

Quadro 11 – MÉTODO VERIFICAPRESSAO

```
function VerificaPressao(var Press: Integer): Boolean;
```

Este método (imaginário) dispara um código para verificar a pressão em algum vasilhame. Se a pressão exceder o valor passado no parâmetro `Press`, o método deve retornar um valor verdadeiro. Em outro caso, se a pressão estiver menor, o método deve retornar no parâmetro `Press` o valor real da pressão. A adição do atributo `ref` informa que o método pode consumir o valor do parâmetro e modificá-lo para saída.

Depois de feita a requisição, uma possível resposta será formada conforme descrito no quadro 13, considerando que o vasilhame tem uma pressão igual a 80.

Quadro 12 – MÉTODO VERIFICAPRESSAO SERIALIZADO EM SOAP

```

{ Chamada para o método VerificaPressao }
var
  Valor: Integer;
begin
  Valor := 100;
  If VerificaPressao(Valor) then
    ...
End;

{ O pacote SOAP dessa requisição }

<soap:body>
  <m:VerificaPressao xmlns:m="http://tempuri.org">
    <Press href="#p0"/>
  </m:VerificaPressao>
  <m:Valor xmlns:m="http://tempuri.org" id="p0">
    100
  </m:Valor>
</soap:body>

```

Quadro 13 – RESPOSTA PARA O MÉTODO VERIFICA PRESSAO

```

<soap:body>
  <m:VerificaPressao xmlns:m="http://tempuri.org">
    <return>false</return>
    <Press href="#p0">
  </m:VerificaPressao>
  <m:Valor xmlns:m="http://tempuri.org" id="p0">
    80
  </m:Valor>
</soap:body>

```

4.3.2 ENUMERAÇÕES

Seely (2002) explica que as enumerações tem sido usadas a longo tempo em programação e mostra que a maneira que o protocolo SOAP faz sua serialização é bastante simples e intuitiva. Enumerações são representadas na memória em tempo de execução na forma de valores inteiros. Mesmo utilizando representações textuais, conforme o quadro 14, a

representação na memória será sempre feita por números. Sendo assim, os valores de `cAzul`, `cBranco` e `cPreto` serão representados em memória respectivamente como 0, 1, 2.

Quadro 14 – ENUMERAÇÃO

```
TCor = (cAzul, cBranco, cPreto)

procedure Corfavorita(Cor: TCor);
```

Para representar enumerações, o protocolo SOAP simplesmente volta a usar a representação textual. O pacote resultante da requisição gerada pelo método `Corfavorita`, recebendo como parâmetro o valor `cAzul`, está representado no quadro 15.

Quadro 15 – BODY DA MENSAGEM SOAP GERADO

```
<soap:Body>
  <m:Corfavorita xmlns:m="http://tempuri.org">
    <Cor>cAzul</Cor>
  </m:Corfavorita>
</soap:Body>
```

4.3.3 ARRAY DE BYTES

A habilidade de representar um *array* de bytes permite ao SOAP transportar dados que não podem ser representados como uma simples *string*. Utilizando esse recurso pode ser transmitidos sons, imagens, vídeos e qualquer tipo de dado binário (Seely 2002).

As regras do protocolo SOAP definem um mecanismo simples que consiste em converter os bytes para o formato Base64 e gravar esses dados convertidos no elemento XML referente ao parâmetro. Como o XML define alguns elementos como sendo reservados para seu uso, é necessário fazer uma conversão caso a codificação em Base64 contenha esses caracteres. Esta conversão consiste em transformar os caracteres reservados em outros. O quadro 16 ilustra essa relação de conversão.

Quadro 16 – CARACTERES RESERVADOS QUE DEVEM SER CONVERTIDOS

Caractere Especial	Novo Valor
<	<
>	>
'	'
"	"
&	&

Segundo Scribner (2001), umas das vantagens disto é que o resultado textual do que antes era binário garante a transmissão pela Internet visto que os *firewalls* vêm esses dados de forma textual.

A codificação Base64 é tipicamente utilizada para codificar anexos em *e-mails* pois os servidores SMTP são particularmente sensíveis a certos valores textuais e caracteres. Essa codificação troca cada byte por uma representação textual “garantida”.

Aplicando a codificação Base64 ao texto “Hello World”, será obtido o conteúdo conforme o quadro 17.

Quadro 17 – CODIFICAÇÃO BASE64

SGVsbG8gV29YBGQ=

O quadro 18 exemplifica o uso de um método chamado `EnviaBytes`, que recebe parâmetro um *buffer*.

Quadro 18 – MÉTODO PARA TRANSMITIR DADOS BINÁRIOS

```

<soap:Body>
  <m:EnviaBytes xmlns:m="http://tempuri.org">
    <bytes href="#p0"/>
  </m:EnviaBytes>
  <m:bytes xsi:type="xsd:base64binary" id="p0">
    SGVsbG8gV29YBGQ=
  </m:bytes>
</soap:Body>
```

Conforme descrito nos tópicos anteriores, tipos de dados simples são relativamente fáceis de se codificar. A serialização de tipos de dados compostos possui algumas características especiais que serão discutidas a seguir.

4.3.4 TIPOS COMPOSTOS DE DADOS

A serialização de tipos compostos pode ser um pouco complicada pois deve prever a serialização de qualquer tipo composto, seja ele matriz (*array*) ou estrutura (*struct*). Estes tipos de dados podem ser representados de uma diversidade infinita de formas, e as regras de serialização devem prever todas elas.

Scribner (2001) explica que no protocolo SOAP as estruturas são tipos de dados compostos constituídas de elementos acessados por nome. Matrizes, por outro lado, são acessadas pela posição ordinal. O resultado disto é que na serialização de matrizes são usados *tagnames* genéricos, e nas estruturas, o nome fornecido pela formação original deve ser mantido.

4.3.4.1 ESTRUTURAS

Na serialização de uma estrutura o nome dos atributos é usado para distinguir entre os seus valores. O quadro 19 exemplifica a serialização da estrutura `TAluno`, utilizada no método `VerificaMatrícula`.

Quadro 19 – ESTRUTURA TALUNO

```
TAluno = Record
  Nome      : String;
  Codigo    : Integer;
  Endereço  : String;
  Telefone  : String;
end;

<soap:Body>
  <m:VerificaMatricula xmlns:m="http://tempuri.org">
    <Aluno>
      <Nome>Cristiano Fornari Colpani</Nome>
      <Codigo>18261</Codigo>
      <Endereco>Avenida Tom Jobim, Copacabana</Endereco>
      <Telefone>47 30373209</Telefone>
    </Aluno>
  </m:VerificaMatricula>
</soap:Body>
```

4.3.4.2 MATRIZES

Quando matrizes são serializadas, é importante lembrar que elas podem ter diferentes formas e tamanhos. Os principais tipos de matrizes são: simples, parciais e esparsas. Para obter um exemplo considere o quadro 20. O atributo `arrayType` especifica que existem 3 elementos na matriz.

Quadro 20 – MATRIZ SIMPLES

```
Num: array[1..3] of Integer;

Num[1] := 1;
Num[2] := 5;
Num[3] := 99;

<m:Num soapenc:arrayType="xsd:int[3]" xmlns:m="http://tempuri.org">
  <int>1</int>
```

```
<int>5</int>
<int>99</int>
</m:Num>
```

O quadro 21 exemplifica uma matriz parcial, ou seja, uma matriz simples que não teve todos os seus elementos preenchidos. O atributo `offset` especifica até onde a matriz foi preenchida.

Quadro 21 – MATRIZ PARCIAL

```
Num[1] := 10;
Num[2] := 20;

<m:Num soapenc:arrayType="xsd:int[3]" xmlns:m="http://tempuri.org"
  soapenc:offset="[2]">
  <int>10</int>
  <int>20</int>
</m:Num>
```

O quadro 22 exemplifica uma matriz esparsa, ou seja, apenas algumas posições foram preenchidas.

Quadro 22 – MATRIZ ESPARSA

```
Num[100] := 10;
Num[333] := 20;

<m:Num soapenc:arrayType="xsd:int[350]" xmlns:m="http://tempuri.org">
  <int soap:position="[100]">10</int>
  <int soap:position="[333]">20</int>
</m:Num>
```

4.3.5 REMOTE PROCEDURE CALL

Segundo W3C (2000b) um dos objetivos do protocolo SOAP é encapsular funcionalidades de *Remote Procedure Call* (RPC) utilizando as funcionalidades do XML. Esse encapsulamento é baseado nos conceitos de serialização discutidos anteriormente. De uma forma geral, para utilizar um mecanismo de RPC são necessárias algumas informações:

- nome do procedimento ou método que será chamado;
- parâmetros e seus valores.

Porém o protocolo SOAP não possui nenhuma linguagem para definição dessa interface. O mecanismo de RPC é definido por um tipo de mensagem e conta com o protocolo a que está amarrado para mapear a mensagem até o destino. No caso do HTTP, a URI requisitada diz à aplicação SOAP o objeto que deve ser utilizando na execução da chamada.

4.4 WEB SERVICES

O uso de *web services* na Internet está crescendo rapidamente por possibilitar a interoperabilidade e comunicação entre aplicações. Estes serviços provêm um modelo para implementar a comunicação entre aplicações de diferentes tipos e plataformas. À medida que se promove interoperabilidade entre as aplicações elas se tornam mais complexas e para que continuem bem estruturadas precisam ser baseadas em uma arquitetura. A especificação da arquitetura de *web services* pode ser encontrada em (W3C, 2002).

Segundo W3C (2002), o *web service* é um software identificado por uma URI, do qual as interfaces podem ser descritas e descobertas por instrumentos baseados em XML e ainda suportar interações diretas com outros softwares utilizando mensagens em XML transportadas sobre protocolos de Internet.

Scribner (2001) mostra em sua obra que a maioria das implementações de *web services* são construídas sobre o protocolo *Hypertext Transfer Protocol* (HTTP), utilizando o protocolo SOAP como padrão para as mensagens e uma forma para descrição das interfaces chamada *Web Services Description Language* (WSDL).

4.4.1 WSDL

Scribner (2001) define WSDL como a linguagem utilizada para descrever como um software deve interagir como um *web service* em particular. Aplicações cliente utilizam os documentos WSDL para entender a estrutura lógica e a sintaxe do *web service*.

Todo documento WSDL define um serviço como uma coleção de portas. Essas portas são URLs e definem a localização do serviço. Para que esses serviços possam responder as solicitações estas precisam estar de acordo com o que o *web service* está esperando.

4.4.2 UTILIZAÇÃO DE WEB SERVICES

Visivelmente uma das áreas mais indicadas para a implementação de *web services* é a integração de processos de negócio em aplicações B2B. Por mais de duas décadas os desenvolvedores vêm tentando integrar processos de negócio utilizando uma combinação de software e protocolos de comunicação. O *Electronic Data Interchange* (EDI) tem sido a tecnologia escolhida por muitos anos. O seu principal problema é o alto custo, pois esbarra

principalmente na estrutura necessária e no tempo de desenvolvimento nas aplicações para que fiquem integradas aos processos de negócio, tornando-se assim uma solução cara, possível apenas para grandes empresas.

Quando o XML se tornou popular, claramente foi visto como um mecanismo barato e simples para implementar soluções B2B. Segundo Leventhal (2002), um grupo foi formado e designado para desenvolver uma tecnologia baseada em XML, que poderá tornar-se o padrão para comunicação em processos de negócio. Este esforço é conhecido como *Electronic Business XML* (ebXML) e está sendo patrocinado pelo *Organization for the Advancement of Structured Information Standards* (OASIS) e pelo *United Nation's Center for Trade Facilitation and Electronic Business* (UN/CEFACT). O padrão ebXML escolheu o protocolo SOAP como padrão para empacotamento de suas mensagens.

Segundo Seely(2002), outra funcionalidade importante designada aos *web services* é a integração de sistemas heterogêneos, pois a maioria das empresas possui sistemas computacionais distribuídos em diversas plataformas. A integração destes sistemas pode trazer redução nos custos e evolução tecnológica de todo o sistema.

Os protocolos de RPC mais comuns nunca conseguiram prover uma solução satisfatória para esse problema. Enquanto era possível fazer com que sistemas muito diferentes se comunicassem, o tempo envolvido para fazer isto normalmente não compensava os benefícios. Uma das maiores dificuldades em fazer sistemas diferentes se comunicarem é que os fornecedores escolhem padrões que podem especificamente trazer melhoras na performance de suas próprias plataformas. A interpretação de padrões é outra causa da incompatibilidade entre os protocolos, pois normalmente as especificações são complexas e não podem ser entendidas sem a presença do fornecedor.

5 DESENVOLVIMENTO DO PROTÓTIPO

O presente trabalho resultou na criação de um protótipo de sistema para troca de catálogos baseado na implementação de um *web service*. Como um *web service* é apenas um recurso para acessar e publicar métodos de uma aplicação, tornou-se necessário o desenvolvimento de duas aplicações de comércio eletrônico que fazem acesso ao sistema de troca de catálogos e formam assim o conceito de uma aplicação B2B.

A seguir apresentar-se-ão detalhes sobre a especificação e a implementação do protótipo de software baseado em um *web service* e das aplicações consumidoras dos recursos do protótipo.

5.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O objetivo do desenvolvimento deste trabalho é criar o protótipo de software para troca de catálogos. Este protótipo é baseado na implementação de um *web service* e por isso disponibiliza suas funcionalidades através de métodos remotos que são acessados por aplicações que implementam o acesso a essas chamadas através do protocolo SOAP.

A aplicação para troca de catálogos fica disponível em um servidor HTTP e sendo assim suas funcionalidades podem ser acessadas por todas as aplicações que tenham acesso a este servidor. Se este servidor HTTP estiver disponível na Internet então a aplicação poderá ser acessada por qualquer outra aplicação que tenha acesso a Internet e implemente o protocolo SOAP como cliente.

Para possibilitar a interconexão entre todas essas partes é necessário que as mesmas utilizem um padrão como o SOAP para se comunicar. O conjunto da implementação do software para troca de dados com as aplicações de comércio eletrônico implementadas como exemplo de funcionalidade, mostra como é possível fazer a integração entre esses sistemas e que essa integração é possível mesmo que as plataformas de desenvolvimento ou ambiente sejam diferentes, pois cada aplicação é executada em uma camada independente.

Os tópicos seguintes descrevem a especificação e implementação da aplicação principal e das aplicações exemplo, sendo estas implementadas em diferentes plataformas.

5.2 ESPECIFICAÇÃO

Neste tópico serão apresentadas as especificações do protótipo de software para troca de catálogos e das aplicações de comércio eletrônico.

O protótipo desenvolvido neste trabalho visa atender a necessidade de duas ou mais empresas que necessitam trocar catálogos de produtos de forma eletrônica para efetivar uma possível compra. O protótipo então funciona como um repositório de regras de negócio que permite que seus recursos sejam acessados por outras aplicações. Essas aplicações são especificadas nos casos de uso e serão posteriormente implementadas em plataformas de desenvolvimento diferentes para ilustrar a funcionalidade e a independência da comunicação de aplicações baseadas em SOAP.

Deste modo, como a aplicação principal apenas mantém regras de negócio e permite que seus recursos sejam acessados por outras telas, todos os cadastramentos necessários serão feitos pelas aplicações especificadas nos casos de uso. A interação com a aplicação de troca de catálogos é efetivada entre os sistemas que se comunicam com ela e suas regras de negócio, sendo que o usuário final apenas interage com as aplicações de comércio eletrônico.

