UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
COORDENAÇÃO DE INFORMÁTICA
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

CLAUDINEI RODRIGUES JUNIOR

UM *WEB SERVICE* PARA BUSCA DE PREÇOS DE PRODUTO

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA
2010

CLAUDINEI RODRIGUES JUNIOR

UM *WEB SERVICE* PARA BUSCA DE PREÇOS DE PRODUTO

Trabalho de conclusão de curso de graduação, apresentado à disciplina Trabalho de Diplomação, do curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas da Coordenação de Informática – COINF – da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para a obtenção do título de Tecnólogo.

Orientadora: Profª. Drª. Simone Nasser Matos

Co-Orientador: Prof. Dr. João Henrique Kleinschmidt

PONTA GROSSA
2010

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS PONTA GROSSA
GERÊNCIA DE ENSINO E PESQUISA
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
DISCIPLINA DE TRABALHO DE DIPLOMAÇÃO

TERMO DE APROVAÇÃO

UM *WEB SERVICE* PARA BUSCA DE PREÇOS DE PRODUTO

Claudinei Rodrigues Junior

Este Trabalho de Diplomação foi considerado adequado como cumprimento das exigências legais do currículo do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas e aprovado em sua forma final pela Coordenação de Informática da Universidade Tecnológica Federal do Paraná – Campus Ponta Grossa.

Profª. SIMONE NASSER MATOS
Orientadora

Profª. HELYANE BRONOSKI BORGES
Responsável pelo Trabalho de Diplomação

Prof. ANDRÉ KOSCIANSKI
Coordenador do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

Banca Examinadora:

Prof. LOURIVAL APARECIDO GÓIS

Prof. JOÃO UMBERTO FURQUIM DE SOUZA

AGRADECIMENTOS

Agradeço à minha família pela força e apoio: minha mãe, Elza, minha irmã, Daysiane e meu cunhado, Fábio.

Agradeço à professora Simone Nasser Matos pela dedicação, compreensão com meus erros e imensurável paciência ao longo deste trabalho. Agradeço também ao professor João Henrique Kleinschmidt que se dispôs a co-orientar este trabalho.

Agradeço a todos aqueles que, de uma forma ou outra, me apoiaram no desenvolvimento deste trabalho e/ou ao longo do curso.

Agradeço em especial a meu pai, Claudinei Rodrigues, que, infelizmente, faleceu em 2008, não podendo ver a realização deste trabalho e o término do curso.

RESUMO

RODRIGUES, Claudinei J. **Um *web service* para busca de preços de produto.** 2010, 69f. Trabalho de Conclusão de Curso – Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2010.

Atualmente não há no mercado um *software* que implemente diversos métodos de formação de preço de venda e que seja gratuito. Conscientes deste problema, o Grupo de Pesquisa em Engenharia de *Software* (GPES) da Universidade Tecnológica Federal do Paraná (UTFPR), Campus Ponta Grossa, está desenvolvendo um *framework* que implemente diversos métodos de formação de preço de venda. Um dos métodos que será implementado, denominado método baseado nas decisões das empresas concorrentes, depende dos preços de venda praticados pelo mercado. Como não há um serviço gratuito e livre na *web* que permita a obtenção de preços de venda de um produto, este trabalho visa desenvolver um *web service* que busque os preços de vendas de produtos em sítios de *e-commerce* para auxiliar o GPES na implementação deste método. Além disso, o *web service* permite o uso automatizado por aplicações e é de código aberto.

Palavras-chave: *web service*; métodos de formação de preço de venda; engenharia de *software*.

ABSTRACT

RODRIGUES, Claudinei J. **A web service for product selling prices search.** 2010, 69f. Final Paper – Systems Analysis And Development Technology, Federal Technological University of Paraná. Ponta Grossa, 2010.

Currently there is not on the market a free software that implements several pricing methods. Due about this problem, the Software Engineering Research Group (GPES) from Federal Technological University of Paraná (UTFPR), Campus Ponta Grossa, has been developing a framework that implements several pricing methods. One of those methods, known as pricing method based on the bidder decisions, depends on pricing method used in market. There is not at the internet a free and open source service that searches selling prices of a product. Helping GPES to achieve its goals, this paper aims to develop a web service that searches on the internet selling prices of a product. Therefore, the web service developed is open source and allows automated use by applications.

Keywords: web service; pricing methods; software engineering.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

FTP	*File Transfer Protocol*
GPES	Grupo de Pesquisa em Engenharia de *Software*
HTML	*Hyper Text Markup Language*
HTTP	*Hyper Text Transfer Protocol*
HTTPR	*Hyper Text Transfer Protocol Reliable*
JAX-WS	Java API *for* XML - *Web Service*
JMS	Java *Message Service*
JSF	Java *Server Faces*
RMI	*Remote Method Invocation*
RPC	*Remote Procedure Call*
SMTP	*Simple Mail Transfer Protocol*
UDDI	*Universal Description, Discovery and Integration*
URL	*Uniform Resource Locator*
UTFPR	Universidade Tecnológica Federal do Paraná
WSDL	*Web Services Descriptor Language*
WSIT	*Web Service Interoperability Technology*
XML	*eXtensible Markup Language*
XSD	XML *Schema*

LISTA DE ACRÔNIMOS

BEEP	*Blocks Extensible Exchange Protocol*
CORBA	*Common Object Request Broker*
DCOM	*Distributed Component Object Model*
DIME	*Direct Internet Message Encapsulation*
POJO	*Plain Old Java Object*
SOA	*Service Oriented Architecture*
SOAP	*Simple Acess Object Protocol*
URI	*Uniform Resource Identifier*
WUST	WSDL, UDDI e SOAP *Technologies*

SUMÁRIO

1 INTRODUÇÃO

A competitividade do mercado é intensa. Definir preços competitivos que agradem aos consumidores e permitam às empresas obterem lucros é uma tarefa complexa. A sobrevivência de uma empresa está diretamente ligada aos preços praticados. Um preço de venda mal-definido pode comprometer desde seu crescimento até sua estabilidade sócio-econômica.

Ao longo dos anos, diversos estudiosos de administração, tais como Santos (2001) e Sardinha (2005), criaram métodos para o cálculo do preço de venda. Cada método utiliza um conjunto de variáveis distintas e é adequado para situações específicas.

Cabe aos gestores conhecer os métodos, aplicá-los, analisar os resultados e decidir pelo melhor preço. Entretanto, o mercado é agressivo. Há diversos outros fatores que um gestor deve considerar além do preço e nem sempre sobra tempo para realizar os cálculos.

Atualmente não há alternativa aos gestores senão realizar os cálculos manualmente, pois os *softwares* disponíveis no mercado implementam somente alguns métodos e, em sua maioria, são pagos.

Conscientes do problema enfrentado pelos gestores e da falta de *softwares* que os auxiliem a tomar a decisão pelo preço de venda adequado mais agilmente, o Grupo de Pesquisa em Engenharia de *Software* (GPES), da Universidade Tecnológica Federal do Paraná (UTFPR), vem desenvolvendo um *framework*, denominado *Framemk*, que implementa alguns métodos de formação de preço de venda. Este *framework* será gratuito e poderá ser utilizado por qualquer um que precise definir preços de vendas.

Este trabalho tem como objetivo colaborar com o projeto do GPES na implementação do método conhecido por “método baseado nas decisões das empresas concorrentes” (SARDINHA, 1995). Este método utiliza como parâmetros de entrada os preços de um produto praticados pelo mercado e o *web service* desenvolvido neste trabalho é uma solução para o GPES no que tange à obtenção dos preços de um dado produto, pois, apesar de existirem sítios que relacionam os preços de produto na internet, tais como BuscaPe (BUSCAPE, 2010), JaCotei (JACOTEI, 2010), entre outros, estes não disponibilizam nenhum mecanismo ou

meio de acesso para realizar as buscas de maneira automatizada e não são de códigos abertos para serem acoplados a novas aplicações. E não é legal, conforme foi constatado através de contato por email com as empresas responsáveis por estes sitos, desenvolver uma aplicação que obtenha e utilize os preços cadastrados.

1.1 OBJETIVOS

Os objetivos deste trabalho estão descritos a seguir.

1.1.1 Objetivo Geral

Desenvolver um *web service* que realize buscas de preços de um produto em sítios de *e-commerce*.

1.1.2 Objetivos Específicos

- Realizar um estudo sobre *web services.*
- Utilizar somente ferramentas gratuitas para desenvolver o *web service.*
- Criar uma interface administrativa para o *web service.*

1.2 PROBLEMA

O primeiro sítio de comparação de preços de venda de produtos que surgiu na internet foi o Streetprice (STREETPRICE, 2010), criado em 1997. Depois deste, vários sítios de comparação de preços sugiram, como BuscaPe, JaCotei e Bondfaro (BONDFARO, 2010).

Apesar de existirem pequenas variações nos modelos de funcionamento de sítios de comparação de preços, todos funcionam, basicamente, da mesma maneira:

são feitos acordos comerciais com lojistas e responsáveis por sítios de *e-commerce* para que seus produtos sejam relacionados nas buscas feitas no sítio.

Este modelo de funcionamento implica em dois problemas para o GPES:

- Somente são relacionados produtos de empresas e/ou lojas e/ou sítios de *e-commerce* que tenham feito um acordo comercial com o sítio de comparação de preços.
- Os resultados retornados pelos sítios não podem ser usados por outras aplicações devido aos acordos comerciais realizados, conforme apresentado no Anexo A.

Assim sendo, o GPES não pode utilizar os dados de sítios de comparação de preços para implementação do método baseado nas decisões das empresas concorrentes.

O *web service* desenvolvido neste trabalho visa ser uma solução para o GPES no que tange à obtenção dos preços de venda de um produto. Para isso, a aplicação irá buscar em sítios de *e-commerce* pelos preços praticados de um produto.

1.3 ORGANIZAÇÃO DO TRABALHO

Este trabalho está divido em cinco capítulos. O capítulo 2 apresenta a fundamentação teórica dos *web services* e sua arquitetura. O capítulo 3 descreve o desenvolvimento do serviço e apresenta a arquitetura do *web service* proposto. O capítulo 4 apresenta os resultados obtidos por este trabalho. E, por fim, o capítulo 5 conclui o trabalho e faz propostas para trabalhos futuros.

2 WEB SERVICES: UMA VISÃO GERAL

Este capítulo descreve as principais características de um *web service*. A seção 2.1 apresenta os motivos de seu surgimento. A seção 2.2 descreve as principais tecnologias envolvidas na construção desta tecnologia. A seção 2.3 explica o funcionamento de um *web service*. A seção 2.4 discorre sobre a arquitetura dos *web services* sob duas perspectivas. E, por fim, a seção 2.5 discute algumas das principais limitações impostas pelo modelo de *web services*.

2.1 SURGIMENTO

O uso da Internet resultou no surgimento de novas tecnologias e na evolução das já existentes. Estas tecnologias tornam transações através da Web mais seguras, rápidas e fáceis de serem realizadas pelos usuários, até mesmo pelos leigos.

A Internet permitiu que não somente diversos negócios possuam suas interfaces Web com os consumidores, mas também que surgissem negócios que funcionem somente na Internet, sem existir uma loja física, como é o caso do Amazon, Submarino, entre outros.

Embora o crescimento da Internet atenda aos modelos de negócios com usuários humanos, as transações de grande porte e que não dependem de interação humana, realizadas entre corporações, governos, empresas, etc., encontram dificuldades no ambiente Web devido à heterogeneidade das plataformas, diferenças entre sistemas operacionais, baixa escalabilidade de modelos de sistemas distribuídos amplamente aceitos, tais como: CORBA (*Common Object Request Broker*), RMI (*Remote Method Invocation*), DCOM (*Distributed Component Object Model*), entre outros (ABINADER; LINS, 2006).

Os Web Services foram concebidos para prover um modelo adequado para este tipo de transações realizadas entre aplicações e classificadas como *bussiness-to-bussiness* (B2B).

2.2 CONCEITOS

Um *Web Service* é qualquer serviço disponível na Internet que use um sistema de mensagens com XML (*eXtensible Markup Language*) padronizado e não esteja preso à linguagem de programação ou sistema operacional. Este tipo de serviço permite que duas aplicações se comuniquem através de uma interface bem definida (CERAMI, 2002).

É desejável, ainda que não indispensável, que os *Web Services* possam ser facilmente encontrados através de mecanismos simples de busca, sejam auto-descritivos e usem padrões comuns da Internet. Se estes aspectos forem atendidos, a integração automática de aplicações se torna possível.

Para que os serviços possam ser facilmente encontrados, há um diretório comum onde são armazenadas informações sobre os serviços disponíveis. Este diretório é chamado de UDDI (*Universal Description, Discovery and Integration*) (CERAMI, 2002).

As descrições de serviços disponíveis no UDDI são documentos no padrão WSDL (*Web Service Definition Language*). Independente de a descrição estar disponível no UDDI, este possui, além de outras informações, a URL que aponta para o local onde está armazenada a descrição completa do serviço.

Nas seções seguintes detalham-se as tecnologias empregadas, conhecidas como WUST (WSDL, UDDI e SOAP *Technologies*). Estas tecnologias unidas a XML compõem a estrutura básica dos *Web Services*.

2.2.1 XML

A *Extensible Markup Language (XML)* é o padrão adotado para o transporte de dados (ABINADER; LINS, 2006; CERAMI, 2002). Simples de ser compreendida, fácil de ser transportada e compatível com diversas plataformas, a XML é a responsável por grande parte da interoperabilidade entre aplicações de ambientes de heterogêneos permitida pelos *Web Services*.

A XML serve de base para as outras três tecnologias fundamentais dos *Web Services*: SOAP, UDDI e WSDL.

2.2.2 WSDL

A *Web Services Descriptor Language* (WSDL) é a linguagem utilizada para criar uma auto-descrição pública do serviço (ABINADER; LINS, 2006; CERAMI, 2002).Entre as informações que podem constar neste documento estão: o padrão utilizado para troca de mensagem (SOAP, JMS (*Java Message Service*, entre outros), protocolo de transporte utilizado (HTTP, HTTPR (*Hyper Text Transfer Protocol Reliable*), DIME (*Direct Internet Message Encapsulation*), entre outros) e o endereço lógico (URL) do serviço.

A WSDL é baseada na XML e o formato das descrições é definido e validado utilizando XML *Schema* (XSD).

A possibilidade de criar descrições dos serviços e, com isso, facilitar a integração automatizada de aplicações, consiste em uma das principais vantagens dos *Web Services* sobre os outros modelos distribuídos.

2.2.3 UDDI

O *Universal Description Discovery and Integration* (UDDI) é o diretório onde as informações sobre os *Web Services* podem ser armazenadas (ABINADER; LINS, 2006; CERAMI, 2002). O diretório contém mecanismos que permitem a descoberta (procura) de serviços.

Uma das principais vantagens da utilização do UDDI é a garantia de que todos os *Web Services* disponíveis que atendam aos requisitos exigidos pelo solicitante tenham sido considerados durante o processo de escolha.

Um provedor de serviços pode armazenar três tipos de informações em um registro UDDI. Estas informações são classificadas da seguinte maneira:

- Páginas Brancas: informações sobre a empresa, incluindo nome do negócio, endereço e regras de tributação. A principal função das páginas brancas é viabilizar a descoberta de serviços a partir da identificação de cada empresa (CERAMI, 2002).

- Páginas Amarelas: os *Web Services* são descritos e agrupados utilizando categorias. Estas informações permitem a descoberta de serviços baseados na categoria do serviço.
- Páginas Verdes: fornecem informações técnicas que descrevem o comportamento do serviço e as funções suportadas. Entre estas informações está a URL que aponta para o documento WSDL.

2.2.4 SOAP

O *Single Object Acess Protocol* (SOAP) é o protocolo adotado pela maioria dos *Web Services* para troca de mensagens (ABINADER; LINS, 2006; CERAMI, 2002). Baseado em XML, o SOAP é projetado para que possa ser utilizado sobre qualquer protocolo, incluindo SMTP (*Simple Mail Transfer Protocol*), HTTP (*Hyper Text Transfer Protocol*) e FTP (*File Transfer Protocol*).

O funcionamento do SOAP é similar ao do RPC (*Remote Procedure Call*) em que: a mensagem enviada pelo cliente deve informar qual método deverá ser invocado e os valores de seus parâmetros, se estes existirem. O serviço devolverá uma mensagem contendo o valor retornado pelo método. Esta abordagem deve ser cuidadosamente projetada para causar o menor acoplamento possível entre as aplicações, pois o servidor irá esperar parâmetros de tipos de dados específicos e um formato de mensagem rigorosamente definido. Se a linguagem ou plataforma do solicitante não possuir o tipo de dado esperado ou for incapaz de gerar a mensagem no formato requerido, a comunicação estará comprometida.

Uma alternativa ao modelo de comunicação baseado em RPC é o modelo fundamentado em troca de mensagens. Neste modelo, o servidor espera um documento inteiro para processar e não uma mensagem contendo uma sequência rigorosa de parâmetros. Se o modelo for implementado de maneira síncrona, o solicitante irá esperar algum retorno do serviço, seja somente uma confirmação do recebimento do documento ou algum valor ou documento. Porém, se a implementação for feita de maneira assíncrona, o cliente não aguardará uma resposta, cabendo ao serviço decidir se algo deverá ser enviado ao cliente.

Em ambos os modelos de comunicação, o conteúdo da mensagem estará dentro de um envelope SOAP.

2.3 MODELO DE FUNCIONAMENTO

A figura 1 ilustra a criação, disponibilização, registro e utilização de um *Web Service*.

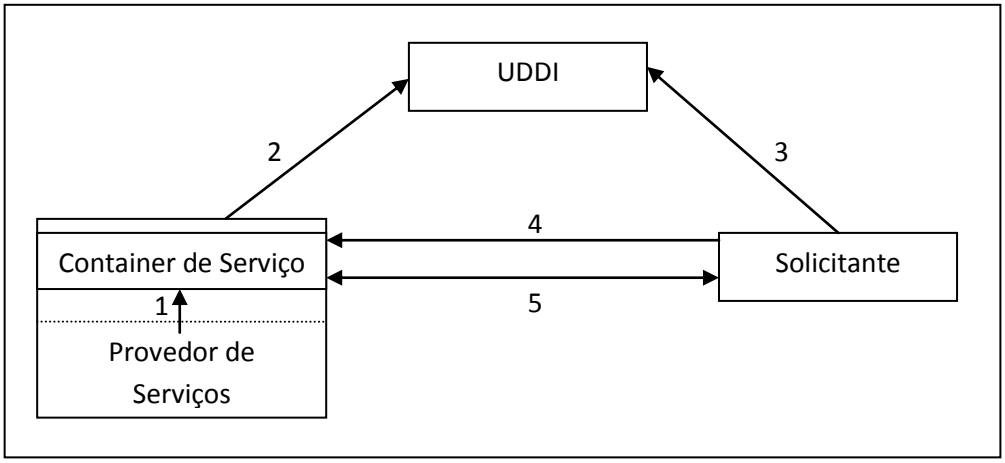

Figura 1 - Modelo de Funcionamento de um *Web Service*
Fonte: Autoria Própria.

O provedor de serviços cria e disponibiliza um *Web Service*, junto com a descrição do serviço (WSDL), em um *Container* de Serviço (1). Após, ele registra e publica o serviço em um diretório UDDI (2). Um solicitante busca no diretório (3) por um *Web Service* que satisfaça seus requisitos. Depois de encontrar, o cliente solicita ao provedor a descrição do serviço no padrão WSDL (4). Utilizando as informações obtidas, o solicitante estabelece conexão com o *Container* de Serviço para iniciar a troca de mensagens (5).

2.4 ARQUITETURA DOS WEB SERVICES

Esta seção descreve a arquitetura de *web services* sob duas perspectivas. A seção 2.4.1 é inteiramente baseada em Booth *et. al.* (2009) e analisa a arquitetura sob a ótica de conceitos e seus relacionamentos. A seção 2.4.2 discute a arquitetura sob a perspectiva dos *stakeholders*.

2.4.1 Perspectiva De Conceitos e Relacionamentos

Um conceito é aquilo que identifica algo que deve ser considerado na implementação da arquitetura (Booth *et. al.*, 2009). O relacionamento mostra como os conceitos se associam. Para facilitar a compreensão de como os conceitos se relacionam são utilizados mapas conceituais.

Para facilitar o estudo da arquitetura dos *Web Services*, a arquitetura foi divida em quatro modelos, que são subconjuntos de funções arquiteturais que se encarregam de um determinado aspecto.

A Figura 2 ilustra o modelo de *Web Services* dividindo-o em quatro modelos básicos.