Para ilustrar então a funcionalidade da implementação do protótipo são necessários três casos de uso distintos. “Parametriza sistema” é definido pela carga de dados principal no protótipo, que consiste em cadastrar as empresas fornecedoras e receptoras de catálogos. “Solicita e recebe catálogos” consiste em cadastrar os produtos que a empresa deseja receber e criar a ordem de solicitação, e finalmente caso de uso “envia de catálogo”, que é o ato de enviar os catálogos para atender a necessidade de uma empresa que está solicitando os catálogos.

A especificação deste protótipo foi baseada na técnica de orientação a objetos *Unified Modeling Language* (UML) com base em (Larman, 2000), utilizando-se a ferramenta *Poseidon*. Esta ferramenta foi escolhida por possuir uma versão *freeware* e por dar suporte a todos os diagramas da UML utilizados nesta especificação. Os diagramas utilizados foram: diagrama de classes, diagrama de casos de uso ou “*use cases*” e diagrama de seqüência. Mais informações sobre esta ferramenta podem ser encontradas na documentação oficial do *Poseidon* disponível em (BOGER 2001).

5.2.1 APLICAÇÃO PARA TROCA DE CATÁLOGOS

A aplicação principal, que centraliza todos os conceitos lógicos relacionados ao processo de troca de catálogos, deve ficar em uma camada independente das camadas de interface com o usuário e sua especificação contempla apenas a construção de um conjunto de regras de negócio que possibilitam a execução de suas funcionalidades. Posteriormente no capítulo relativo à implementação serão descritos e discutidos os recursos utilizados para possibilitar o acesso a todas essas funcionalidades.

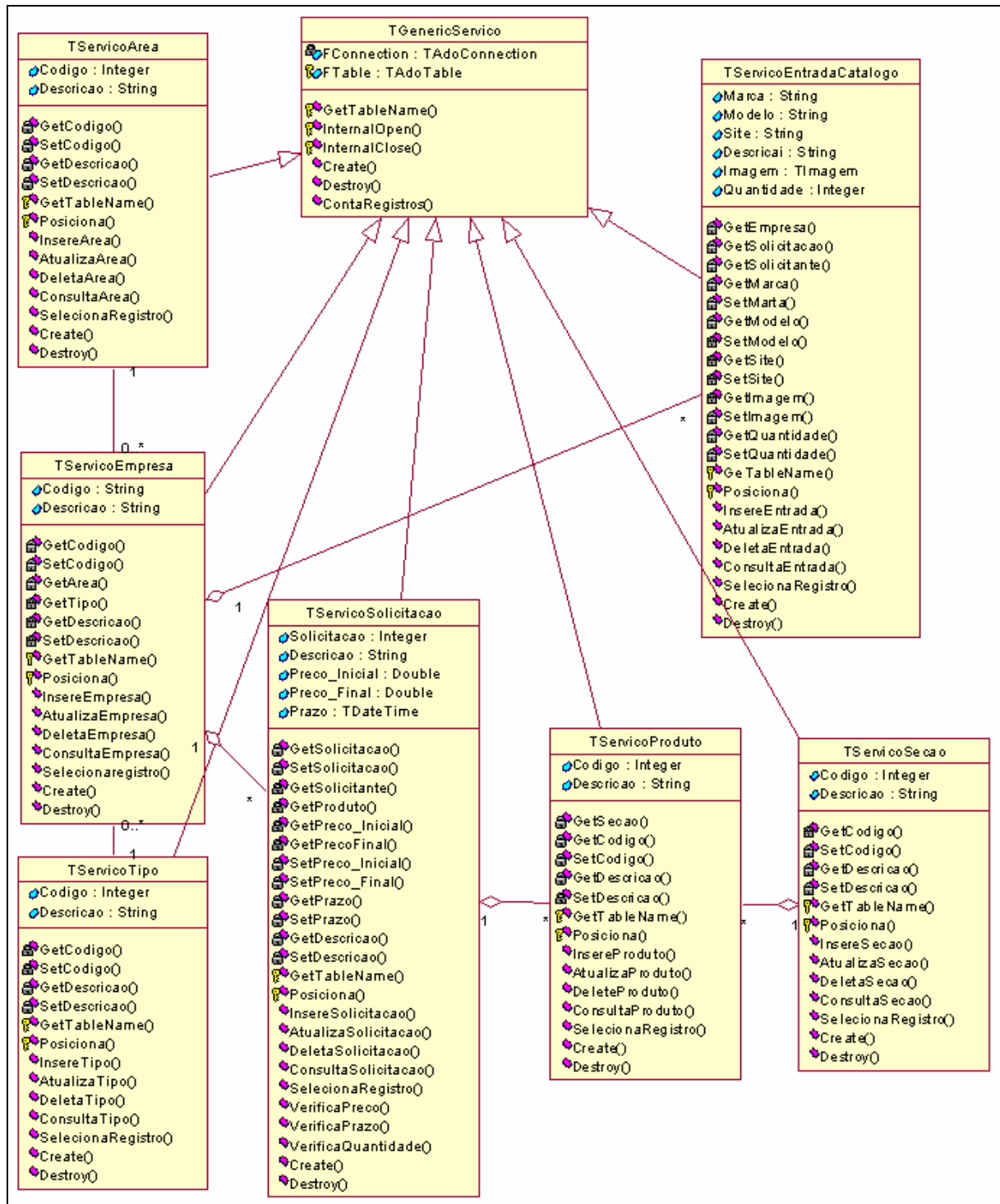
Para especificar as regras de negócio que a aplicação mantém foi utilizado o diagrama de classes descrito na figura 5.

As classes especificadas neste diagrama usam um prefixo *TServico* pois na fase de implementação são utilizadas como *web services*. Todas as classes do protótipo são derivadas da classe *TGenericServico* pois esta define as principais funcionalidades para efetuar a abertura física das tabelas relacionadas às classes além de definir um modelo de construção a ser seguido por todas as outras.

A seguir são descritas as funcionalidades de cada classe:

- a) *TGenericServico*: classe principal do protótipo que define um modelo para construção das classes descendentes e define a funcionalidade para abertura física das tabelas bem como a conexão com o banco de dados;
- b) *TServicoArea*: define a área das empresas que irão trocar catálogos, como alimentícia, construção civil, eletrodomésticos;
- c) *TServicoTipo*: diferencia as empresas do cadastro de empresas entre compradoras receptoras de catálogo) e fornecedores;
- d) *TServicoEmpresa*: mantém o cadastro das empresas envolvidas nas trocas;
- e) *TServicoSecao*: define a seção em que se enquadra algum produto;
- f) *TServicoProduto*: define o produto que uma empresa deseja comprar, como geladeira, fogão, vídeo-game, monitor;
- g) *TServicoSolicitacao*: é a ordem que uma empresa gera para receber catálogos;
- h) *TServicoEntradaCatalogo*: recipiente onde são mantidos os catálogos enviados pelos fornecedores.

Figura 5 – DIAGRAMA DE CLASSES DA APLICAÇÃO



5.2.2 CASOS DE USO

A interação entre os sistemas desenvolvidos para exemplificar a funcionalidade da aplicação principal pode ser melhor compreendida em três diagramas de casos de uso (*use case*) e seus respectivos diagramas de seqüência.

5.2.2.1 PARAMETRIZAÇÃO DO SISTEMA

Este diagrama descreve a parametrização e carga de dados para algumas classes do protótipo. Essa parametrização é sempre necessária quando a aplicação começa a operar e não possui nenhum dado ou ainda quando são incluídas mais empresas e fornecedores no processo de troca.

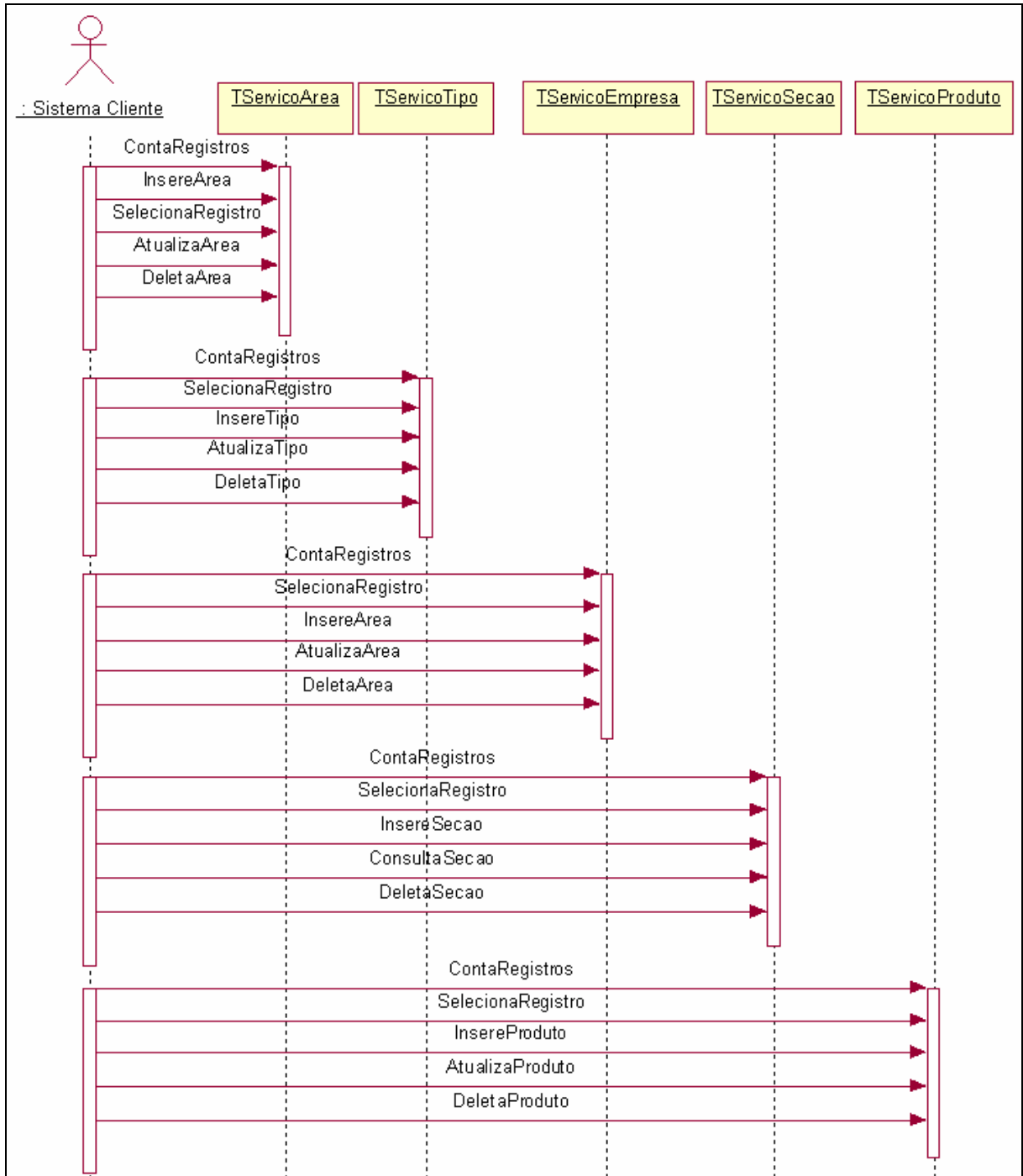
A figura 6 apresenta o diagrama de caso de uso onde um sistema cliente interage com os recursos necessários para parametrizar as classes do protótipo de troca de dados.

Figura 6 – CASO DE USO “PARAMETRIZAÇÃO DO SISTEMA”



O processo pode ser melhor compreendido observando o diagrama de seqüência da figura 7. Sempre que um sistema necessite parametrizar os dados principais do sistema o esquema observado no diagrama de seqüência começa com o sistema cliente requisitando a criação dos objetos *TServicoArea*, *TServicoTipo*, *TServicoEmpesa*, *TServicoSecao* e *TServicoProduto*. O sistema cliente envia várias mensagens a esses objetos com o intuito de visualizar a manipular dados. Iniciando pelo *TServicoArea*, o sistema cliente executa os métodos *ContaRegistros* e *SelecionaRegistro* para fazer a visualização dos dados já existentes. Se necessário, são disparadas chamadas para fazer a manipulação dos dados, neste caso, *InserArea*, *AtualizaArea* e *DeletaArea*. Essa regra é seguida na parametrização de todos os objetos.

Figura 7 – DIAGRAMA DE SEQUÊNCIA “PARAMETRIZAÇÃO DO SISTEMA”

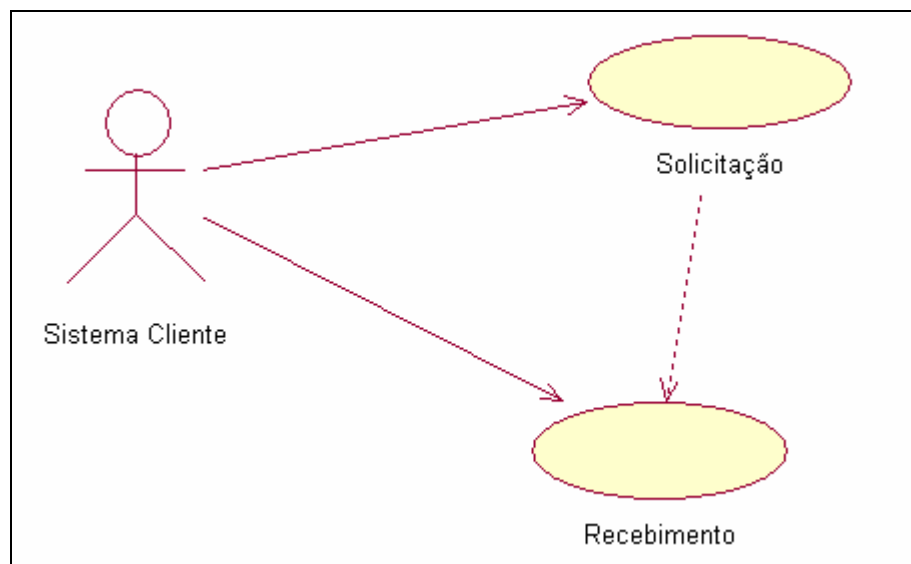


5.2.2.2 SOLICITAÇÃO E RECEBIMENTO DE CATÁLOGOS

A solicitação e o recebimento de catálogos são talvez os processos mais importantes para ilustrar a funcionalidade da aplicação. Este diagrama de caso de uso descreve o processo em que o sistema da empresa que quer solicitar catálogos eletrônicos cadastra suas necessidades de compra de produtos no sistema na forma de catálogos. Depois de cadastrada uma solicitação pode-se verificar periodicamente o recebimento de catálogos para tal solicitação. Uma solicitação pode receber catálogos até que o prazo seja expirado.