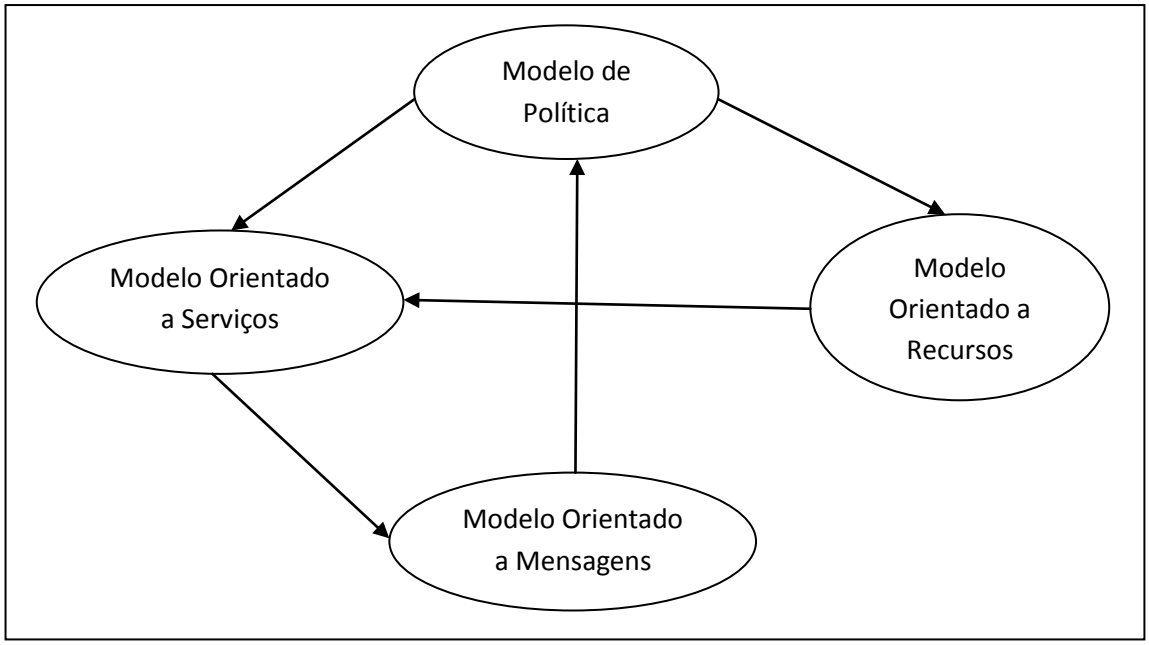

Figura 2 - Modelos Arquiteturais dos *Web Services*
Fonte: Autoria Própria.

2.4.1.1 Modelo orientado a mensagens

Este modelo foca a estrutura das mensagens, o relacionamento entre o receptor e o emissor da mensagem e como as mensagens são transmitidas. O funcionamento deste modelo está ilustrado na Figura 3.

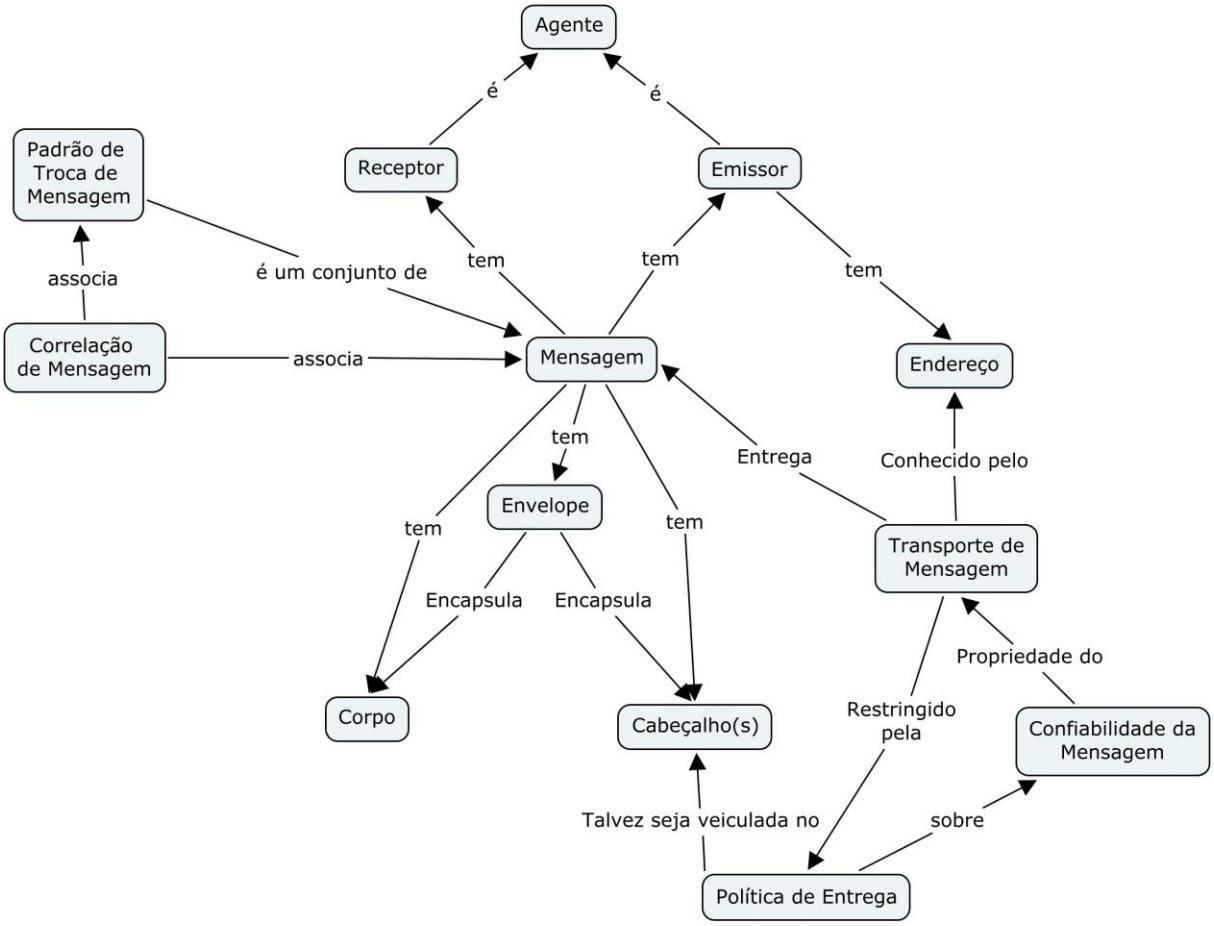

Figura 3 - Modelo Orientado a Mensagens
Fonte: Adaptado de Booth et. al., 2009

O endereço (*address)* é a informação usada para descrever como e onde a mensagem deve ser entregue. Não há um padrão para o endereçamento, sendo este dependente do mecanismo de transporte adotado. Por exemplo, se for adotado o HTTP, o endereço será a URL.

A política de entrega, *Delivery Policy*, define as regras para a entrega de mensagem, tais como: a garantia da qualidade de serviço, emprego de mecanismos de segurança, entre outras.

A mensagem (*message*) é a unidade de dados enviada do transmissor para o receptor e seu formato é definido na descrição do serviço. Cada mensagem possui um envelope que encapsula um ou mais cabeçalho (*header)* e somente um corpo (*body*).

Para que um receptor (*receiver*) de uma mensagem possa identificar a mensagem como sendo a resposta de uma determinada requisição realizada é necessário que haja uma maneira para que se possa estabelecer correlação das mensagens (*message correlation*). E para garantir que tanto o receptor quanto o

emissor (*sender*) usem os mesmos padrões para troca de mensagens, tratamento de falhas e correlação de mensagens é necessário que eles adotem um padrão de troca de mensagens (*message exchange pattern*).

2.4.1.2 Modelo orientado a serviços

O principal objeto do modelo orientado a serviços, ilustrado na Figura 4, é explicar o relacionamento entre o agente e os serviços que ele provê e as requisições recebidas.

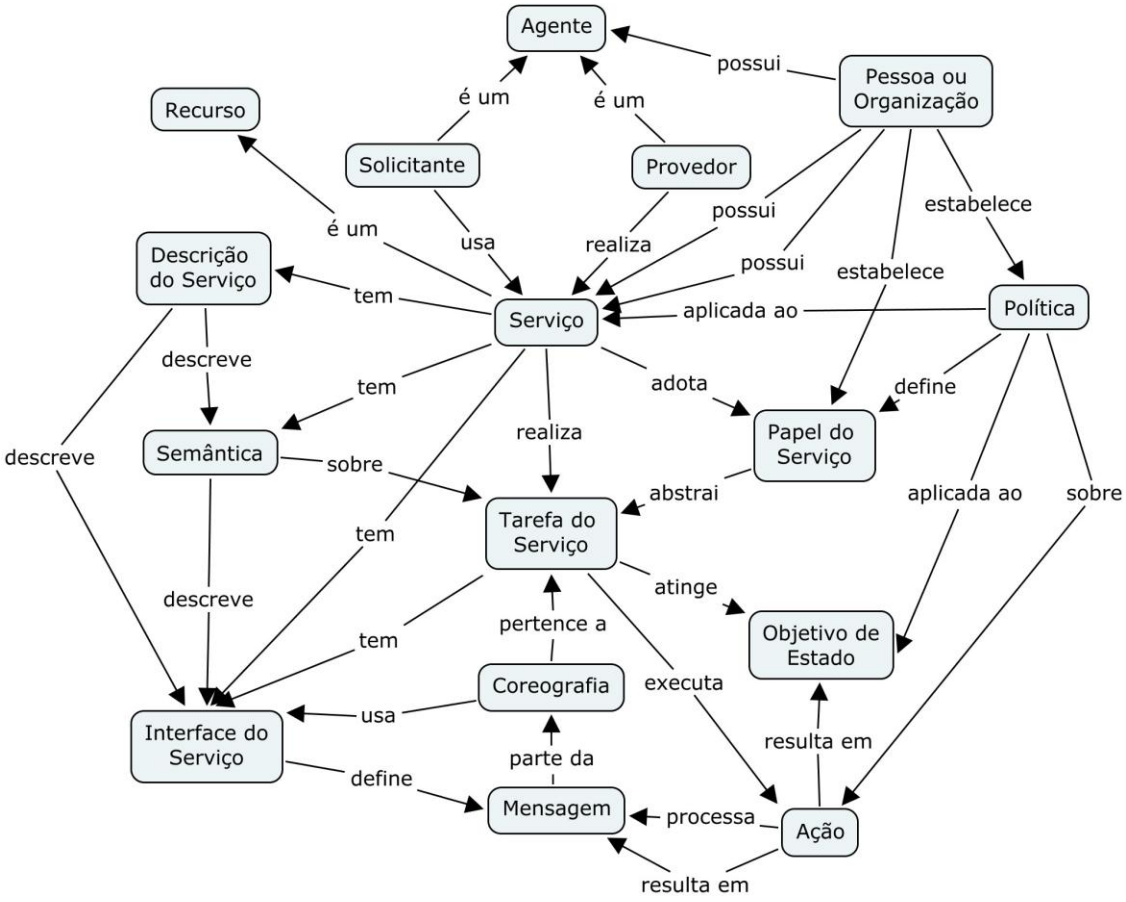

Figura 4 - Modelo Orientado a Serviços
Fonte: Adaptado de Booth et. al., 2009

Uma ação (*action*) é qualquer mudança observável de estado no agente (*agent*), podendo ser este ser um solicitante (*requester agent*) ou um provedor (*provider agent*).