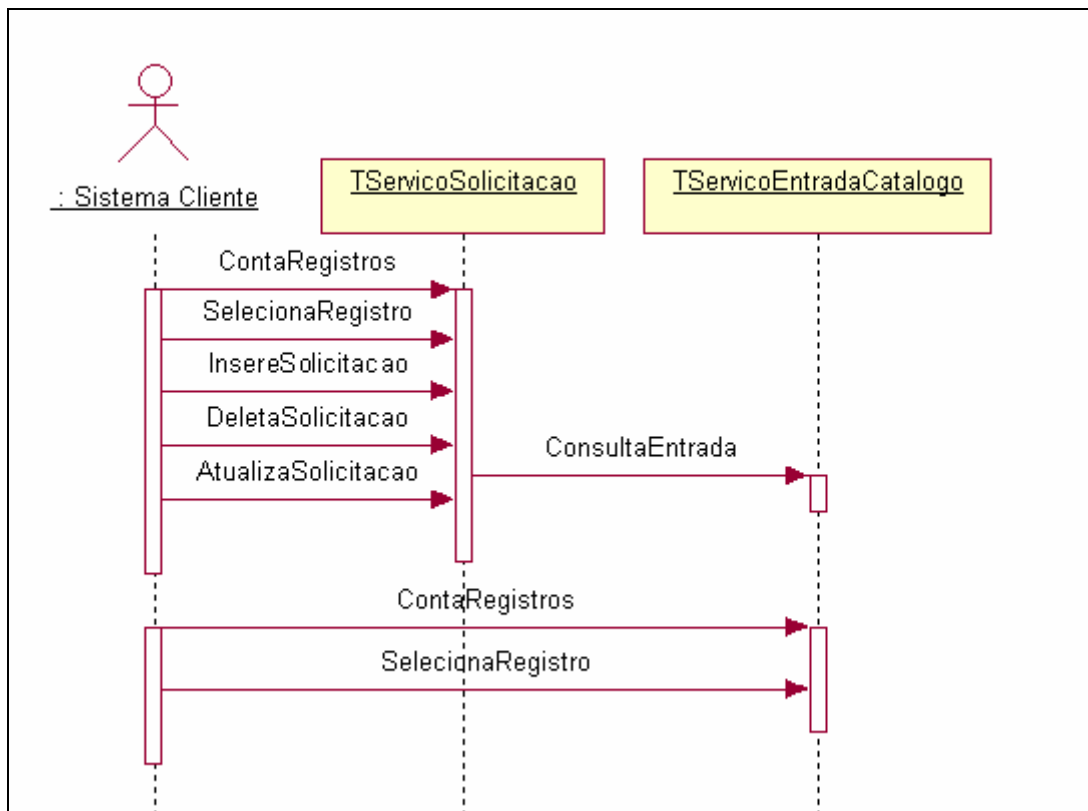
A figura 8 apresenta o diagrama de caso de uso onde o sistema da empresa que deseja solicitar catálogos faz o cadastro da solicitação e baseado nas solicitações cadastradas pode consultar o recebimento de catálogos.

Figura 8 – CASO DE USO “SOLICITAÇÃO E RECEBIMENTO DE CATÁLOGOS”



Observando ainda o diagrama de seqüência correspondente à solicitação e recebimento de catálogos, representado na figura 9 pode-se notar todo o fluxo de mensagens entre os objetos para efetivar a solicitação ou o recebimento de catálogos.

A classe *TServicoSolicitacao* é utilizada para efetivar o processo de solicitação, e a classe *TServicoEntradaCatalogos*, para consultar o recebimento.

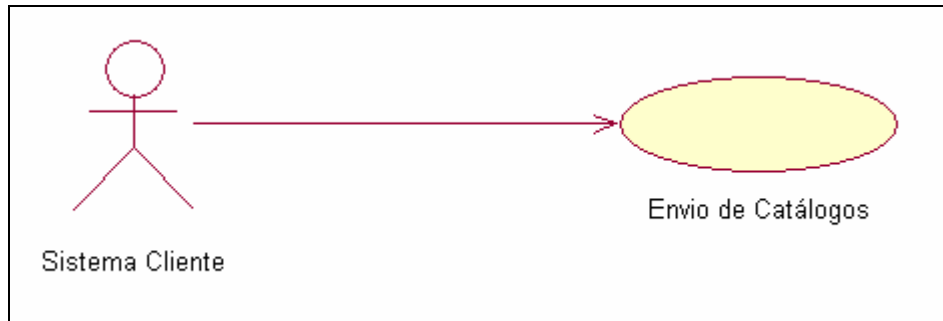
Figura 9 – DIAGRAMA DE SEQUÊNCIA “SOLICITAÇÃO E RECEBIMENTO”

5.2.2.3 ENVIO DE CATÁLOGOS

Dos três processos principais que ocorrem na interação entre outros sistemas e a aplicação principal, o envio de catálogos é talvez o mais complexo por demandar uma série de consistências relativas à solicitação. Neste diagrama de caso de uso é descrito o processo onde a empresa que deseja enviar catálogos verifica dentro da aplicação de troca de catálogos quais são as necessidades dos compradores e envia catálogos de produtos para essas empresas baseado nos parâmetros cadastrados por elas também no mesmo sistema. A empresa que deseja enviar os catálogos prepara um sistema baseado no seu estoque de produtos para acessar a aplicação de troca de catálogos. Acessando esta aplicação são feitas consultas por área para verificar quais empresas estão com solicitações de catálogo em aberto. Identificando essas solicitações a empresa que deseja enviar cadastros verifica em seus estoques quais são os produtos disponíveis que possam atender os parâmetros da solicitação, tais como faixa de preço e quantidade. Se qualquer necessidade de uma solicitação for atendida pode-se então enviar um catálogo contendo informações sobre os produtos candidatos a atenderem tal solicitação.

A figura 10 apresenta o diagrama de caso de uso onde uma empresa fornecedora utiliza seu sistema para acessar o sistema de troca de catálogos e enviar as solicitações.

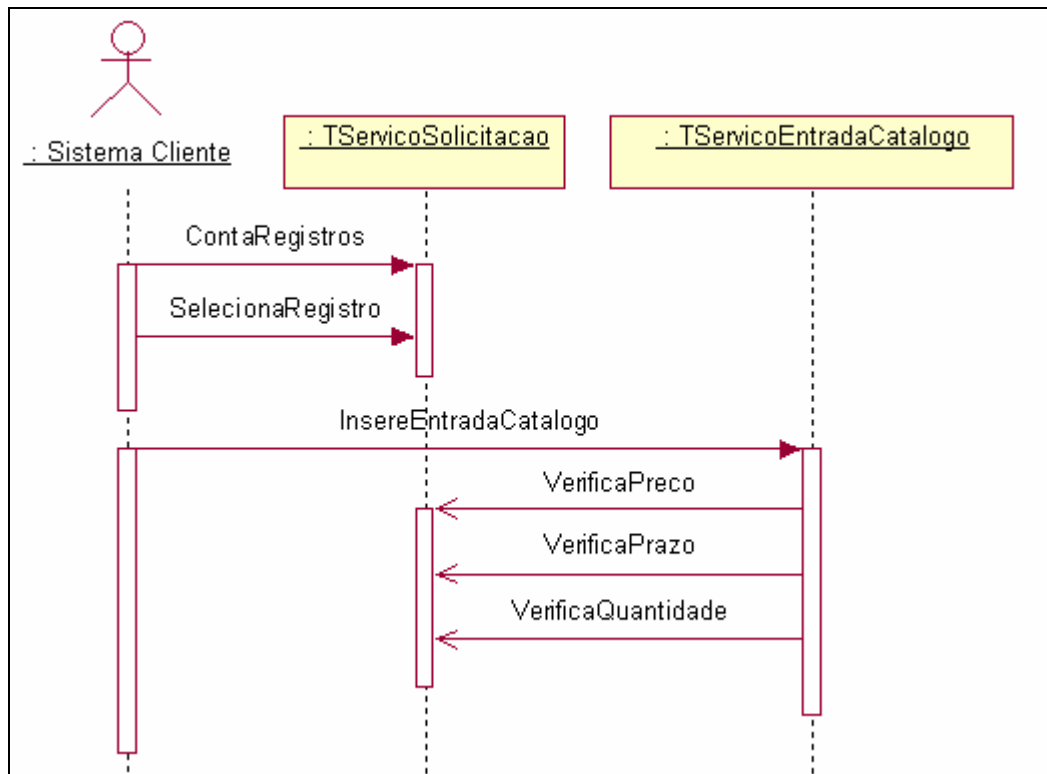
Figura 10 – CASO DE USO “ENVIO DE CATÁLOGOS”



Observando o diagrama de seqüência deste caso de uso que está disposto na figura 11, pode-se perceber como funciona exatamente o processo de envio de catálogos, baseando-se no fluxo da troca de mensagens entre os objetos.

Para enviar um catálogo o sistema cliente deve fazer acesso às solicitações das empresas que estão querendo receber catálogos. Para efetuar essa verificação deve ser criada a classe *TServicoSolicitacao* que permite listar todas as solicitações feitas que estão cadastradas na aplicação de troca de catálogos e que estão com o prazo em aberto. A partir daí pode-se escolher as empresas ou as áreas que possuem maior afinidade a receber os catálogos e então partir para a criação da classe *TServicoEntradaCatalogo*. Essa classe está pronta para receber os dados dos produtos sendo que podem ser enviados mais de um item de catálogo para cada solicitação. Verificando no estoque de seu sistema cliente, a empresa que deseja enviar catálogos então submete seus produtos ao envio de catálogos. A solicitação irá consistir se os campos relativos ao produto a ser enviado estão de acordo com a solicitação. Estando de acordo com os parâmetros da solicitação o produto é enviado para a aplicação de troca de catálogos através da solicitação remota do método de gravação *InsererEntradaCatalogo()*.

Figura 11 – DIAGRAMA DE SEQUÊNCIA “ENVIA CATÁLOGO”



5.3 IMPLEMENTAÇÃO

Nesta seção são discutidos todos os aspectos técnicos relativos ao processo de implementação deste conjunto de aplicações que forma o protótipo, pois além da implementação principal existem as implementações dos sistemas que interagem com a aplicação principal.

5.3.1 TÉCNICAS E FERRAMENTAS UTILIZADAS

O protótipo de software para troca de catálogos entre aplicações de comércio eletrônico foi desenvolvido utilizando o ambiente de desenvolvimento *Borland Delphi 6*. Este ambiente permite gerar aplicações compatíveis com o protocolo SOAP. A forma de aplicação escolhida para hospedar os objetos acessíveis via SOAP é chamada *web service*. O banco de dados utilizado para armazenar os dados desta aplicação é o *Microsoft Access XP*, com base nas informações fornecidas por (Morgado, 1996).

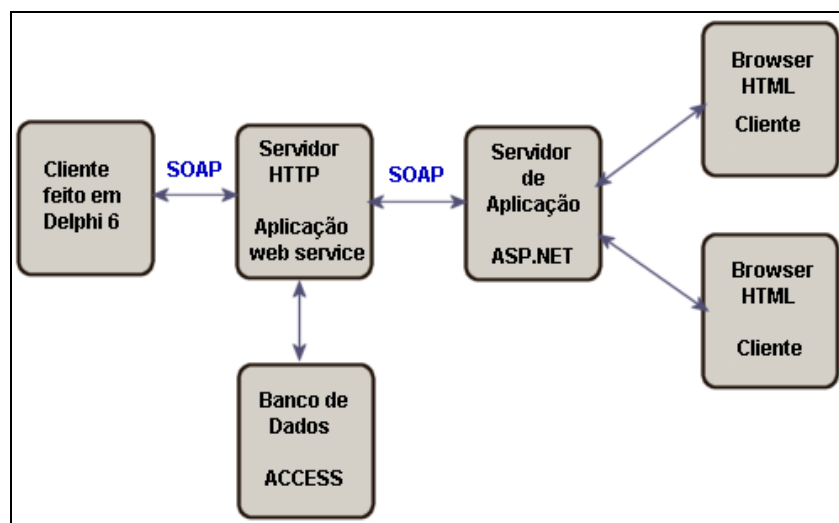
O servidor HTTP utilizado para hospedar o *web service* é o *Internet Information Services* da *Microsoft* e foi escolhido porque além de possibilitar a hospedagem da aplicação

principal, possui todas as extensões necessárias para executar as aplicações cliente que foram desenvolvidas utilizando-se a plataforma de desenvolvimento *Microsoft .Net*.

As aplicações cliente baseadas na plataforma *Microsoft .Net* foram desenvolvidas utilizando o ambiente *Visual Studio .Net*, o modelo de aplicações *Asp.Net*, utilizando a linguagem C#. Essas aplicações então são interpretadas pelo servidor de aplicações *Asp.Net* gerando sua interface em um *browser HTML*.

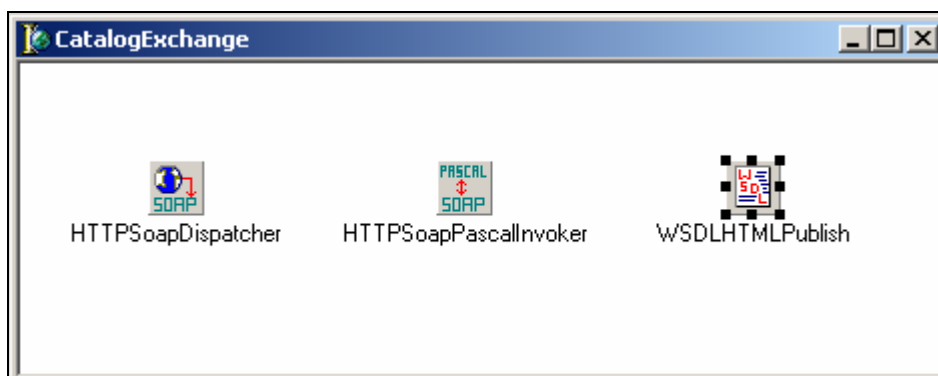
Foi desenvolvida ainda uma aplicação cliente utilizando também o ambiente Borland Delphi 6. O principal motivo de desenvolver as aplicações cliente em plataformas de desenvolvimento diferentes é mostrar que a implementação de objetos baseados em SOAP pode ser acessada por qualquer outra linguagem de programação, mesmo que em ambientes diferentes. A figura 12 exemplifica a interação entre outros aplicativos que se comunicam com a aplicação principal.

Figura 12 – INTERAÇÃO COM A APLICAÇÃO PRINCIPAL



5.3.2 APLICAÇÃO PRINCIPAL

A aplicação principal foi desenvolvida utilizando os recursos do ambiente Borland Delphi 6 para criação de *web services*. Este recurso funciona basicamente como um módulo *web*, que é utilizado como base para desenvolvimentos de CGIs comuns. Porém, para que esta aplicação passe de um CGI comum para um *web service* são adicionados alguns componentes que habilitam as funcionalidades necessárias. A figura 13 mostra os componentes de uma aplicação SOAP em tempo de desenvolvimento, dentro do ambiente.

Figura 13 – APLICAÇÃO SERVIDORA SOAP NO AMBIENTE DELPHI

A funcionalidade destes componentes é peça chave para que o *web service* funcione corretamente e para que as aplicações cliente possam acessar seus objetos. A seguir é descrita especificamente a funcionalidade de cada componente:

- a) o componente `HTTPSoapDispatcher` é responsável por receber uma requisição de um cliente SOAP;
- b) o componente `HTTPSoapPascalInvoker` realiza a tarefa de transformar as requisições SOAP em chamadas de interfaces implementadas em Pascal;
- c) o componente `WSDLHTMLPublish` é utilizado para retornar a definição dos objetos no formato WSDL.

Depois de criar a estrutura da aplicação é necessário implementar uma interface para cada objeto que se queira possibilitar o acesso via SOAP. Este objeto deve ser herdado da classe `TInvokableClass`. Todos os objetos deste protótipo são herdados da classe `TGenericService` pois esta faz herança da `TInvokableClass`. Além disso a interface implementada deve ter como base a interface `IInvokable`.

O quadro 23 mostra a definição da classe `TServicoArea` e o quadro 24 mostra a definição da interface `IServicoArea`.

Quadro 23 – DEFINIÇÃO DA CLASSE TSEVICAOAREA

```

TServiceArea = class(TGenericService, IServiceArea)
private
  function GetCodigo: Integer;
  procedure SetCodigo(Value: Integer);
  function GetDescricao: String;
  procedure SetDescricao(Value: String);
protected
  function GetTableName: String; override;
  function Posiciona(_Codigo: Integer): Boolean;
public
  procedure InsereArea(_Codigo: Integer; const _Descricao: String); stdcall;
  procedure AtualizaArea(_Codigo: Integer; const _Descricao: String); stdcall;
  procedure DeletaArea(_Codigo: Integer); stdcall;
  function ConsultaArea(_Codigo: Integer; var Descricao_: WideString): Boolean; stdcall;
  function ContaRegistros: Integer; stdcall;
  function SelecionaRegistro(BookMark: Integer;
    var Codigo_: Integer;
    var Descricao_: WideString
  ): Boolean; stdcall;
  property Codigo: Integer read GetCodigo write SetCodigo;
  property Descricao: String read GetDescricao write SetDescricao;
end;

```

Pode-se notar que a classe *TServiceArea* implementa a interface *IServiceArea*. A aplicação SOAP desenvolvida em Delphi utiliza essa interface para fazer acesso ao objeto.

Quadro 24 – DEFINIÇÃO DA INTERFACE ISERVICOAREA

```

IServiceArea = interface(IInvokable)
['{CC28B2A3-6374-4FE5-BA0D-D8F673720B24}']

  procedure InsereArea(_Codigo: Integer; const _Descricao: String); stdcall;

  procedure AtualizaArea(_Codigo: Integer; const _Descricao: String); stdcall;

  procedure DeletaArea(_Codigo: Integer); stdcall;

  function ConsultaArea(_Codigo: Integer; var Descricao_: WideString): Boolean; stdcall;

  function ContaRegistros: Integer; stdcall;

  function SelecionaRegistro(BookMark: Integer;
    var Codigo_: Integer;
    var Descricao_: WideString
  ): Boolean; stdcall;
end;

```

Além de implementar a classe e a interface ainda é necessário registrá-las para que os componentes definidos anteriormente consigam interpretar as requisições SOAP e ligá-las aos objetos implementados. O registro de todas as interfaces do protótipo foi implementado na *unit URegisterAllClasses*, e parte deste fonte pode ser visto no quadro 25. Cada classe e cada interface são registradas separadamente, e sua disponibilização para requisições SOAP só torna-se possível a partir deste ponto.

Quadro 25 – REGISTRO DAS INTERFACES E CLASSES

```

uses
  InvokeRegistry;

initialization
  InvRegistry.RegisterInvokableClass(UServiceArea.TServicoArea);
  InvRegistry.RegisterInvokableClass(UServiceTipo.TServicoTipo);
  InvRegistry.RegisterInvokableClass(UServiceEmpresa.TServicoEmpresa);
  InvRegistry.RegisterInvokableClass(UServiceProduto.TServicoProduto);
  InvRegistry.RegisterInvokableClass(UServiceSecao.TServicoSecao);
  InvRegistry.RegisterInvokableClass(UServiceSolicitacao.TServicoSolicitacao);
  InvRegistry.RegisterInvokableClass(UServiceEntradaCatalogo.TServicoEntradaCatalogo);

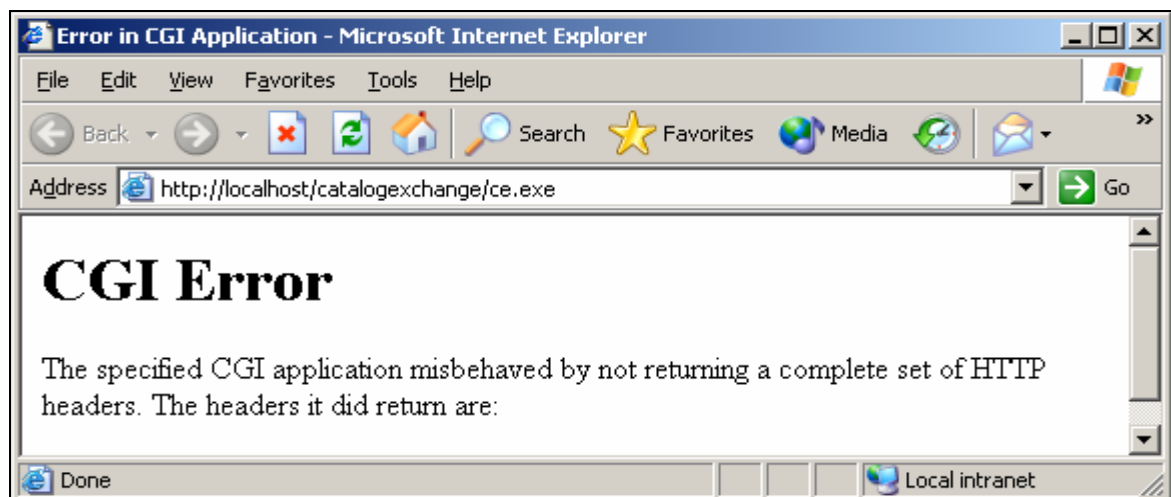
  InvRegistry.RegisterInterface(TypeInfo(UServiceArea.IServicoArea));
  InvRegistry.RegisterInterface(TypeInfo(UServiceTipo.IServicoTipo));
  InvRegistry.RegisterInterface(TypeInfo(UServiceEmpresa.IServicoEmpresa));
  InvRegistry.RegisterInterface(TypeInfo(UServiceProduto.IServicoProduto));
  InvRegistry.RegisterInterface(TypeInfo(UServiceSecao.IServicoSecao));
  InvRegistry.RegisterInterface(TypeInfo(UServiceSolicitacao.IServicoSolicitacao));
  InvRegistry.RegisterInterface(TypeInfo(UServiceEntradaCatalogo.IServicoEntradaCatalogo));
...

```

A porção de código indicada no quadro 25 é executada cada vez que uma aplicação cliente faz uma requisição. A partir das interfaces registradas a aplicação procura qual é a interface utilizada para atender a solicitação e cria uma instância da classe correspondente para atendê-la.

Após compilar a aplicação é possível verificar todos os objetos que estão publicados através da solicitação do WSDL. O resultado da aplicação compilada é um programa que executa em um servidor web, chamado de *web service*. Essa aplicação foi disponibilizada no diretório *CatalogExchange* do servidor HTTP e pode ser acessada a partir do site <http://localhost/CatalogExchange/ce.exe>. A aplicação está então disponível para acesso a partir de qualquer cliente que implemente o protocolo SOAP. Se alguém tentar acessar diretamente este *site*, vai receber uma mensagem de erro, conforme a figura 14.

Figura 14 – ERRO AO ACESSAR DIRETAMENTE A APLICAÇÃO



Este erro acontece pois esta aplicação não deve ser acessada diretamente por um *browser*, visto que o papel de um *web service* não é apresentar dados para um navegador, e sim permitir que objetos sejam acessados remotamente.

A interação entre os objetos disponíveis e outras aplicações é implementada em clientes SOAP, também chamados de *client web services*. Estes objetos acessam a aplicação principal utilizando o protocolo SOAP e trocam mensagens, disparando métodos do servidor, enviando e recebendo dados conforme seu retorno.

Uma aplicação cliente acessa o servidor a partir de regras definidas no WSDL de cada aplicação servidora. A implementação baseada no ambiente *Delphi* disponibiliza o WSDL adicionando-se o sufixo */wsdl/NomeDaInterface*. Existe ainda um recurso interessante que gera um documento HTML com a lista de todos os objetos disponíveis e o respectivo link para seu WSDL. Este documento pode ser obtido adicionando apenas o sufixo */wsdl* ao final do caminho do servidor onde se encontra a aplicação. A figura 15 mostra o documento com todas as interfaces pertencentes ao protótipo. Esse documento é gerado fazendo-se uma requisição ao site *http://localhost/catalogexchange/ce.exe/wsdl*.

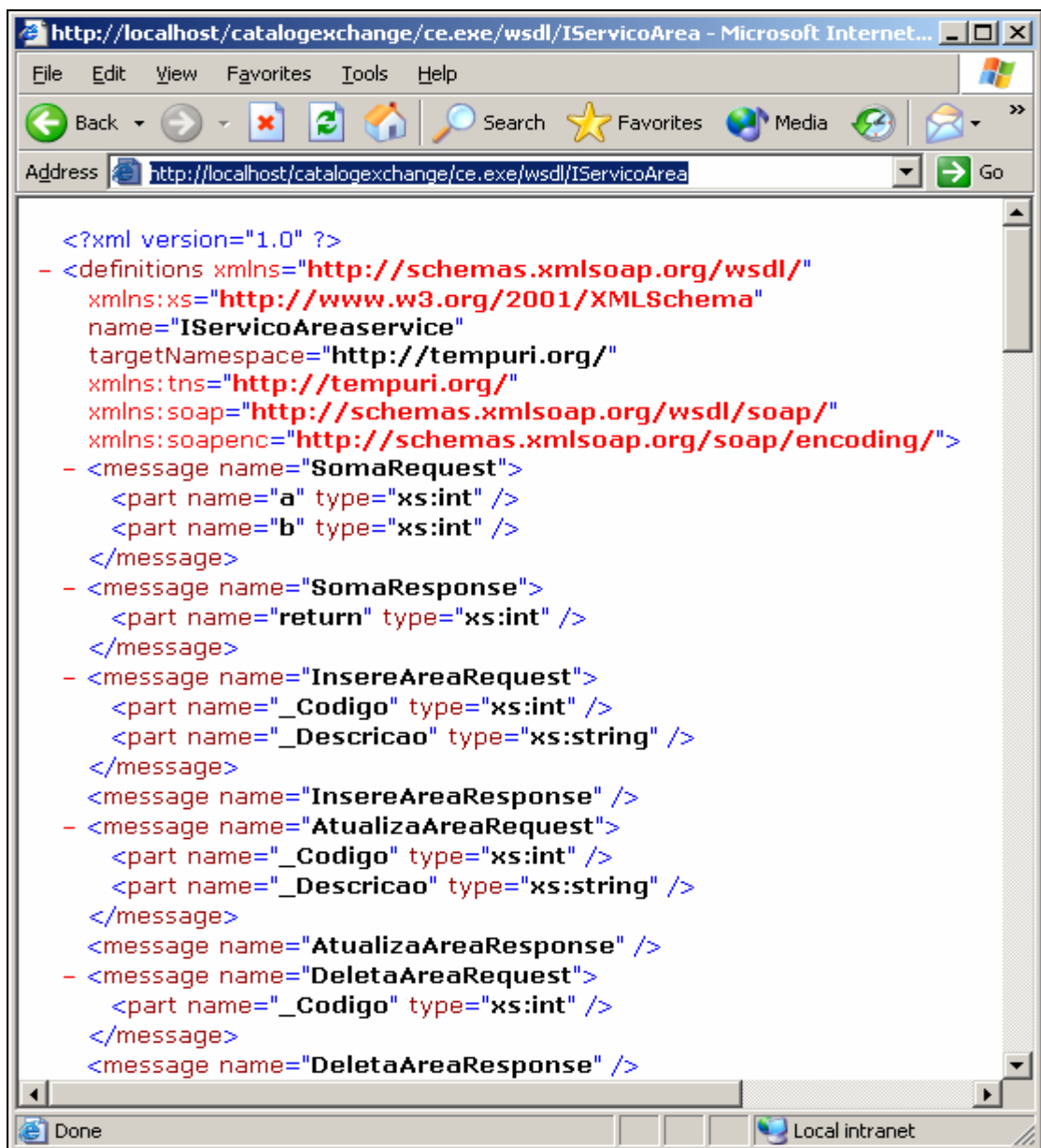
Figura 15 – LISTA DE INTERFACES DO PROTÓTIPO

Port Type	Namespace URI	Documentation	WSDL
IServicoArea	urn:UServiceArea-IServicoArea		IServicoArea
IServicoTipo	urn:UServiceTipo-IServicoTipo		IServicoTipo
IServicoEmpresa	urn:UServiceEmpresa-IServicoEmpresa		IServicoEmpresa
IServicoProduto	urn:UServiceProduto-IServicoProduto		IServicoProduto
IServicoSecao	urn:UServiceSecao-IServicoSecao		IServicoSecao
IServicoSolicitacao	urn:UServiceSolicitacao-IServicoSolicitacao		IServicoSolicitacao
IServicoEntradaCatalogo	urn:UServiceEntradaCatalogo-IServicoEntradaCatalogo		IServicoEntradaCatalogo
IWSDLPublish	http://www.borland.com/namespaces/Types	Lists all the PortTypes published by this Service	IWSDLPublish

A partir deste documento fica mais fácil descobrir quais são os objetos publicados para acesso e pode-se partir para a solicitação do WSDL de cada objeto. O documento listado na figura 15 mostra que para cada classe implementada no protótipo está disponível para acesso via SOAP.

Na coluna titulada WSDL são exibidos *links* para os documentos WSDL de cada interface. Fazendo uma chamada pelo *browser* ao link relativo à interface *IServicoArea* será obtido um documento XML que contém a descrição detalhada de todas as características dos métodos da interface *IservicoArea*. A figura 16 mostra o resultado desta solicitação.