A coreografia (*choreography*) define a sequência e as condições sob as quais diversos agentes irão atuar, sendo que cada um disponibiliza sua interface de

serviço (*service interface*) para realizar uma tarefa de serviço (*service task*) e atingir o estado de serviço desejado (*goal state*).

A diferença entre a coreografia e o padrão de troca de mensagens consiste em que a primeira se preocupa não somente com o padrão utilizado para troca de mensagens, mas também com as condições do serviço. O padrão para troca de mensagens se preocupa somente em como as mensagens são trocadas.

2.4.1.3 Modelo orientado a recursos

O modelo orientado a recursos foca nos recursos que são relevantes para a arquitetura de um *web services*. Considera-se que os web services em si são, também, recursos. Seu funcionamento está ilustrado na Figura 5.

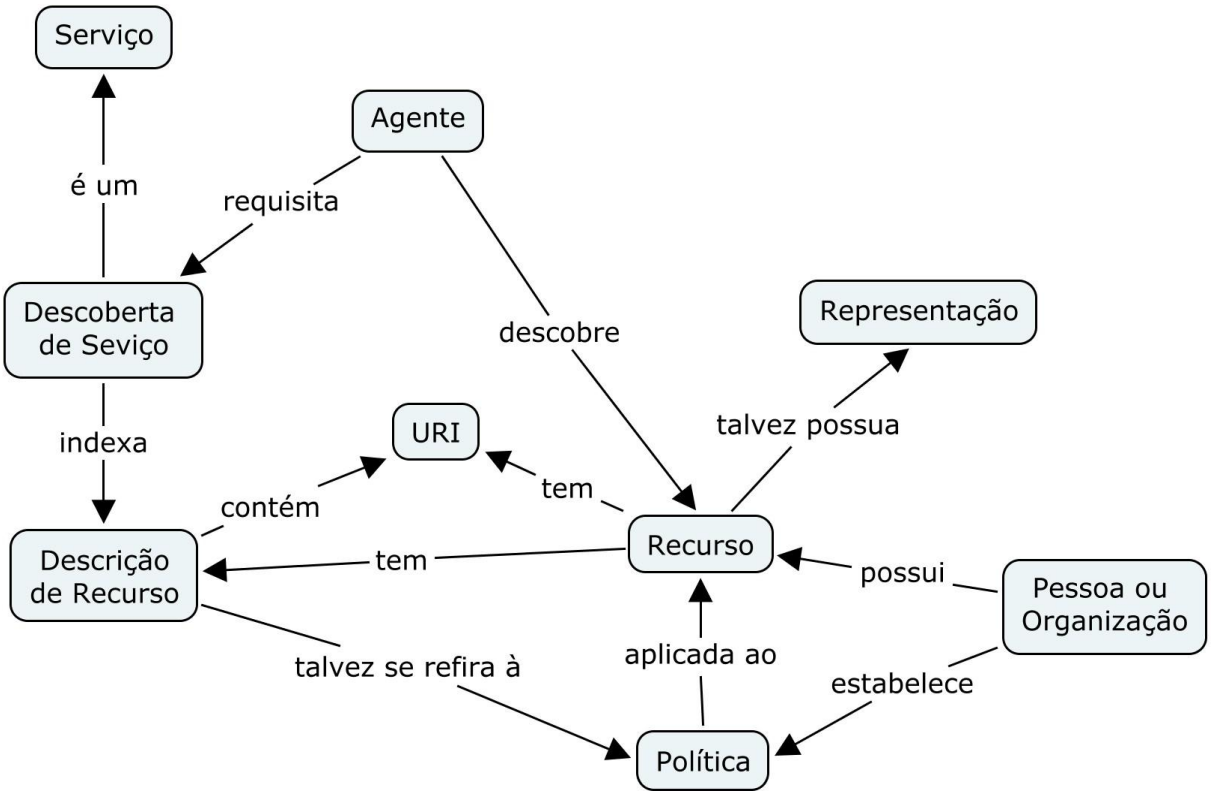

Figura 5 - Modelo Orientado a Recursos
Fonte: Adaptado de Booth et. al., 2009

Cada recurso possui um URI (*Uniform Resource Identifier*), ou seja, um identificador único, e uma descrição (*resource description*). Um agente pode usar a descoberta de serviço (*discovery service*) que é um mecanismo que indexa as

descrições de serviços para publicar e encontrar serviços baseados em critérios de funções ou semântica.

Um recurso pode ter uma representação (*representation*) que é um objeto que descreve o estado do recurso.

2.4.1.4 Modelo de política

O modelo de política, ilustrado na Figura 6, foca nos aspectos da arquitetura relacionados à política de serviço e, por consequência, à qualidade do serviço e à segurança.

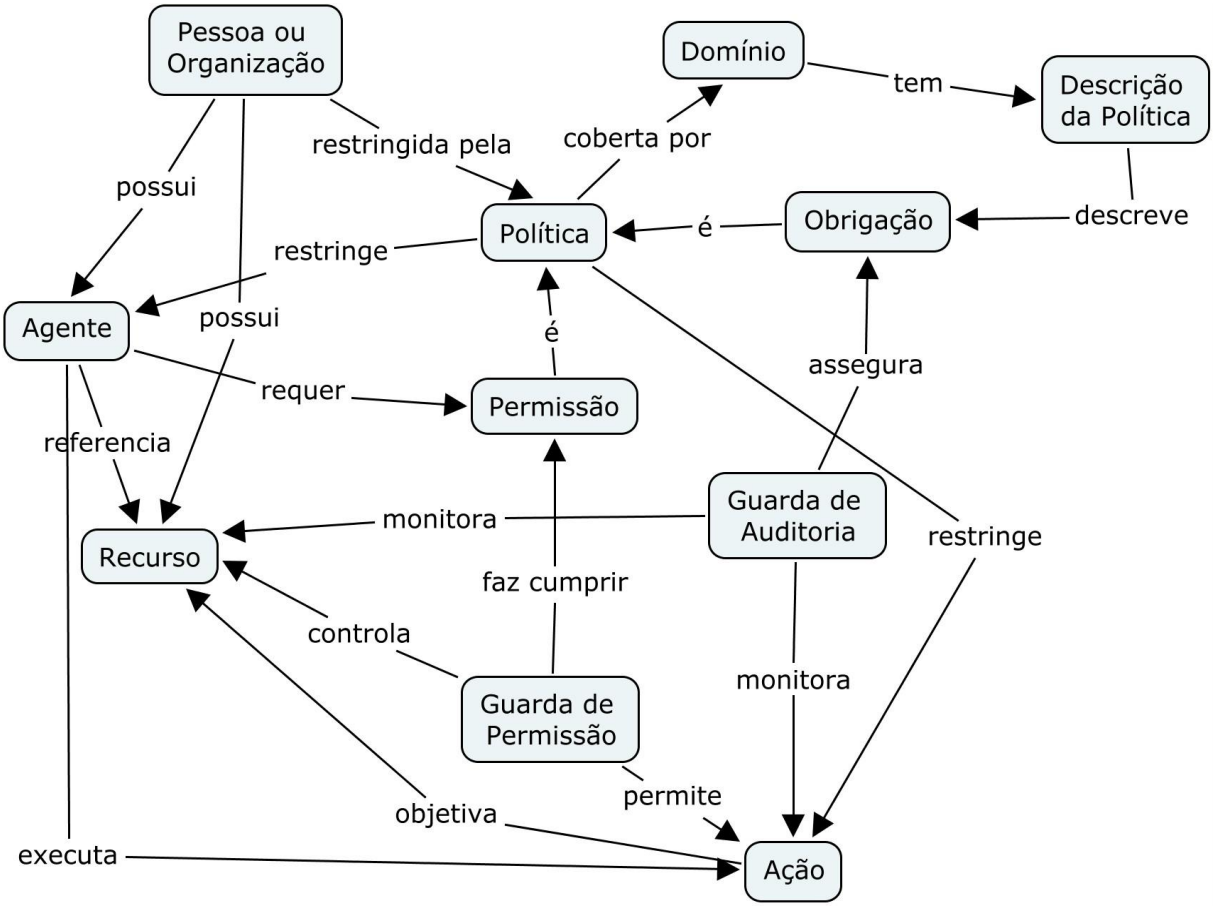

Figura 6 - Modelo de Política
Fonte: Adaptado de Booth et. al., 2009

O guarda de auditoria (*Audit guard*) é um mecanismo utilizado para monitorar as ações (*action*) e os recursos *(resource*) a fim de garantir o cumprimento das obrigações (*obligation*) do serviço.

A obrigação do serviço é uma política (*policy*) especificada na descrição de política (*Policy description*). O conjunto de recursos e agentes que são afetados por esta política é chamado de domínio (*domain*).

A permissão (*permission*) é um tipo de política que diz quais ações e estados de um agente ou recurso são permitidos. Para assegurar o cumprimento da permissão é implantado o guarda de permissão (*permission guard*).

2.4.2 Perspectiva dos *Stakeholders*

Esta seção irá analisar a arquitetura dos *web services* sob a perspectiva dos *stakeholders*.

2.4.2.1 Arquitetura orientada a serviço

A arquitetura orientada a serviço (*Service Oriented Architecture* – SOA) é um estilo arquitetural que promove a perda de acoplamento entre aplicações. Isso permite o uso de serviços através de uma rede e tais serviços podem ser consumidos por diversas aplicações em diferentes plataformas (MAHMOUD, 2010).