Figura 16 – WSDL DA INTERFACE ISERVICOAREA



```

<?xml version="1.0" ?>
- <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  name="IServicoAreaservice"
  targetNamespace="http://tempuri.org/"
  xmlns:tns="http://tempuri.org/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
- <message name="SomaRequest">
  <part name="a" type="xs:int" />
  <part name="b" type="xs:int" />
</message>
- <message name="SomaResponse">
  <part name="return" type="xs:int" />
</message>
- <message name="InsereAreaRequest">
  <part name="_Codigo" type="xs:int" />
  <part name="_Descricao" type="xs:string" />
</message>
<message name="InsereAreaResponse" />
- <message name="AtualizaAreaRequest">
  <part name="_Codigo" type="xs:int" />
  <part name="_Descricao" type="xs:string" />
</message>
<message name="AtualizaAreaResponse" />
- <message name="DeletaAreaRequest">
  <part name="_Codigo" type="xs:int" />
</message>
<message name="DeletaAreaResponse" />

```

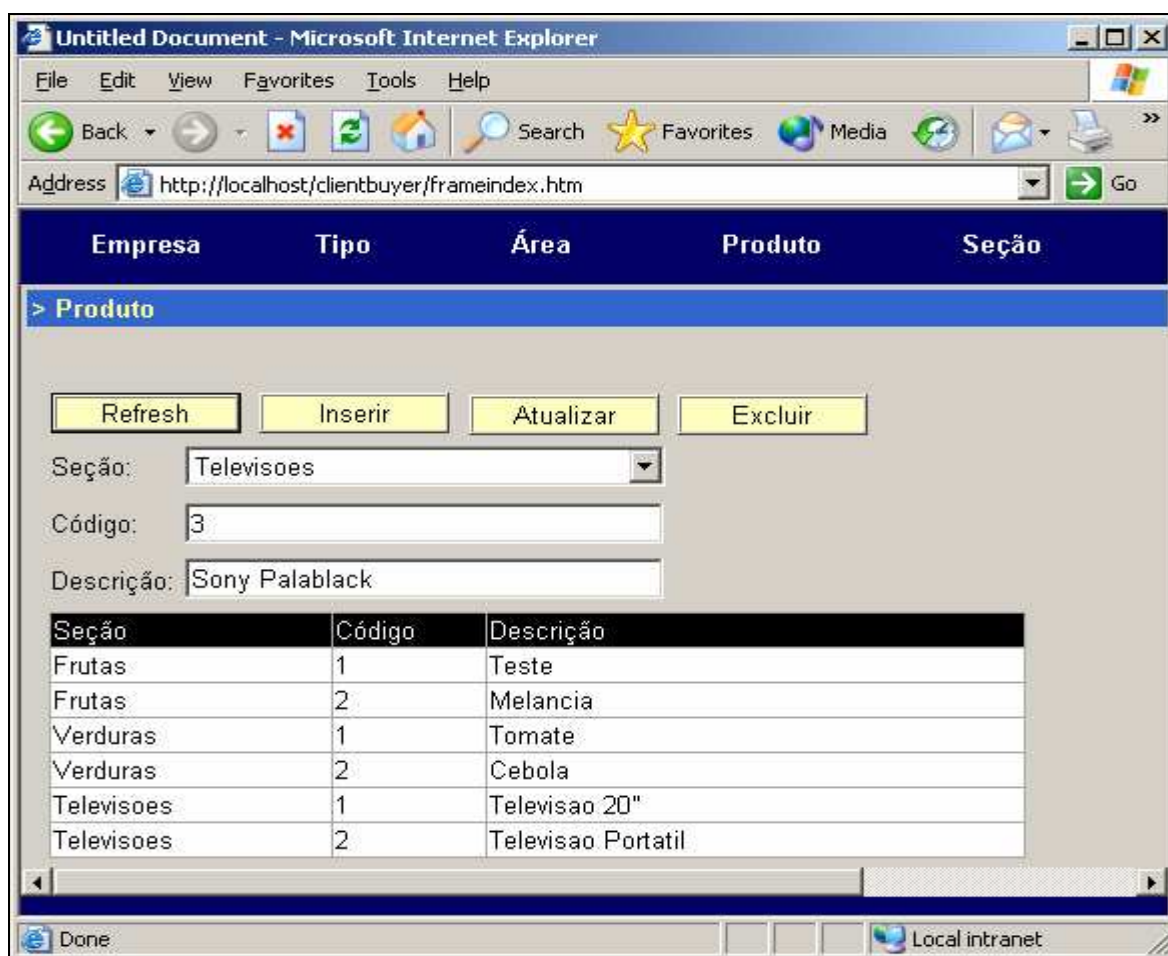
A figura 16 mostra apenas uma parte do documento WSDL da interface *IServicoArea*. Este documento pode ser extraído por completo acessando-se o site <http://localhost/catalogexchange/ce.exe/wsd/IServicoArea>. Neste ponto do trabalho o WSDL é apenas utilizado para visualizar os métodos da interface *IServicoArea*. Nas próximas seções cada WSDL será utilizado para implementar o acesso das aplicações cliente.

5.3.2.1 PARAMETRIZAÇÃO DO SISTEMA

Na implementação das telas de entrada de dados para parametrização dos objetos principais do sistema foi utilizada a ferramenta *Visual Studio .Net*. O objetivo desta implementação é criar recursos para acessar os objetos da aplicação principal.

Para cada objeto foi criada uma tela que permite o cadastro e a visualização dos dados. Essas telas fazem acesso aos métodos dos objetos disponíveis na aplicação principal. O modelo básico pode ser observado na figura 17.

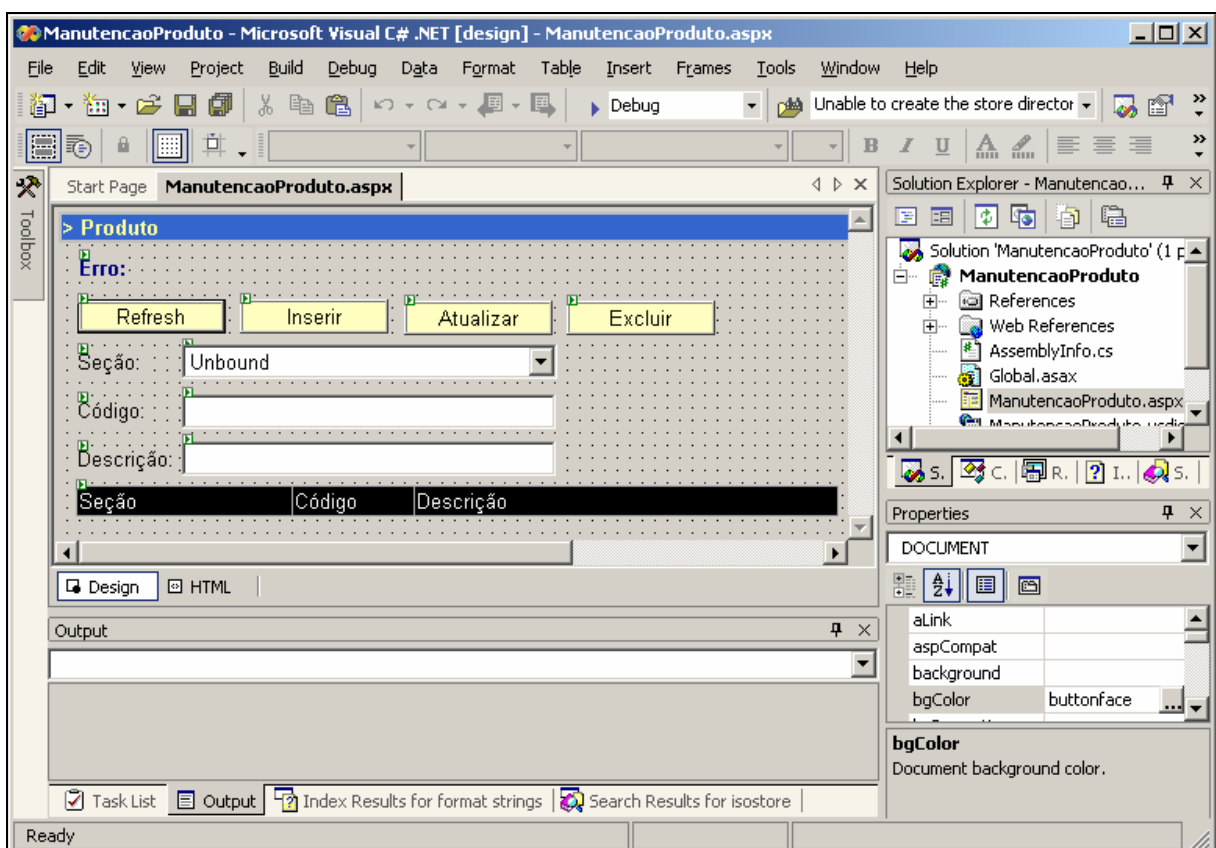
Figura 17 – MODELO DAS TELAS DE CADASTRO



Este modelo de tela é utilizado em toda parametrização do sistema e foi implementado em linguagem C#. O ambiente *Visual Studio .Net* oferece vários recursos para tornar a implementação de aplicações que acessam *web services* fácil e transparente. Após montar o layout básico da página são adicionados objetos para fazer acesso a aplicação principal. Este processo é bastante intuitivo e não requer a codificação manual de grande parte do código fonte necessário para gerar a tela.

A figura 18 mostra o layout da tela em tempo de desenho, dentro do *Visual Studio .Net*. Para adicionar os objetos que fazem acesso aos *web services* existe um *wizard* que monta o objeto de acesso. A figura 19 mostra este *wizard* sendo utilizado para gerar um objeto de acesso a interface *IServicoProduto*. O quadro 26 mostra um fragmento do código gerado por este *wizard*. O código completo pode ser encontrado no anexo 1 deste trabalho.

Figura 18 – LAYOUT BÁSICO DA TELA DE CADASTRO



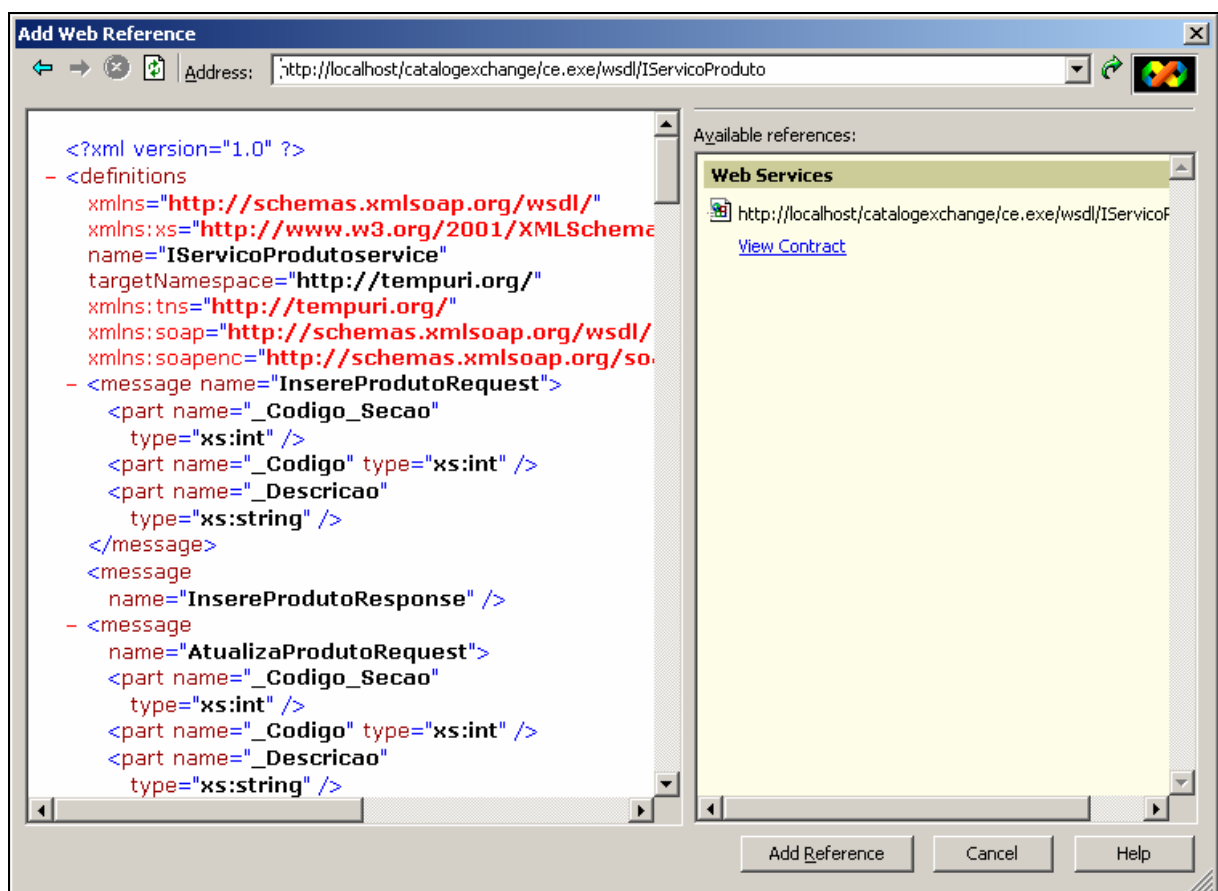
Quadro 26 – CODIGO DO OBJETO DE ACESSO A INTERFACE ISERVICOPRODUTO

```

/// <remarks/>
[System.Web.Services.Protocols.SoapRpcMethodAttribute("urn:UserServiceProduto-
IServicoProduto#ConsultaProduto", RequestNamespace="urn:UserServiceProduto-IServicoProduto",
ResponseNamespace="urn:UserServiceProduto-IServicoProduto")]
[return: System.Xml.Serialization.SoapElementAttribute("return")]
public bool ConsultaProduto(int _Codigo_Secao, int _Codigo, ref string Descricao_) {
    object[] results = this.Invoke("ConsultaProduto", new object[] {
        _Codigo_Secao,
        _Codigo,
        Descricao_});
    Descricao_ = ((string)(results[1]));
    return ((bool)(results[0]));
}

```

Figura 19 – WIZARD PARA GERAR OBJETO DE ACESSO



Além de montar o *layout* da tela de definir os objetos de acesso são necessárias algumas linhas de código C# para manipular os dados obtidos no acesso aos objetos da aplicação principal. Essas linhas de código estão normalmente ligadas aos eventos dos botões de manipulação (inserir, atualizar, excluir, refresh) e basicamente tratam de passar os parâmetros para os objetos que fazem acesso a aplicação principal. Um exemplo deste código é mostrado no quadro 27, onde é tratado o evento de exclusão.

Quadro 27 – EVENTO DE EXCLUSÃO

```
private void btExcluir_Click(object sender, System.EventArgs e)
{
    lbError.Visible = false;
    if (tbCodigo.Text.Trim() == "")
    {
        lbError.Visible = true;
        lbError.Text = "Erro: Informe o campo Código";
    }
    else
    {
        int Secao = Convert.ToInt32(ddArea.SelectedItem.Value);
        int Codigo = Convert.ToInt32(tbCodigo.Text);
        try
        {
            IProduto.DeletaProduto(Secao, Codigo);
        }
        catch(Exception ecp)
        {
            lbError.Visible = true;
            lbError.Text = ecp.Message;
        }
    }
    btRefresh_Click(sender, e);
}
```

5.3.2.2 SOLICITAÇÃO E RECEBIMENTO DE CATÁLOGOS

Na implementação das telas de solicitação e recebimento de catálogos foi utilizada a ferramenta *Visual Studio .Net*. O objetivo desta implementação é exemplificar como uma aplicação de comércio eletrônico pode integrar-se à aplicação principal para solicitar e receber catálogos. Para tanto foram criadas duas telas, utilizando as mesmas técnicas da aplicação anterior.