A SOA permite a descoberta automatizada de serviços, para isso é necessário que os serviços sejam descritos utilizando uma linguagem processável por máquinas e registrados em um local onde possam ser encontrados.

A figura 7 ilustra o esquema de funcionamento da SOA.

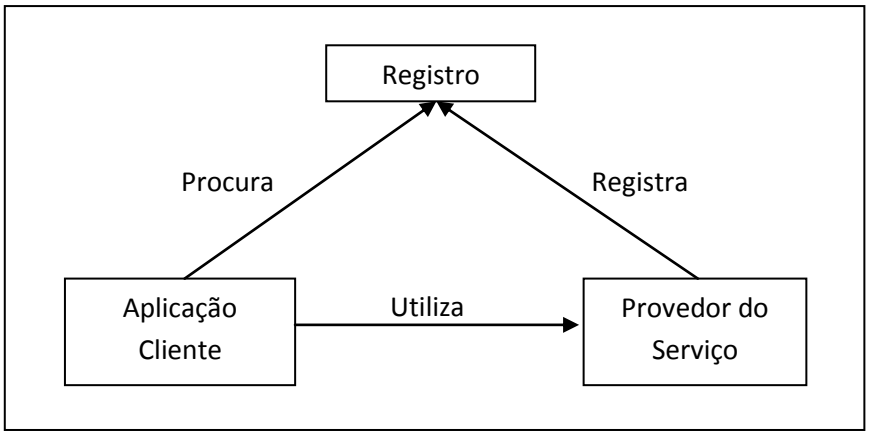

Figura 7 - Funcionamento da SOA
Fonte: Autoria Própria

Como é possível observar na figura 7, inicialmente o provedor do serviço registra as informações sobre seu serviço em um registro utilizando uma linguagem descritiva processável por máquinas. Em seguida, o cliente busca no registro um serviço que atenda às suas necessidades. Quando encontrar, o cliente obterá as informações necessárias para que ele possa utilizar o serviço (HANSEN, 2007).

Uma das maneiras de realizar a SOA é por meio de um *web services*. A linguagem descritiva processável por máquinas utilizada é a WSDL e o local onde os serviços podem ser registrados é o diretório UDDI. Um dos objetivos primordiais da SOA e, por conseqüência, dos *web services* é permitir a interoperabilidade entre aplicações heterogêneas. Para atingir este objetivo, as tecnologias empregas pelos *web services* são baseadas em XML.

2.5 LIMITAÇÕES DOS WEB SERVICES

Segundo Abinader e Lins (2006), as principais limitações impostas pelo modelo de *Web Services* são:

- O uso do HTTP não garante a entrega da mensagem e nem a sequência correta de entrega. Há protocolos alternativos, como HTTPR (*Hyper Text Transfer Protocol Reliable)*, BEEP (*Blocks Extensible Exchange Protocol*), DIME (*Direct Internet Message Encapsulation)*, entre outros, porém estes não são padrões adotados na Internet.
- Os mecanismos de segurança para *Web Services* não são padronizados. Cada fornecedor de *Container* cria sua própria solução e compromete a portabilidade entre fornecedores.
- A escalabilidade do serviço pode ser comprometida quando há dependência de sistemas legados, pois estes podem ser pouco escaláveis.
- O desempenho dos *Web Services* é gravemente comprometido pelo uso do SOAP, pois este transporta XML, que é grande se comparado a dados binários e a extração de dados do envelope SOAP, o que consome muito processamento. O *parsing* de arquivos XML também consome

processamento e, dependendo do padrão adotado para manipular os arquivos, utiliza muita memória.

- Não há um padrão para cobrança e precificação de uso do serviço, restando ao provedor utilizar um serviço intermediário para monitorar o uso do *Web Service* e cobrar o cliente. Alguns fabricantes desenvolvem soluções proprietárias, porém, estas comprometem a portabilidade entre fornecedores.
- Se for necessário alterar uma interface do serviço, os clientes devem ser notificados da alteração para que façam as adaptações necessárias nas aplicações clientes. Outro inconveniente é a necessidade de ter que gerar a descrição do serviço novamente.

3 UM WEB SERVICE PARA BUSCA DE PREÇOS DE PRODUTO

Este capítulo apresenta o desenvolvimento do *web service* proposto. A seção 3.1 explica a importância da formação do preço de venda. A seção 3.2 discute as vantagens da criação de um *web service* para auxiliar o processo de formação do preço de venda. A seção 3.3 apresenta as características do *web service* proposto. E, por fim, a seção 3.4 explica a arquitetura do serviço.

3.1 FORMAÇÃO DO PREÇO DE VENDA

A formação do preço de venda é fundamental para a sobrevivência de uma empresa no mercado. Há vários fatores a serem considerados para decidir qual o preço de venda ideal de um produto ou serviço. Entre eles pode-se destacar o posicionamento estratégico, demanda, mercado, fixação de preços para penetração, para concorrência ou maximização de retorno (OLIVEIRA; CREMA, 2009).

A determinação do preço de venda exige do gestor conhecimento sobre os fatores que dão origem ao preço de venda bem como seus custos. Algumas empresas não conseguem determinar de forma precisa os custos e despesas, tornando difícil a verificação posterior do lucro obtido com as vendas. Isto pode comprometer o crescimento da empresa e até mesmo sua estabilidade econômico-financeira.

Segundo Oliveira e Crema (2009) “decisões referentes à definição do preço envolvem aspectos muitas vezes analisados de forma empírica, baseadas em informações subjetivas, sem qualquer embasamento teórico”. Entretanto, a competitividade do mercado não permite que uma empresa forme seus preços de venda baseadas em informações subjetivas.

Há diversos aspectos que uma empresa deve considerar para formar o preço de venda de seu produto, como análise do ciclo de vida do produto, comportamento do consumidor, mensuração e previsão da demanda. Entre as maneiras destaca-se a análise de mercado. Realizar uma análise de mercado é elencar informações para conhecer profundamente o ambiente no qual a empresa está inserida (SANTOS, 2001).

Uma das etapas da análise de mercado é a análise da concorrência, que avalia a relação dos produtos com a organização (SARDINHA, 1995). Informações sobre produto ou serviço, praça, gerenciamento, promoções, finanças e preços praticados são elencadas. O *web service* proposto neste trabalho auxilia a análise da concorrência, obtendo os preços de mercados praticados pela concorrência.

3.2 IMPORTÂNCIA DA CRIAÇÃO DO *WEB SERVICE* PROPOSTO

Até o presente momento não há no mercado um *software* que implemente diversos métodos de formação do preço de venda, obrigando os gestores a usarem vários *softwares* ou realizarem os cálculos manualmente.

Conscientes deste problema, o Grupo de Pesquisa em Engenharia de Software (GPES) da Universidade Tecnológica Federal do Paraná (UTFPR), Campus Ponta Grossa, está desenvolvendo um *framework* que implementa vários métodos de formação do preço de venda, tais como: formação de preço de venda com base no custo pleno (NASCIMENTO; VARTANIAN, 1999), formação de preço com custeio baseado em atividades (NAKAGAWA, 1995), método do SEBRAE-PR (SEBRAEPR, 2010), entre outros. Um destes métodos, chamado de método baseado nas decisões das empresas concorrentes, analisa o ambiente externo a empresa e utiliza como parâmetro de entrada os preços de venda praticados pelo mercado (SARDINHA, 1995). Por isso, surgiu a necessidade de se construir um *web service*, proposto neste trabalho, para busca de preço de venda de um determinado produto.

Embora o único cliente previsto para o *web service* seja o *framework* do GPES, o serviço pode ser consumido por diversos outros clientes, como, por exemplo: um *software* doméstico que exiba os preços de produtos procurados pelo usuário, um *software* de um dispositivo móvel que notifique o usuário quando um produto desejado estiver disponível por um preço determinado, entre outros.

3.3 CARACTERÍSTICAS DO *WEB SERVICE* PROPOSTO

O *web service* pode ser configurado através de uma interface administrativa para web desenvolvida com JSF (JSF, 2010). Através desta interface o usuário pode:

- cadastrar novos sítios de *e-commerce*;
- atualizar as informações dos sítios já cadastrados;
- definir *as configurações de proxy;*
- informar o número máximo de *threads* de busca de preço sendo executadas simultaneamente. Esta configuração é importante para otimizar o desempenho da aplicação em função do *hardware* e da largura de banda disponível.

Sempre que um sítio é cadastrado, a aplicação obtém seu HTML (*HyperText Markup Language*) e realiza o *parsing* do documento. Este *parsing*, ou seja, a análise do documento e a criação de uma estrutura de dados com os elementos do documento, logo após, procura um formulário de busca baseado em palavras-chaves de vários idiomas. Se este for encontrado, extraem-se os parâmetros necessários para realizar uma requisição. Caso contrário, o usuário é informado que o *web service* não pode realizar buscas nesse sítio e que ele não será cadastrado.

O *web service* foi desenvolvido usando a JAX-WS (*Java API for XML – Web Service)* (JAX-WS, 2010) com as extensões providas pela WSIT (*Web Service Interoperability Technology*). A JAX-WS fornece os métodos básicos para a criação e disponibilização de *web services* enquanto as extensões WSIT provêem mecanismos para garantir interoperabilidade, otimização e confiabilidade de mensagens e suporte a transações.

A interface do *web service* disponibiliza o método *getProductPrice* que aceita como parâmetro o nome do produto do tipo *String*. Quando um consumidor invoca o método, o serviço recupera do banco de dados os sítios cadastrados bem como seus parâmetros de requisição. Estes parâmetros são os valores esperados pelo sítio e, normalmente, são compostos por um parâmetro para o nome do produto e outro para categoria do produto. São, então, criadas as requisições e enviadas para os sítios.

Assim que são obtidos os HTMLs resultantes das requisições, eles são analisados pelo *parser*. Os preços e as descrições obtidas dos HTML são retornados pelo *web service* para o consumidor.

Todo o processo de criação das requisições, envio e análises dos retornos são feitas em paralelo até o número máximo de *threads* de requisição definido pelo usuário. Caso o usuário não tenha definido nenhum valor explicitamente, o valor padrão é dez. O funcionamento do *web service* proposto está ilustrado na Figura 8.