5.3.2.3 ENVIO DE CATÁLOGOS

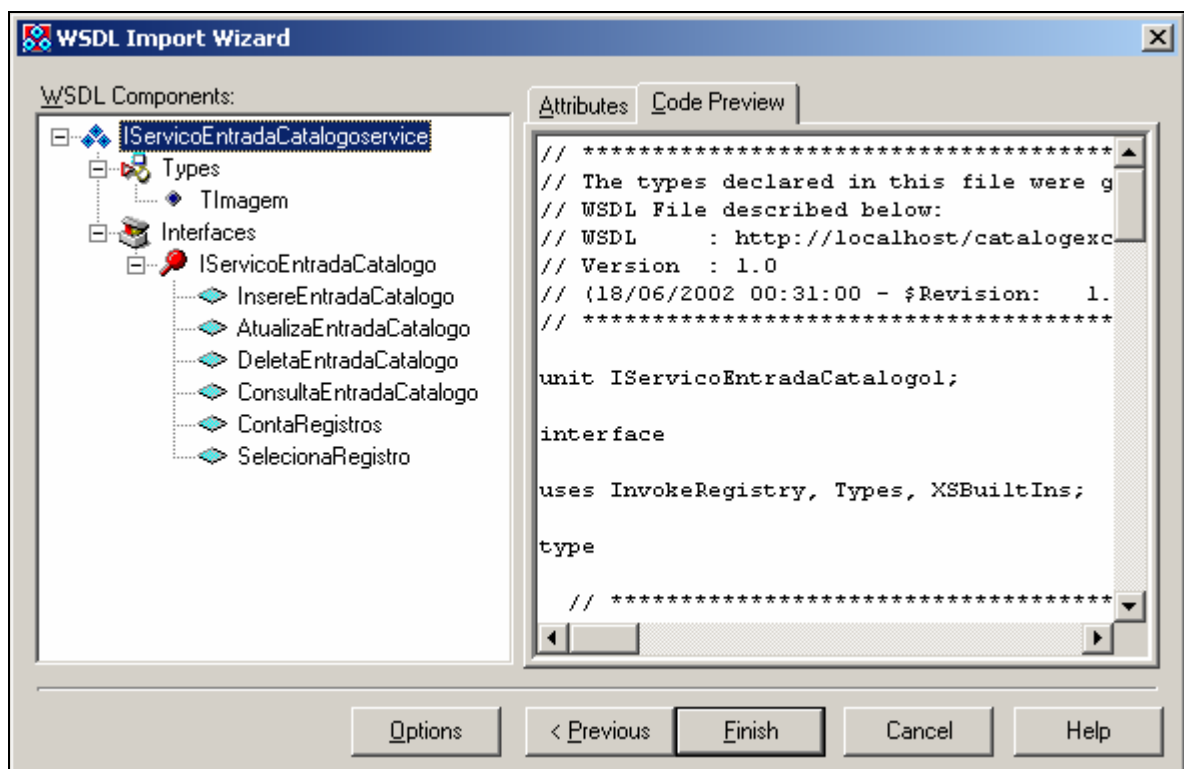
Para implementação da aplicação de envio de catálogos foi utilizada a ferramenta *Borland Delphi 6*. O objetivo desta implementação é mostrar que as aplicações desenvolvidas em compatibilidade com o protocolo SOAP podem ser acessadas a partir de diferentes linguagens de programação, e de diferentes ambientes.

Esta aplicação simula uma aplicação básica que manipula o estoque de produtos de uma empresa que deseja enviar catálogos relativos a esses produtos a empresas possivelmente compradoras. Para isto esta aplicação interage com a aplicação principal para verificar quais são as empresas que estão cadastradas na aplicação e desejam receber catálogos. Verificadas as solicitações podem ser enviados catálogos para suprir essas solicitações.

A implementação de clientes para aplicações SOAP, no ambiente *Borland Delphi 6* inicia como a implementação de uma aplicação normal. A diferença é que nas aplicações típicas a manipulação dos dados é feita utilizando-se diretamente o banco de dados, e em uma aplicação que utiliza objetos localizados em outro servidor, a manipulação de dados é feita através de requisições a objetos remotos utilizando o protocolo SOAP.

O ambiente *Borland Delphi 6*, como o *Visual Studio .Net*, também fornece um *wizard* para criar algum método de acesso aos objetos remotos. Porém, neste caso, ao invés de um objeto é gerada uma função que retorna uma interface. Esta função manipula um objeto chamado *THHTPRIO* que conecta a aplicação principal, faz a requisição e retorna uma interface capaz de acessar os métodos da aplicação principal (Cantu 2001). A figura 20 ilustra o *wizard* utilizado para gerar a interface de acesso a interface *IServicoEntradaCatalogo*.

Figura 20 – WIZARD PARA GERAR INTERFACE DE ACESSO



Este *wizard* gera uma *unit* que contém uma função para retornar a interface de acesso aos métodos do objeto do servidor. Essa função é nomeada com o prefixo *Get* e o nome do objeto ou interface a ser acessada.

Quadro 28 – FUNÇÃO PARA RETORNAR A INTERFACE

```

function GetIServicoEntradaCatalogo(UseWSDL: Boolean; Addr: string): IServicoEntradaCatalogo;
const
  defWSDL = 'http://localhost/catalogexchange/ce.exe/wsdl/IServicoEntradaCatalogo';
  defURL  = 'http://localhost/catalogexchange/ce.exe/soap/IServicoEntradaCatalogo';
  defSvc  = 'IServicoEntradaCatalogoservice';
  defPrt  = 'IServicoEntradaCatalogoPort';
var
  RIO: THTTTPRIO;
begin
  Result := nil;
  if (Addr = '') then
  begin
    if UseWSDL then
      Addr := defWSDL;
    else
      Addr := defURL;
    end;
  RIO := THTTTPRIO.Create(nil);
  try
    if UseWSDL then
    begin
      RIO.WSDLLocation := Addr;
      RIO.Service := defSvc;
      RIO.Port := defPrt;
    end else
      RIO.URL := Addr;
      Result := (RIO as IServicoEntradaCatalogo);
    finally
      if Result = nil then
        RIO.Free;
      end;
    end;
  end;
end;

```

Além da função são gerados tipos de dados contidos no objeto e a interface de acesso. O exemplo completo da *unit* gerada pode ser encontrado no anexo 3.

A partir das interfaces de acesso aos objetos da aplicação principal, são verificados as necessidades de envio de catálogos e os catálogos podem ser enviados. O acesso aos objetos é efetuado quando os eventos dos botões são disparados. O quadro 29 mostra o evento em que o botão para buscar as empresas vendedoras é disparado.

Quadro 29 – EVENTO QUE FAZ ACESSO A APLICAÇÃO PRINCIPAL

```

procedure TfrmSeller.btnAreaClick(Sender: TObject);
var
  IArea: IServicoArea;
  Mark, i: Integer;

  Codigo_: Integer;
  Descricao_: String;

  Rec: TRegistroServicoArea;
begin
  ClearArea;
  IArea := GetIServicoArea;
  try
    DisableEmpresa;
    Mark := IArea.ContaRegistros;
    for i := 1 to Mark do
    begin
      if IArea.SelecionaRegistro(i, Codigo_, Descricao_) then
        begin

```

```

Rec := TRegistroServicoArea.Create;
try
  Rec.Codigo := Codigo_;
  Rec.Descricao := Descricao_;
  FAuxArea.AddItem(i, cbArea.Items.Count, Rec);
  cbArea.Items.Add(Descricao_);
except
  Rec.Free;
  raise;
end;
end;
if cbArea.Items.Count > 0 then
begin
  cbArea.ItemIndex := 0;
  cbArea.Enabled := True;
  EnableEmpresa;
end;
finally
  IArea := nil;
end;
end;

```

5.3.3 OPERACIONALIDADE DA IMPLEMENTAÇÃO

O processo de troca de catálogos entre aplicações de comércio eletrônico utilizando o protótipo desenvolvido neste trabalho pode ser dividido em três partes distintas:

- a) parametrização do sistema: onde o sistema recebe as informações sobre as empresas e produtos a serem manipuladores durante o processo;
- b) solicitação e recebimento: baseado nas parametrizações anteriores é possível criar solicitações de entrada de catálogo por parte das empresas compradoras. O recebimento é verificado após o envio de catálogos (item c);
- c) envio de catálogos: acessando os objetos da aplicação de troca de catálogos é possível verificar que empresas estão solicitando catálogos e proceder o envio.

Para exemplificar a funcionalidade de todas essas partes o sistema receberá o cadastro de empresas dos ramos de eletrodomésticos e som automotivo. A empresa do ramo de eletrodomésticos desejará receber catálogos de aparelhos de ar-condicionado e máquinas de lavar roupa. A empresa do ramo de som automotivo desejará receber catálogos de aparelhos de cd *player* do tipo simples ou *changer*.

Serão efetuados todos os cadastros necessários, descritos mais detalhadamente nos cenários a seguir, e a partir destes cadastros serão criadas as solicitações de catálogos. A aplicação cliente de envio de catálogos então verificará a necessidade de catálogos e

procederá com o processo de envio. Após isto pode ser verificado o recebimento de catálogos por parte das empresas que desejam receber os catálogos.

5.3.3.1 CENÁRIO “PARAMETRIZAÇÃO DO SISTEMA”

O cenário “parametrização do sistema” ocorre quando existe a necessidade do sistema ser preparado para um processo de troca de catálogos.

Esta parametrização é feita através do acesso via *browser* à aplicação de parametrização, que foi implementada segundo os conceitos de uma página *Web*, fazendo acesso aos objetos da aplicação principal. A tela inicial da aplicação de parametrização pode ser observada na figura 21, onde é exibido um menu contendo os cadastros a serem parametrizados (empresa, tipo, área, produto, seção) e a primeira página de cadastro relativa ao cadastro de empresas.

Figura 21 – TELA PRINCIPAL DA APLICAÇÃO DE PARAMETRIZAÇÃO

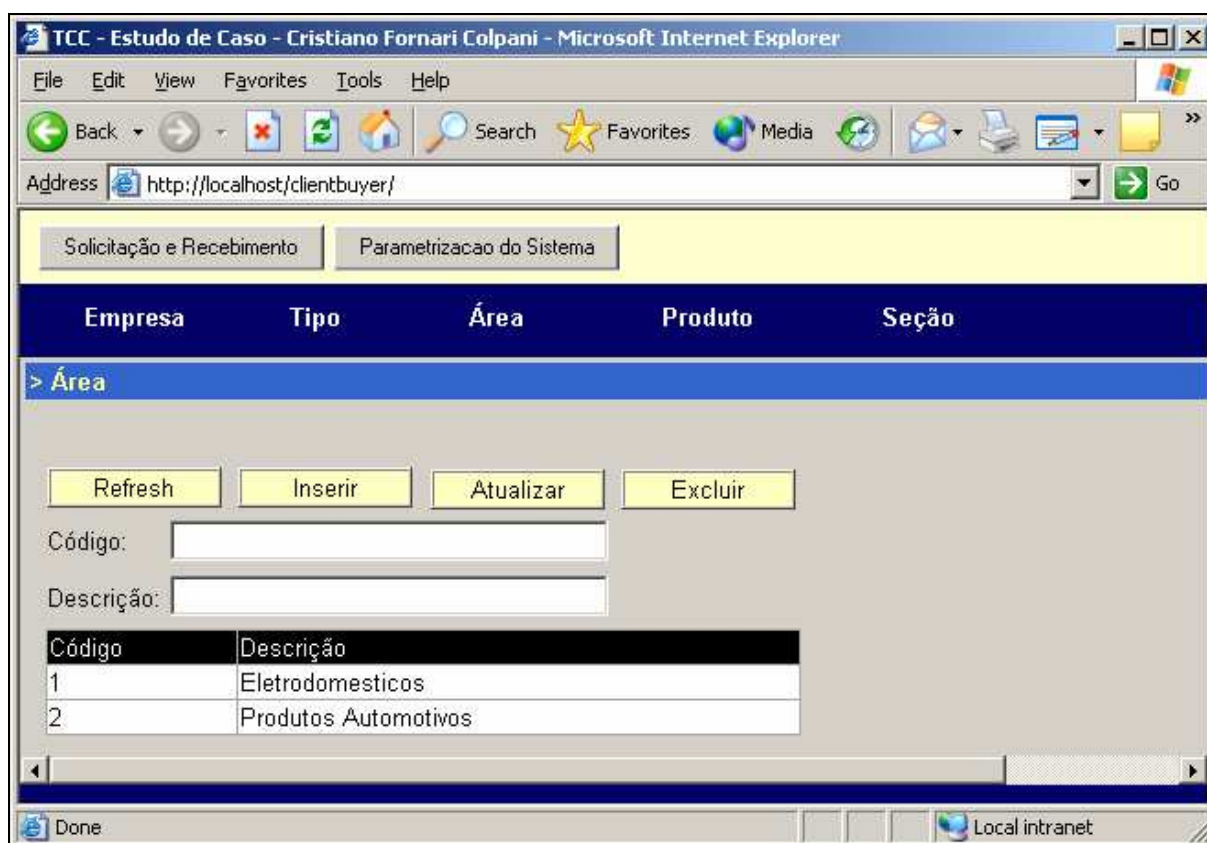
Tipo	Área	Código	Descrição
Vendedora	Eletrodomesticos	1	Tom Eletrodomesticos
Vendedora	Produtos Automotivos	2	Jerry Som Automotivo
Compradora	Eletrodomesticos	3	Garfield Eletro Shop
Compradora	Produtos Automotivos	4	Oddie Acessorios p/ Carro

Ao cadastrar uma empresa deve ser informado seu tipo (vendedora ou compradora), a área de atuação, importante para que as empresas vendedoras possam enviar os catálogos, um código, que é o identificador do registro e uma descrição para identificar a empresa.

A figura 21 já conta com as 4 empresas necessárias ao estudo de caso cadastradas. As empresas cadastradas podem ser de diversas áreas e tipos. Clicando no menu superior na opção “área”, será aberta a tela de customização de área. Este tela é apresentada conforme a figura 22.

Pode-se notar nesta tela também que já existem áreas pré-cadastradas para o desenvolvimento do estudo de caso. Essas áreas podem ser alteradas, incluídas ou excluídas, devendo tomar como preocupação o contexto a que estão relacionadas nas empresas e solicitações de troca.

Figura 22 – PARAMETRIZAÇÃO DE ÁREA



Além de customizar as empresas envolvidas no processo de troca é necessário parametrizar os produtos dos quais os catálogos serão solicitados. Clicando no menu superior, na opção “Seção”, podem ser cadastradas as seções para o posterior cadastro dos produtos.

A figura 23 mostra o cadastro de seção. A seção serve para separar os produtos como se fosse a organização de produtos em uma prateleira de supermercado ou loja de departamentos. A seção é diferente da área visto que compreende uma diferenciação bem específica dos tipos de produto. Já a área é relativa ao ramo da empresa envolvida.

Para que essa diferenciação fique mais clara, pode-se considerar o exemplo utilizado neste estudo de caso, em que a área das empresas compradoras e vendedoras é a área de Eletrodomésticos e os produtos enviados são diferenciados por modelo, e agrupados em seções que compreendem ar-condicionado e lavadoras de roupa.

A figura 24 procura ilustrar este exemplo com o cadastramento de produtos que serão utilizados no processo de troca de catálogos que será descrito nos cenários seguintes.

Figura 23 – PARAMETRIZAÇÃO DE SEÇÃO

Empresa	Tipo	Área	Produto	Seção
> Seção				

Refresh Inserir Atualizar Excluir

Código:

Descrição:

Código	Descrição
1	Condicionadores de Ar
2	Maquinas de Lavar Roupa
3	CD Player Automotivo

Figura 24 – PARAMETRIZAÇÃO PRODUTO



5.3.3.2 CENÁRIO “SOLICITAÇÃO E RECEBIMENTO DE CATÁLOGOS”

O cenário “solicitação e recebimento de catálogos” ocorre quando alguma das empresas compradoras deseja solicitar catálogos para algum produto e posteriormente verifica o recebimento.

A solicitações de catálogos também são cadastradas em uma interface HTML acessada via *browser*. Esta tela pode ser observada na figura 25 e pode ser acessada a partir do menu “Solicitar”.