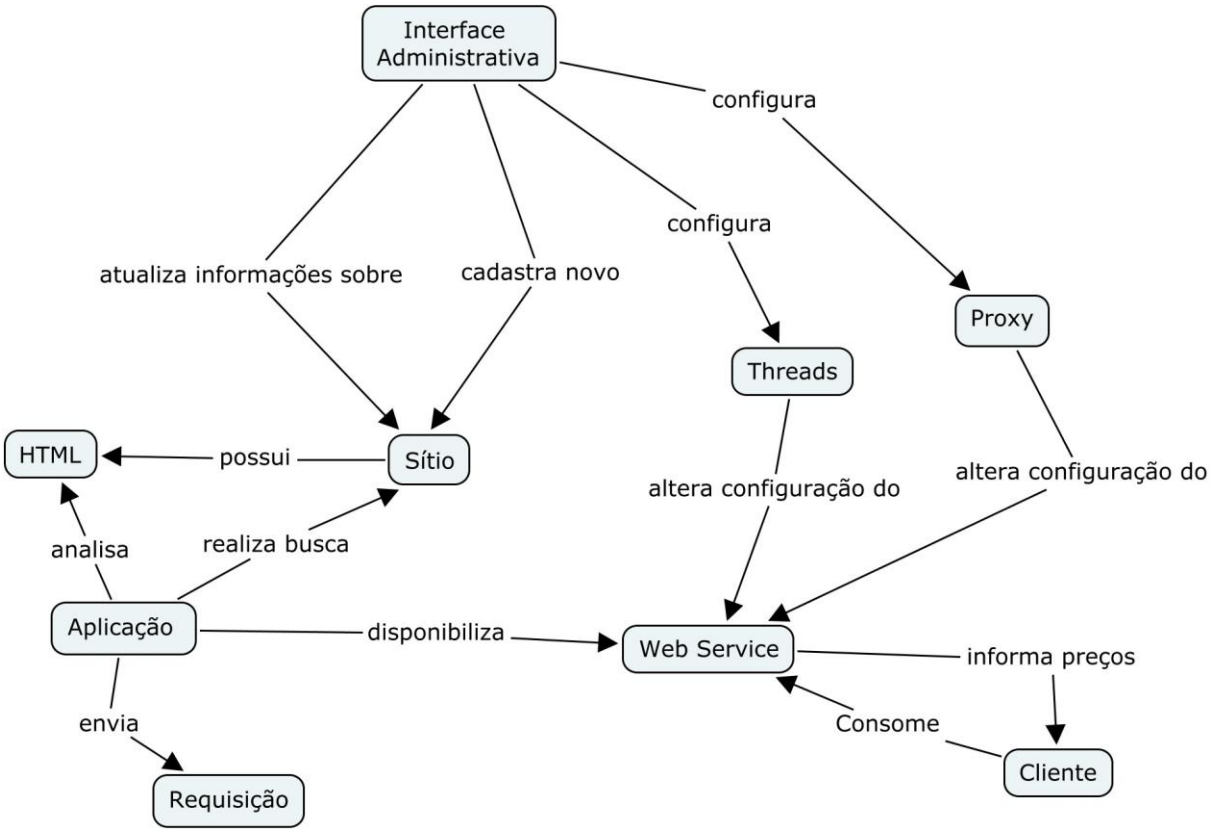

Figura 8 - Modelo de Funcionamento do *Web Service* Proposto
Fonte: Autoria Própria

Observando a figura 8, pode-se notar que é possível cadastrar novos sítios bem como atualizar as informações sobre os já cadastrados. Pode-se também definir as configurações de *proxy* e *threads* através da interface administrativa. A aplicação realiza busca em sítios, envia requisições e analisa HTMLs. Esta aplicação disponibiliza um *web service* para que os clientes possam consumir o serviço fornecido pela aplicação.

3.4 ARQUITETURA DO *WEB SERVICE* PROPOSTO

Esta seção apresenta a arquitetura do *web service* proposto. Ela está divida em duas subseções: a primeira discute a arquitetura de alto nível do serviço e a segunda explica a arquitetura de baixo nível.

As ferramentas utilizadas para realizar a engenharia reversa do *web service* e da base de dados foram o Netbeans 6.7.1 (NETBEANS,2010) e o MySQL Workbench 5.2.17-beta (MYSQL,2010), respectivamente.

3.4.1 Arquitetura de Alto Nível

A figura 9 mostra o diagrama de pacotes no formato UML 2.0 do *web service*, sendo composto por dois módulos principais: o Cliente e o Web Service

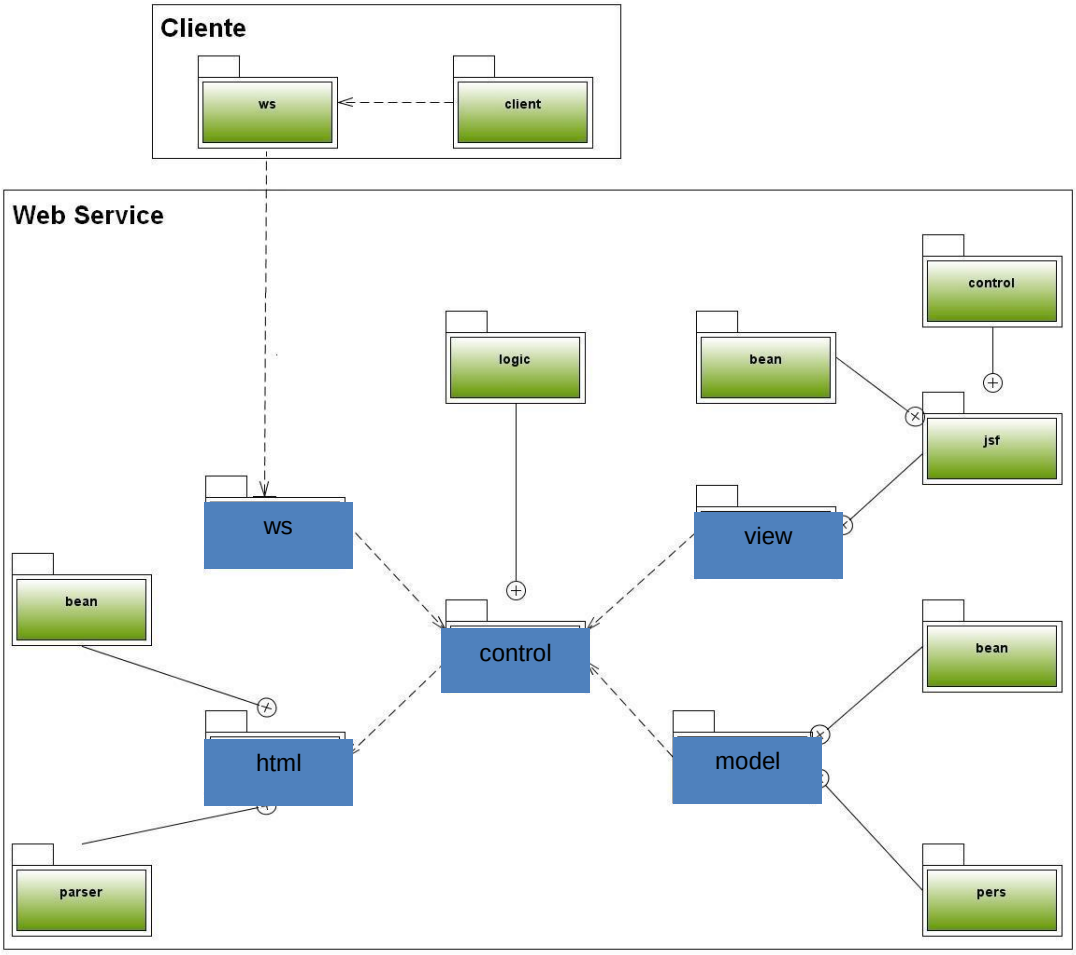

Figura 9 - Diagrama de Pacotes
Fonte: Autoria Própria

A arquitetura do cliente é composta por dois pacotes: *ws* e *client.* O primeiro contém as classes geradas automaticamente pelo Netbeans para acessar o *web service* e o segundo contém as classes responsáveis pela interface gráfica com o usuário e de lógica da aplicação.

A arquitetura do *web service* é composto por cinco pacotes básicos: *control, model, view, ws e html*, representados na cor azul na Figura 9.

O pacote *html* contém dois pacotes: *parser* e *bean*. O primeiro contém as classes responsáveis por analisar os HTMLs dos sítios e extrair informações específicas. O pacote *bean* contém as classes que representam os elementos do *html*, tais como: formulários, campos de texto, entre outros. Todas as instâncias destas classes são POJOs (*Plain Old Java Object*), ou seja, objetos que possuem somente propriedades e métodos *set* e *get*.

O pacote *control* possui as classes responsáveis por converter os valores para um formato adequado para os objetos responsáveis pela persistência. O pacote *logic* contém as classes responsáveis por analisar os dados retornados pelas classes do pacote *html*, criar requisições, enviá-las e analisar os resultados.

O pacote *model* contém dois pacotes: *bean* e *pers*. O primeiro contém as classes que representam as entidades do banco de dados. O segundo é formado pelas classes responsáveis por realizar as operações de cadastro, exclusão, atualização e pesquisa no banco de dados.

O pacote *view* é constituído por classes responsáveis pela interface gráfica do usuário. O *web service* possui uma interface desenvolvida em *jsf* para administrar o serviço. As classes responsáveis pela regra de negócios da interface estão nos pacotes *control* e *bean*.

O modelo de funcionamento, descrito na Seção 2.3 e ilustrado na Figura 1, possui os seguintes participantes em um *web service*: *container* de serviço, provedor de serviço, UDDI e cliente. Neste trabalho, foi utilizado como container para o *web service* o servidor de aplicação Glassfish (GLASSFISH, 2010) e como cliente uma aplicação *desktop*. Não foi utilizado um diretório de registro com o UDDI porque o *web service* não estará disponível para acesso público, somente para o GPES em primeiro momento.

Os modelos arquiteturais apresentados na seção 2.4.1 foram implementados nos *web services* pelas API JAX-WS com as extensões providas pela WSIT. A figura

10 ilustra as partes que compõem as APIs e como estão divididas as responsabilidades entre as elas.

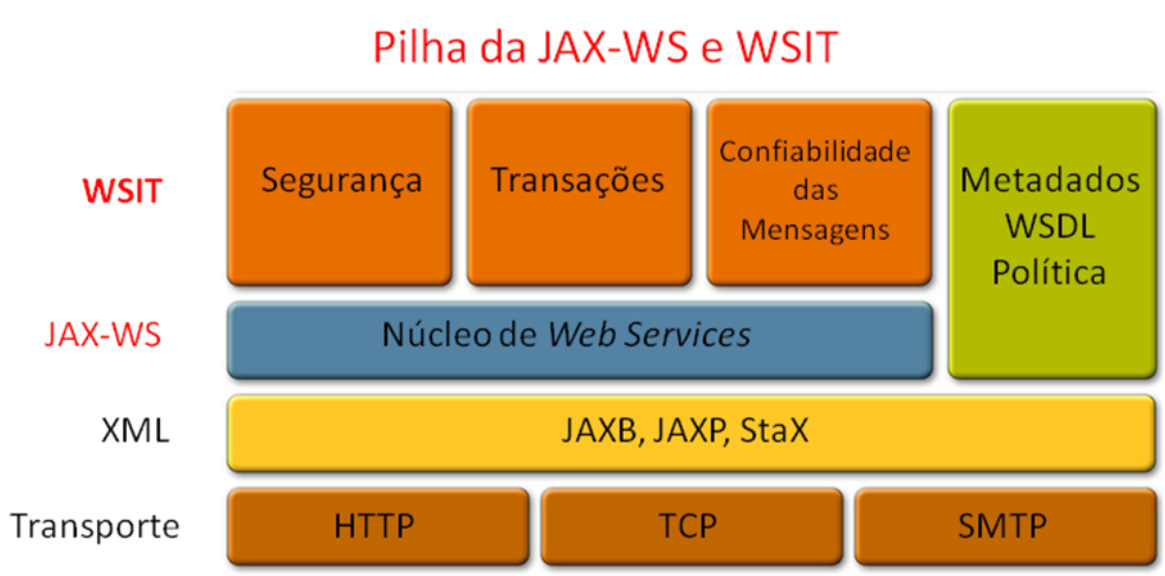

Figura 10 – Pilha da JAX-WS e WSIT
Fonte: Adaptado de Metro (METRO, 2010)

Os conceitos e modelos apresentados na seção 2.4.1 são criados pela JAX-WS e alguns mais específicos, como confiabilidade das mensagens, segurança, política, entre outros, são gerados pela WSIT. As funções específicas relacionadas à XML e ao transporte apresentados na figura 10 fogem ao escopo deste trabalho e estão sendo apresentadas na figura somente para uma melhor compreensão de como estão divididos os papéis na arquitetura das APIs JAX-WS e WSIT.

3.4.2 Arquitetura de Baixo Nível do *Web Service*

A seguir será apresentada a arquitetura de baixo nível do *web service* proposto.

3.4.2.1 Diagrama de Classes do pacote *html*

O pacote *html* contém uma única classe, *EntityConverter*, e uma interface, *HTMLConstant*, ilustradas na Figura 11.

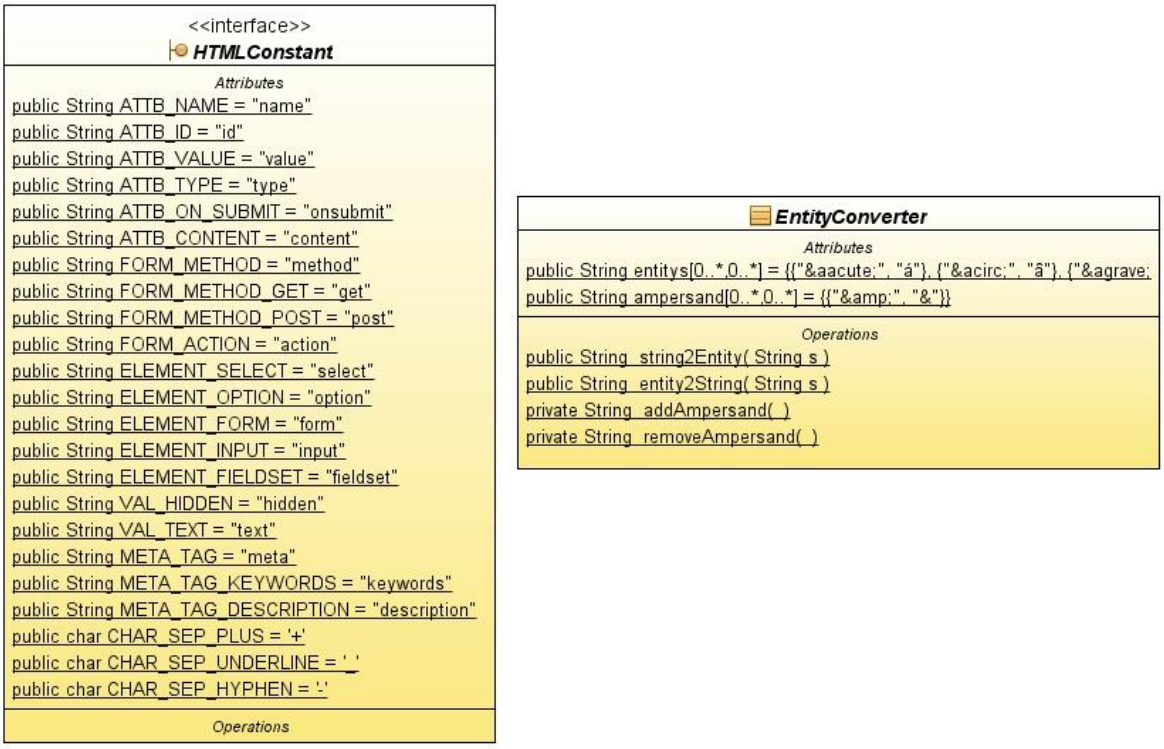

Figura 11 - Diagrama de Classes do pacote *html*
Fonte: Autoria Própria

Alguns caracteres têm um significado especial em HTML, como, por exemplo, “<” que indica o início de uma tag HTML. Este tipo de caractere não é exibido, pois são interpretados pelo *browser* (ou programa que estiver lendo o HTML). Para que este tipo de caractere seja exibido é necessário utilizar as entidades HTML.

A classe *EntityConverter* é responsável por converter entidades html, tais como á, &aatilde;, à entre outras, em caracteres convencionais e vice-versa (de caracteres convencionais para entidades html).

A interface *HTMLConstant* possui várias constantes que representam elementos, atributos e valores de páginas HTMLs, como, por exemplo, os elementos *form* e *select*, os atributos *action* e *method* e os valores *get* e *post*.

3.4.2.2 Diagrama de classes do pacote *html.bean*

As classes deste pacote, visualizadas na Figura 12, representam elementos *html*. A classe *HTMLElement* contém os atributos comuns aos demais elementos HTML é estendida pela classe *Form*, que representa os formulários HTMLs.

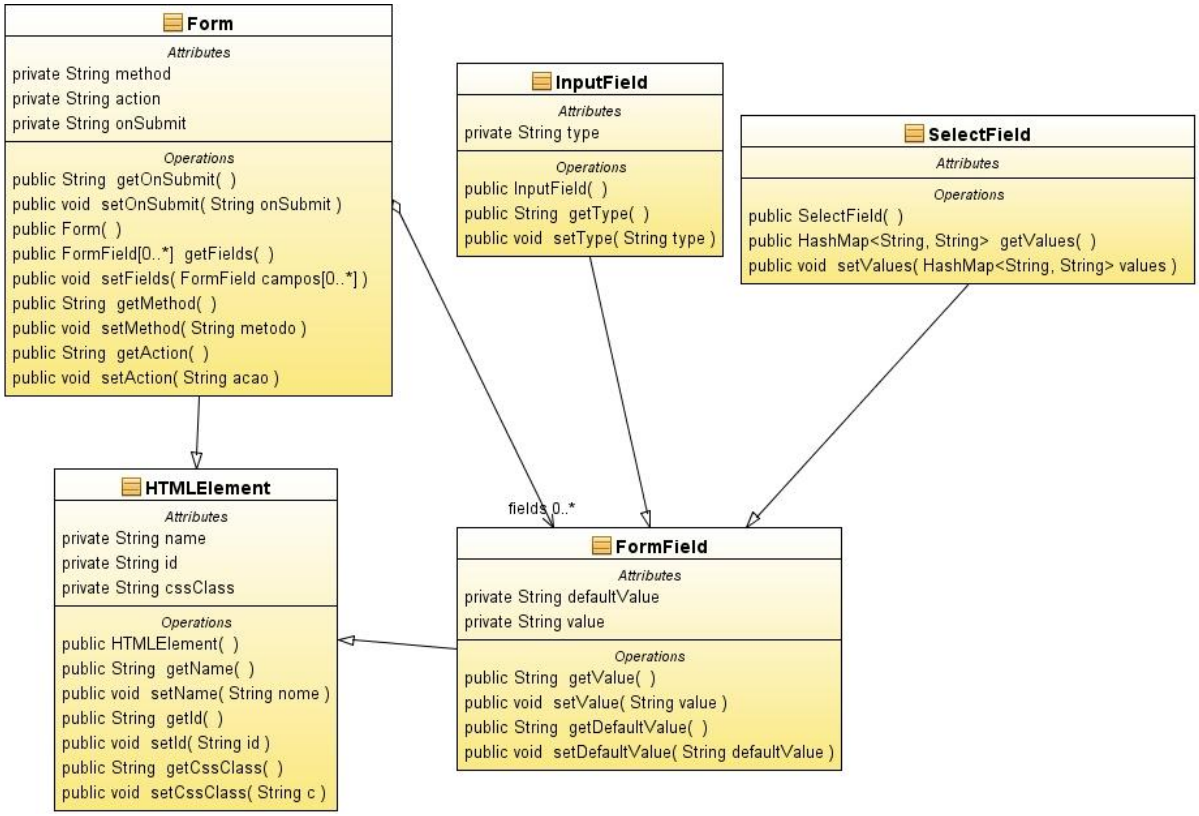

Figura 12 - Diagrama de Classes do Pacote *html.bean*
Autoria: Fonte Própria

A classe *Form* possui um *ArrayList* de *FormField*, que representam os campos de formulários HTMLs. A classe *FormField* possui os atributos comuns de campos HTMLs e as classes *InputField* e *SelectField* contém atributos específicos para campos de entradas e de seleção, respectivamente. Todas as classes do pacote herdam direta ou indiretamente de *HTMLElement*.

3.4.2.3 Diagrama de classes do pacote *html.parser*

As classes do pacote *html.parser* estão ilustradas na Figura 13. A única classe pública do pacote *html.parser* é a *ParserHTML*. Por meio dela é possível realizar as seguintes tarefas: extrair meta-tags dos HTMLs, obter a codificação de uma página e encontrar formulários e preços.