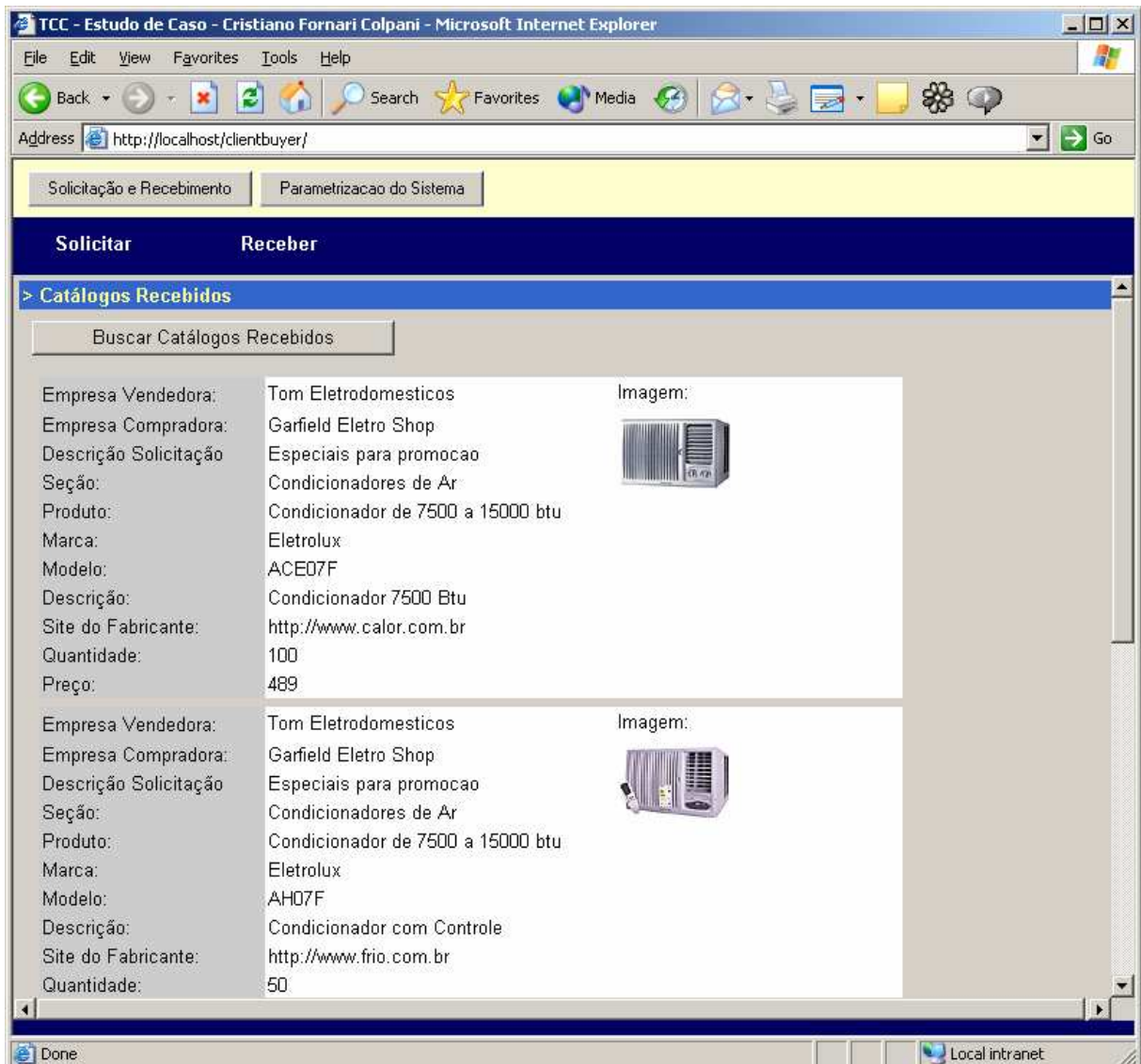
Figura 25 – SOLICITAÇÃO DE CATÁLOGOS

Código	Solicitante	Seção	Produto	P.Inicial	P.Final	Prazo	Descrição
1	Garfield Eletro Shop	Condicionadores de Ar	Condicionador de 7500 a 15000 btu	500	700	21/06/2002	Especiais para promocao
2	Garfield Eletro Shop	Maquinas de Lavar Roupa	Lavad. Normais e Frontais	300	600	28/06/2002	Maquinas Populares
3	Oddie	CD's	CD Simples ou	300	500	28/06/2002	CD's Simples ou

Pode-se observar na figura 25 o cadastramento de 3 solicitações. Duas feitas pela empresa *Garfield Eletro Shop* e uma feita pela empresa *Oddie Som Automotivo*. Essas solicitações serão visualizadas pela aplicação que faz o envio de catálogos que é discutida no tópico a seguir. Após execução do processo descrito no capítulo seguinte deve-se clicar no menu “Receber” para verificar os catálogos que chegaram.

A figura 26 mostra os catálogos que foram recebidos para as solicitações anteriormente cadastradas.

Figura 26 – CATÁLOGOS RECEBIDOS



5.3.3.3 CENÁRIO “ENVIO DE CATÁLOGOS”

O cenário “envia catálogos” ocorre quando é verificada a necessidade de envio de catálogo por alguma empresa vendedora.

A figura 27 mostra a aplicação de envio de catálogos pronta para enviar catálogos. A interface da aplicação de envio de catálogos consiste na seleção da área da empresa que será responsável pela venda, e depois o nome da empresa.

Figura 27 – APLICAÇÃO PARA ENVIO DE CATÁLOGOS

Caso de uso: Envia catálogo

Empresa Vendedora

Área: Eletrodomesticos Empresa: Tom Eletrodomesticos Selecionar

Solicitações por Área

Área: Eletrodomesticos

Código	Solicitante	Seção	Produto	P. Inicial	P. Final	Prazo	Descrição
1	Garfield Eletro Shop	Condicionadores de Ar	Condicionador d...	500.00	700.00	21/06/2002	Especiais para promocao
2	Garfield Eletro Shop	Maquinas de Lavar ...	Lavad. Normais ...	300.00	600.00	28/06/2002	Maquinas Populares

Buscar Solicitações Selecionar

Sistema de Estoque Local

Codigo	Descricao
1	Máquina de Lavar
2	Ar Condicionado
3	Som automotivo

Selecionar Produto Efetivar Envio Selecionar Imagem

Descricao	Marca	Modelo	Quantidade	Preço	Site
Lavadora 5k	Continental	Evolution	100	659	http://www.continental.com.br
Lavadora Smart	Brastemp	BWB22	50	799	http://www.bastemp.com.br
Lavadora Top	Eletrolux	Top 8	30	839	http://www.tectec.com.br
Lavadora Frontal	Bastemp	KitckenEasy	200	799	http://www.bastemp.com.br

Após selecionar a empresa vendedora (botão “Empresa”) é necessário clicar em “Selecionar” (à esquerda da empresa selecionada) para habilitar a próxima parte da parte tela, em que são listadas todas as solicitações de empresas compradoras da área selecionada. Para listar as solicitações é necessário clicar em “Buscar solicitações”. A figura 28 mostra a lista de solicitações da área de *Eletrodomésticos*.

Figura 28 – LISTA DE SOLICITAÇÕES

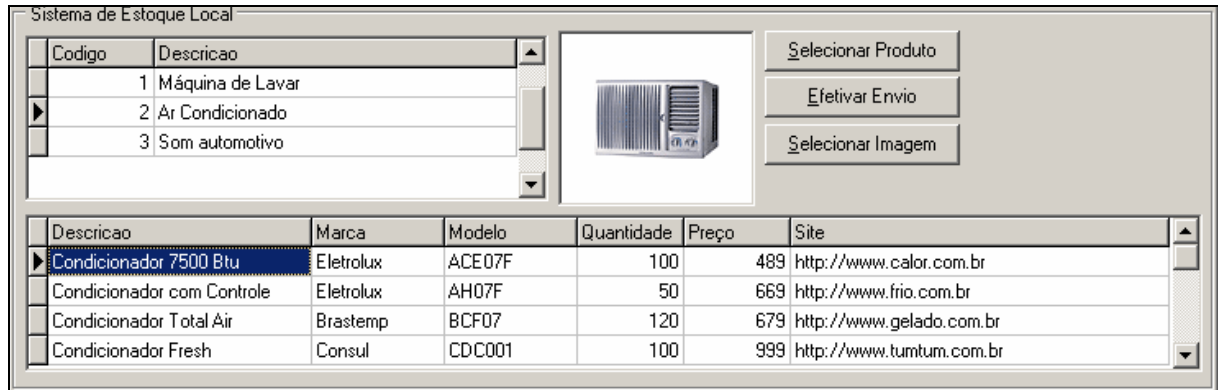
Área: Eletrodomesticos

Código	Solicitante	Seção	Produto	P. Inicial	P. Final	Prazo	Descrição
1	Garfield Eletro Shop	Condicionadores de Ar	Condicionador d...	500.00	700.00	21/06/2002	Especiais para promocao
2	Garfield Eletro Shop	Maquinas de Lavar ...	Lavad. Normais ...	300.00	600.00	28/06/2002	Maquinas Populares

Clicando em “Selecionar”, o primeiro item é selecionado e a aplicação de envio de catálogos está pronta para enviar os catálogos para a solicitação selecionada. É então exibido

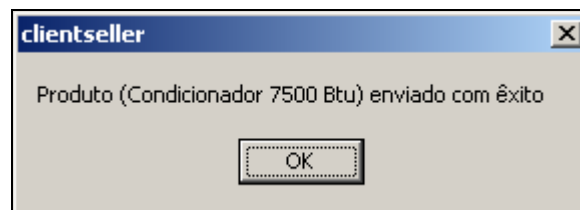
um esboço de um estoque da aplicação local, que contém os produtos que serão enviados. A figura 29 mostra o esboço de estoque dos produtos que serão enviados.

Figura 29 – ESBOÇO DE ESTOQUE LOCAL



Selecionando então o item basta clicar em “Efetivar Envio”. Se o processo for executado com sucesso será exibida uma mensagem, conforme a figura 30.

Figura 30 – SUCESSO NO ENVIO DE CATÁLOGO



6 CONSIDERAÇÕES FINAIS

Este capítulo irá apresentar as conclusões com relação ao trabalho desenvolvido, além das principais dificuldades encontradas e sugestões para continuações e trabalhos relacionados.

6.1 CONCLUSÕES

O objetivo principal deste trabalho, que é criar um protótipo de aplicação para troca de dados entre aplicações de comércio eletrônico utilizando o protocolo SOAP foi alcançado. Os ambientes *Borland Delphi 6* e *Microsoft Visual Studio .Net* mostraram-se adequados ao desenvolvimento de softwares utilizando as tecnologias SOAP e *Web services*.

O protótipo implementado permite que seus objetos sejam acessados por aplicações cliente que utilizem o protocolo SOAP. Essas aplicações podem ser desenvolvidas em qualquer linguagem e plataforma que implemente o protocolo SOAP. Isto pôde ser comprovado desenvolvendo as aplicações cliente de comércio eletrônico em ambientes de desenvolvimento diferentes.

O protocolo SOAP tem diversas vantagens sobre outras maneiras de chamar funções remotamente como DCOM ou CORBA, é simples de implementar, testar e usar, é um padrão da indústria, criado por um consórcio entre várias empresas, que utiliza em boa parte os padrões da *Web*: a comunicação é feita via HTTP através de documentos XML. O XML descreve perfeitamente os dados em tempo de execução e evita problemas causados por inadvertidas mudanças nas funções, já que os objetos chamados têm a possibilidade de sempre validar os argumentos das funções, tornando o protocolo muito robusto. O SOAP define também um padrão chamado WSDL, que descreve perfeitamente os objetos e métodos disponíveis, através de páginas XML acessíveis através da *Web*.

Utilizando a UML, com o apoio da ferramenta *Poseidon*, foi possível ter uma visão geral da aplicação principal antes de iniciar o processo de implementação. Esta visão foi primordial para o desenvolvimento de grande parte deste trabalho.

A utilização do ambiente *Visual Studio .Net* permitiu ainda o estudo e uso da linguagem C# que se mostrou muito eficiente para definir de forma clara os processamentos das aplicações cliente e acessar objetos remotos utilizando o protocolo SOAP.

6.2 DIFICULDADES ENCONTRADAS

A principal dificuldade foi validar a aplicação principal que publica os objetos para serem acessados via SOAP. O processo de implementação correu conforme o previsto no cronograma, porém alguns *bugs* presentes no ambiente *Borland Delphi 6* só foram resolvidos depois da atualização com o *service pack 2*. Embora esses *bugs* impossibilitassem a validação do implemento o problema foi resolvido com a atualização descrita anteriormente no começo da base de implementação.

Outra dificuldade foi a demora inicial para a obtenção dos livros relacionados ao assunto pois estes eram somente comercializados nos EUA e tiveram que ser importados pela Biblioteca Central da FURB.

6.3 EXTENSÕES

Existem muitas possibilidades de se desenvolver trabalhos relacionados ao protocolo SOAP. Este trabalho procurou utilizar as funcionalidades de chamada de procedimentos remotos para possibilitar a criação de um sistema de troca de dados.

Pode-se estudar o protocolo SOAP e fazer um comparativo com outras tecnologias semelhantes, como o DCOM e CORBA, ou estudar a implementação do protocolo SOAP sobre mecanismos de transportes diferentes de HTTP, como FTP ou SMTP.

Pode-se ainda estudar as várias linguagens que implementam recursos para acessar aplicações SOAP e estabelecer um estudo comparativo entre essas linguagens.

ANEXO 1 – CÓDIGO DO OBJETO DE ACESSO A INTERFACE ISERVICOPRODUTO

```

//-----
// <autogenerated>
//   This code was generated by a tool.
//   Runtime Version: 1.0.3705.0
//
//   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 Microsoft.VSDesigner, Version 1.0.3705.0.
//
namespace ManutencaoProduto.localhost1 {
    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="IServicoProdutobinding",
    Namespace="http://tempuri.org/")]
    public class IServicoProdutoservice :
    System.Web.Services.Protocols.SoapHttpClientProtocol {

        /// <remarks/>
        public IServicoProdutoservice() {
            this.Url = "http://localhost/catalogexchange/ce.exe/soap/IServicoProduto";
        }

        /// <remarks/>
        [System.Web.Services.Protocols.SoapRpcMethodAttribute("urn:UServiceProduto-
    IServicoProduto#InsereProduto", RequestNamespace="urn:UServiceProduto-IServicoProduto",
    ResponseNamespace="urn:UServiceProduto-IServicoProduto")]
        public void InsereProduto(int _Codigo_Secao, int _Codigo, string _Descricao) {
            this.Invoke("InsereProduto", new object[] {
                _Codigo_Secao,
                _Codigo,
                _Descricao});
        }

        /// <remarks/>
        public System.IAsyncResult BeginInsereProduto(int _Codigo_Secao, int _Codigo, string
    _Descricao, System.AsyncCallback callback, object asyncState) {
            return this.BeginInvoke("InsereProduto", new object[] {
                _Codigo_Secao,
                _Codigo,
                _Descricao}, callback, asyncState);
        }

        /// <remarks/>
        public void EndInsereProduto(System.IAsyncResult asyncResult) {
            this.EndInvoke(asyncResult);
        }

        /// <remarks/>
        [System.Web.Services.Protocols.SoapRpcMethodAttribute("urn:UServiceProduto-
    IServicoProduto#AtualizaProduto", RequestNamespace="urn:UServiceProduto-IServicoProduto",
    ResponseNamespace="urn:UServiceProduto-IServicoProduto")]
    
```

```

public void AtualizaProduto(int _Codigo_Secao, int _Codigo, string _Descricao) {
    this.Invoke("AtualizaProduto", new object[] {
        _Codigo_Secao,
        _Codigo,
        _Descricao});
}

/// <remarks/>
public System.IAsyncResult BeginAtualizaProduto(int _Codigo_Secao, int _Codigo,
string _Descricao, System.AsyncCallback callback, object asyncState) {
    return this.BeginInvoke("AtualizaProduto", new object[] {
        _Codigo_Secao,
        _Codigo,
        _Descricao}, callback, asyncState);
}

/// <remarks/>
public void EndAtualizaProduto(System.IAsyncResult asyncResult) {
    this.EndInvoke(asyncResult);
}

/// <remarks/>
[System.Web.Services.Protocols.SoapRpcMethodAttribute("urn:UServiceProduto-
IServicoProduto#DeletaProduto", RequestNamespace="urn:UServiceProduto-IServicoProduto",
ResponseNamespace="urn:UServiceProduto-IServicoProduto")]
public void DeletaProduto(int _Codigo_Secao, int _Codigo) {
    this.Invoke("DeletaProduto", new object[] {
        _Codigo_Secao,
        _Codigo});
}

/// <remarks/>
public System.IAsyncResult BeginDeletaProduto(int _Codigo_Secao, int _Codigo,
System.AsyncCallback callback, object asyncState) {
    return this.BeginInvoke("DeletaProduto", new object[] {
        _Codigo_Secao,
        _Codigo}, callback, asyncState);
}

/// <remarks/>
public void EndDeletaProduto(System.IAsyncResult asyncResult) {
    this.EndInvoke(asyncResult);
}

/// <remarks/>
[System.Web.Services.Protocols.SoapRpcMethodAttribute("urn:UServiceProduto-
IServicoProduto#ConsultaProduto", RequestNamespace="urn:UServiceProduto-IServicoProduto",
ResponseNamespace="urn:UServiceProduto-IServicoProduto")]
[return: System.Xml.Serialization.SoapElementAttribute("return")]
public bool ConsultaProduto(int _Codigo_Secao, int _Codigo, ref string Descricao_) {
    object[] results = this.Invoke("ConsultaProduto", new object[] {
        _Codigo_Secao,
        _Codigo,
        Descricao_});
    Descricao_ = ((string)(results[1]));
    return ((bool)(results[0]));
}

/// <remarks/>
public System.IAsyncResult BeginConsultaProduto(int _Codigo_Secao, int _Codigo,
string Descricao_, System.AsyncCallback callback, object asyncState) {
    return this.BeginInvoke("ConsultaProduto", new object[] {
        _Codigo_Secao,
        _Codigo,
        Descricao_}, callback, asyncState);
}

/// <remarks/>
public bool EndConsultaProduto(System.IAsyncResult asyncResult, out string
Descricao_) {
    object[] results = this.EndInvoke(asyncResult);
    Descricao_ = ((string)(results[1]));
    return ((bool)(results[0]));
}
}

```

```

    /// <remarks/>
    [System.Web.Services.Protocols.SoapRpcMethodAttribute("urn:UServiceProduto-
IServicoProduto#ContaRegistros", RequestNamespace="urn:UServiceProduto-IServicoProduto",
ResponseNamespace="urn:UServiceProduto-IServicoProduto")]
    [return: System.Xml.Serialization.SoapElementAttribute("return")]
    public int ContaRegistros() {
        object[] results = this.Invoke("ContaRegistros", new object[0]);
        return ((int)(results[0]));
    }

    /// <remarks/>
    public System.IAsyncResult BeginContaRegistros(System.AsyncCallback callback, object
asyncState) {
        return this.BeginInvoke("ContaRegistros", new object[0], callback, asyncState);
    }

    /// <remarks/>
    public int EndContaRegistros(System.IAsyncResult asyncResult) {
        object[] results = this.EndInvoke(asyncResult);
        return ((int)(results[0]));
    }

    /// <remarks/>
    [System.Web.Services.Protocols.SoapRpcMethodAttribute("urn:UServiceProduto-
IServicoProduto#SelecionaRegistro", RequestNamespace="urn:UServiceProduto-IServicoProduto",
ResponseNamespace="urn:UServiceProduto-IServicoProduto")]
    [return: System.Xml.Serialization.SoapElementAttribute("return")]
    public bool SelecionaRegistro(int BookMark, ref int Codigo_Secao_, ref int
Codigo_Produto_, ref string Descricao_) {
        object[] results = this.Invoke("SelecionaRegistro", new object[] {
            BookMark,
            Codigo_Secao_,
            Codigo_Produto_,
            Descricao_});
        Codigo_Secao_ = ((int)(results[1]));
        Codigo_Produto_ = ((int)(results[2]));
        Descricao_ = ((string)(results[3]));
        return ((bool)(results[0]));
    }

    /// <remarks/>
    public System.IAsyncResult BeginSelecionaRegistro(int BookMark, int Codigo_Secao_,
int Codigo_Produto_, string Descricao_, System.AsyncCallback callback, object asyncState) {
        return this.BeginInvoke("SelecionaRegistro", new object[] {
            BookMark,
            Codigo_Secao_,
            Codigo_Produto_,
            Descricao_}, callback, asyncState);
    }

    /// <remarks/>
    public bool EndSelecionaRegistro(System.IAsyncResult asyncResult, out int
Codigo_Secao_, out int Codigo_Produto_, out string Descricao_) {
        object[] results = this.EndInvoke(asyncResult);
        Codigo_Secao_ = ((int)(results[1]));
        Codigo_Produto_ = ((int)(results[2]));
        Descricao_ = ((string)(results[3]));
        return ((bool)(results[0]));
    }
}
}
}

```

ANEXO 2– UNIT PARA ACESSO A INTERFACE

```
// ***** //
// The types declared in this file were generated from data read from the
// WSDL File described below:
// WSDL      : http://localhost/catalogexchange/ce.exe/wsdl/IServicoEntradaCatalogo
// Version   : 1.0
// (17/06/2002 15:42:17 - $Revision: 1.9.1.0.1.0.1.9 $)
// ***** //

unit IServicoEntradaCatalogoCaller;

interface

uses InvokeRegistry, Types, XSBuiltIns;

type

// ***** //
// The following types, referred to in the WSDL document are not being represented
// in this file. They are either aliases[@] of other types represented or were referred
// to but never[!] declared in the document. The types from the latter category
// typically map to predefined/known XML or Borland types; however, they could also
// indicate incorrect WSDL documents that failed to declare or import a schema type.
// ***** //
// !:unsignedByte   - "http://www.w3.org/2001/XMLSchema"
// !:int             - "http://www.w3.org/2001/XMLSchema"
// !:string          - "http://www.w3.org/2001/XMLSchema"
// !:double          - "http://www.w3.org/2001/XMLSchema"
// !:boolean         - "http://www.w3.org/2001/XMLSchema"

TImagem      = array of Byte;           { "urn:UServiceEntradaCatalogo" }

// ***** //
// Namespace : urn:UServiceEntradaCatalogo-IServicoEntradaCatalogo
// soapAction: urn:UServiceEntradaCatalogo-IServicoEntradaCatalogo%#operationName%
// transport  : http://schemas.xmlsoap.org/soap/http
// style      : rpc
// binding    : IServicoEntradaCatalogobinding
// service    : IServicoEntradaCatalogoservice
// port       : IServicoEntradaCatalogoPort
// URL        : http://localhost/catalogexchange/ce.exe/soap/IServicoEntradaCatalogo
// ***** //
IServicoEntradaCatalogo = interface(IInvokable)
['{DA0548D2-F5B6-99AD-D5DA-97DFAEB66075}']
    procedure InsereEntradaCatalogo(const _Empresa: Integer; const _Solicitacao: Integer;
const _Solicitante: Integer; const _Quantidade: Integer; const _Marca: String; const
_Modelo: String; const _Descricao: String; const _Site: String; const _Preco: Double; const
_Imagem: TImagem); stdcall;
    procedure AtualizaEntradaCatalogo(const _Empresa: Integer; const _Solicitacao: Integer;
const _Solicitante: Integer; const _Quantidade: Integer; const _Marca: String; const
_Modelo: String; const _Descricao: String; const _Site: String; const _Preco: Double; const
_Imagem: TImagem); stdcall;
    procedure DeletaEntradaCatalogo(const _Empresa: Integer; const _Solicitacao: Integer;
const _Solicitante: Integer); stdcall;
    function ConsultaEntradaCatalogo(const _Empresa: Integer; const _Solicitacao: Integer;
const _Solicitante: Integer; var Quantidade_: Integer; var Marca_: String; var Modelo_:
String; var Descricao_: String; var Site_: String; var Preco_: Double; var Imagem_: String):
Boolean; stdcall;
    function ContaRegistros: Integer; stdcall;
    function SelecionaRegistro(const BookMark: Integer; var Empresa_: Integer; var
Solicitacao_: Integer; var Solicitante_: Integer; var Quantidade_: Integer; var Marca_:
String; var Modelo_: String; var Descricao_: String; var Site_: String; var Preco_: Double;
var Imagem_: String): Boolean; stdcall;
end;

function GetIServicoEntradaCatalogo(UseWSDL: Boolean=System.False; Addr: string=''):
IServicoEntradaCatalogo;
```

```

implementation
  uses SOAPHTTPClient;

function GetIServicoEntradaCatalogo(UseWSDL: Boolean; Addr: string):
IServicoEntradaCatalogo;
const
  defWSDL = 'http://localhost/catalogexchange/ce.exe/wsdl/IServicoEntradaCatalogo';
  defURL = 'http://localhost/catalogexchange/ce.exe/soap/IServicoEntradaCatalogo';
  defSvc = 'IServicoEntradaCatalogoservice';
  defPrt = 'IServicoEntradaCatalogoPort';
var
  RIO: THTTTPRIO;
begin
  Result := nil;
  if (Addr = '') then
  begin
    if UseWSDL then
      Addr := defWSDL
    else
      Addr := defURL;
  end;
  RIO := THTTTPRIO.Create(nil);
  try
    if UseWSDL then
    begin
      RIO.WSDLLocation := Addr;
      RIO.Service := defSvc;
      RIO.Port := defPrt;
    end else
      RIO.URL := Addr;
    Result := (RIO as IServicoEntradaCatalogo);
  finally
    if Result = nil then
      RIO.Free;
  end;
end;

initialization
  InvRegistry.RegisterInterface(TypeInfo(IServicoEntradaCatalogo),
'urn:UserServiceEntradaCatalogo-IServicoEntradaCatalogo', '');
  InvRegistry.RegisterDefaultSOAPAction(TypeInfo(IServicoEntradaCatalogo),
'urn:UserServiceEntradaCatalogo-IServicoEntradaCatalogo#%operationName%');
  RemClassRegistry.RegisterXSInfo(TypeInfo(TImagem), 'urn:UserServiceEntradaCatalogo',
'TImagem');

end.

```

REFERÊNCIAS BIBLIOGRÁFICAS

ABITEBOUL, Serge; BUNEMAN, Peter; SUCIU, Dan. **Gerenciando dados na WEB**. Rio de Janeiro: Campus, 2000.

ALBERTIN, Alberto Luiz. **Comércio eletrônico**. São Paulo: Atlas, 2000.

BIRON, Paul V; MALHOTRA, Ashok. **XML schema part 2: datatypes**, maio 2001. Disponível em < <http://www.w3c.org/TR/xmlschema-2/> >. Acesso em: 11 mai 2002.

BOGER, Marko et al. **Poseidon for UML users guide**. [s.l], maio 2001. Disponível em: < <http://www.gentleware.com/products/documentation/PoseidonUsersGuide.html> >. Acesso em: 10 dez 2001.

BROSE, Gerald; VOGEL, Andreas; DUDDY, Keith. **Java programming with corba**. Advanced techniques for building distributed applications. New York: John Wiley & Sons, 2001.

CANTU, Marco. **Dominando o delphi 6**: a bíblia. São Paulo: Makron Books, 2001.

CEPONKUS, Alex; HOODBHOY, Faraz. **Applied XML**: a toolkit for programmers. New York: John Wiley & Sons, 1999.

FIELDS, Duane; KOLB, Mark. **Desenvolvendo na web com java server pages**. Rio de Janeiro: Ciência Moderna, 2000.

HAHN, Harley; Stout, Rick. **Dominando a internet**. São Paulo: Makron Books, 1995.

KALAKOTA, Ravi; ROBINSION, Marcia. **E-business**: estratégias para alcançar o sucesso no mundo digital. Porto Alegre: Bookman, 2002.

LARMAN, Craig. **Utilizando UML e padrões**. Porto Alegre: Bookman, 2000.

LIGHT, Richard. **Iniciando em XML**. São Paulo: Makron Books, 1999.

LEVENTHAL, Michael; LEWIS, David; FUCHS, Matthew. **Designing XML internet applications**. Upper Saddle River: Prentice Hall PTR, 1998.

MARCHAL, Benoit. **XML conceitos e aplicações**. São Paulo: Berkeley, 2000.

MORGADO, Flavio. **Programando com Access**. Rio de Janeiro: IBPI, 1996.

MOULTIS, Natanaya P. **XML black book, solução e poder**. São Paulo: Makron Books, 2000.

SCRIBNER, Kenn; STIVER, Mark C. **Applied SOAP: implementing .NET XML web services**. Indianapolis: Sams, 2001.

SKONNARD, Aaron; GUDGIN, Martin. **Essential XML quick reference: a programmer's XML reference, XPath, XSLT, XML Schema, SOAP and more**. Indianápolis: Perason Education, 2002.

SNELL, James; TIDWELL, Doug; KULCHENKO, Kulchenko. **Programming web services with soap**. Santa Clara: O'Reilly, 2001.

SEELY, Scott. **SOAP: Cross plataform web service development using XML**. Upper Saddle River: Prentice Hall PTR, 2002.

TOMAZ, Fabio. **Protótipo de uma linguagem de programação de script para elaboração de conteúdo dinâmico na WWW**. 2001. 80 f. Monografia (Conclusão de Curso de Bacharel em Ciências da Computação). Universidade Regional de Blumenau, Blumenau.

W3C WORLD WIDE WEB CONSORTIUM. **Extensible markup language (XML) 1.0**. Estados Unidos, outubro 2000a. Disponível em <<http://www.w3.org/XML>>. Acesso em: 20 fev. 2002.

W3C WORLD WIDE WEB CONSORTIUM. **Simple object access protocol (SOAP)**. Estados Unidos, maio 2000b. Disponível em <<http://www.w3.org/XML>>. Acesso em: 20 fev. 2002.

W3C WORLD WIDE WEB CONSORTIUM. **Web Services Architecture Requirements.** Estados Unidos, abril 2002. Disponível em <<http://www.w3c.org/TR/wsa-reqs>>. Acesso em: 30 mar. 2002.