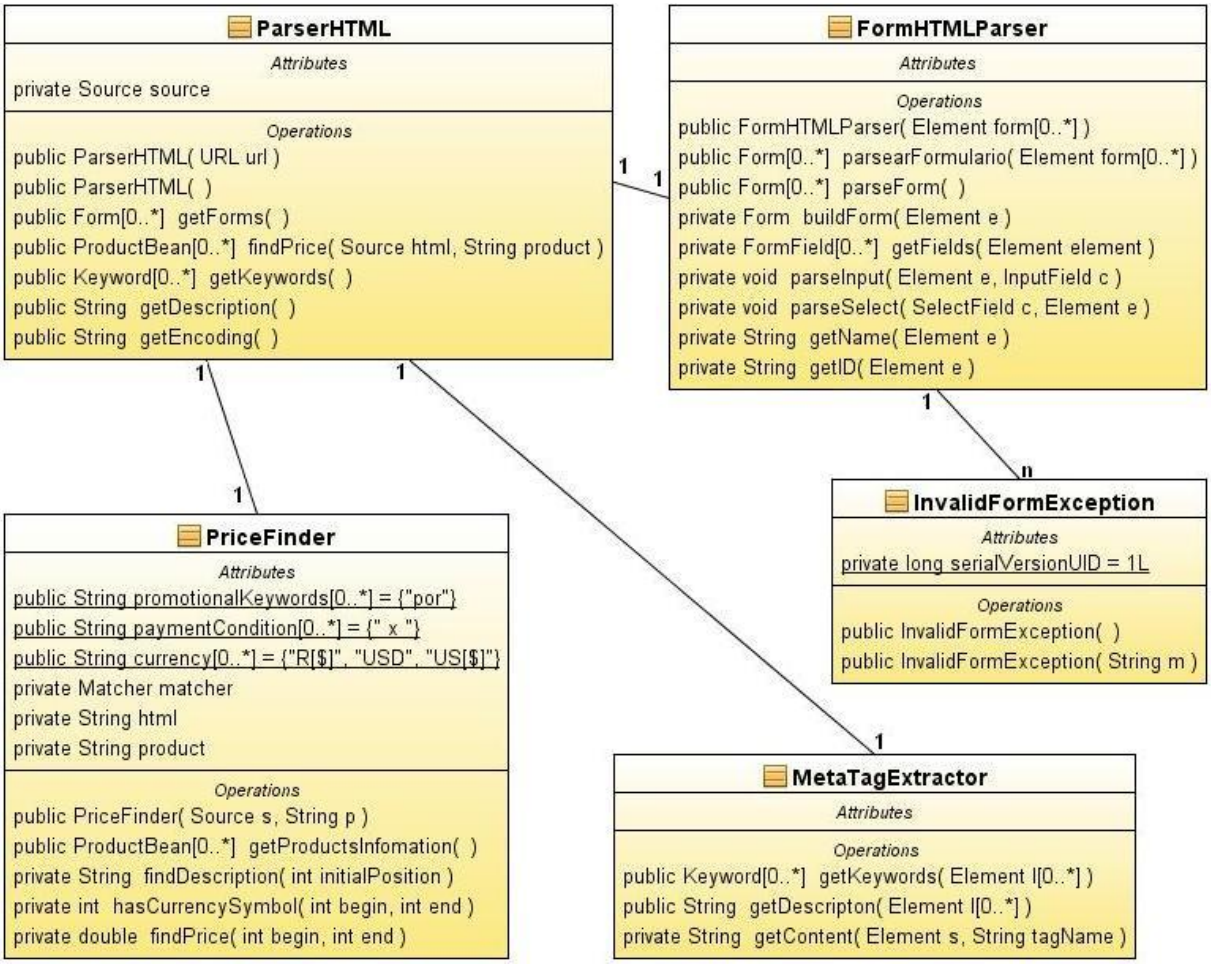

Figura 13 - Diagrama de Classes do Pacote *html.parser*
Fonte: Autoria Própria

É possível obter os formulários contidos em um HTML através do método *getForms*, que retorna um vetor de objetos do tipo *html.bean.Form*. Este método utiliza um objeto do tipo *FormHTMLParser* para buscar os formulários e, caso não seja encontrado algum atributo fundamental para a aplicação, como a *action* do formulário, uma *InvalidFormException* é lançada.

Há dois métodos para obter os conteúdos das meta-tags: *getKeywords* e *getDescription*. O primeiro retorna as palavras-chaves associadas à página e o segundo a descrição do sítio. Os dois métodos utilizam um objeto do tipo *MetaTagExtractor*.

Através do método *findPrice* é possível encontrar os preços de um produto em uma página HTML, usando um objeto do tipo *PriceFinder.* Caso não seja encontrado algum preço no HTML, o método retorna *null.*

3.4.2.4 Diagrama de classes do pacote *control*

As classes deste pacote mostradas na Figura 14 são responsáveis por formatar os valores para o padrão esperado pelos objetos responsáveis pela persistência.

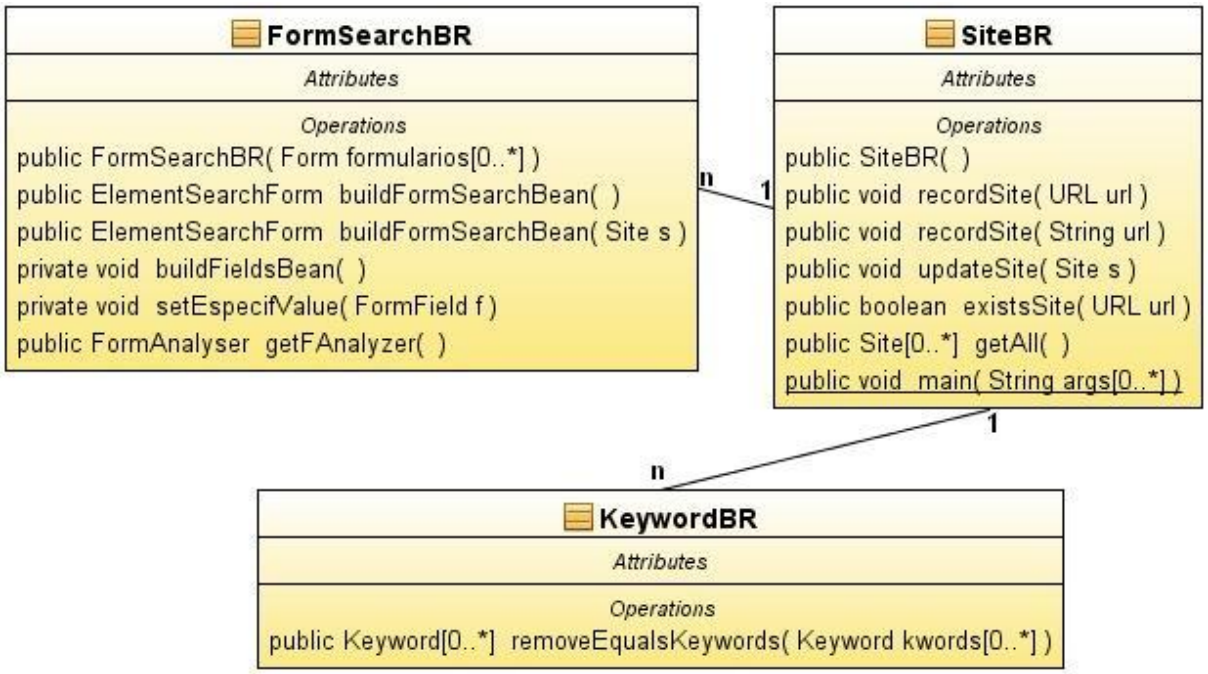

Figura 14 - Diagrama de Classes do Pacote *control*
Fonte: Autoria Própria

A *SiteBR* possui os métodos responsáveis por cadastrar e atualizar sítios bem como verificar a existência destes no banco de dados. Para realizar as operações de cadastro e atualização, a SiteBR utiliza, além de seus próprios métodos, os métodos das classes *KeywordBR* e *FormSearchBR.*

É comum em sítios que uma mesma palavra-chave seja repetida duas vezes ou mais na *meta-tag*. Por isto, para evitar que existam duas palavras-chaves iguais no banco de dados, a classe *KeywordBR* possui o método *removeEqualsKeywords*, que exclui as palavras-chaves repetidas. Este método é chamado pela classe *SiteBR* sempre que uma operação de atualização ou cadastro for realizada.

A classe *FormSearchBR* é responsável por montar os *beans* do formulário de busca do sítio que será persistido ou atualizado. A classe *SiteBR* utiliza a classe *FormSearchBr* para obter os *beans* do formulário do sítio que será persistido ou atualizado.

3.4.2.5 Diagrama de classes do pacote *control.logic*

As classes deste pacote são responsáveis por analisar as informações sobre os HTMLs retornados pela classe *html.parser.ParserHTML*, criar requisições, enviá-las e analisar seus resultados. Estas classes estão ilustradas na Figura 15.

A classe *FormAnalyser* é a responsável por encontrar os formulários de buscas nas páginas HTML. Esta busca é realizada por palavras-chaves de diversos idiomas. Quando um formulário de busca é encontrado, ele é analisado e um *bean* do formulário e seus campos é montado e retornado.

A classe *RequisitionMaker* é a responsável por criar e realizar as requisições. Através da classe *control.SiteBR* obtém-se os parâmetros necessários para realizar as requisições para cada sítio e, usando o nome do produto informado pelo cliente do *web service*, criam-se os objetos do tipo *Request.*

A classe *Request* é abstrata e *GETRequest* é uma implementação concreta de um tipo de requisição. A classe *GETRequest* recebe os parâmetros e utiliza a *URLMaker* para montar a URL da requisição. Se houver um parâmetro inválido, uma *InvalidParamException* é lançada.

A classe *RequisitionMaker* manda as requisições e, se houver algum erro no envio, uma *InvalidRequestException* é lançada. Após enviar, ela obtém os resultados das requisições, passa os HTMLs resultantes para as classes responsáveis pelo *parsing,* que retornam as descrições e os preços dos produtos encontrados. Por fim, a *RequisitionMaker* monta o vetor de *ProductBean* com as informações encontradas e retorna este para a interface do serviço.

3.4.2.6 Diagrama de classes do pacote *ws*

Este pacote contém somente uma classe, *WsInt*, ilustrada na Figura 16. É a partir desta classe que será gerada a interface do serviço.

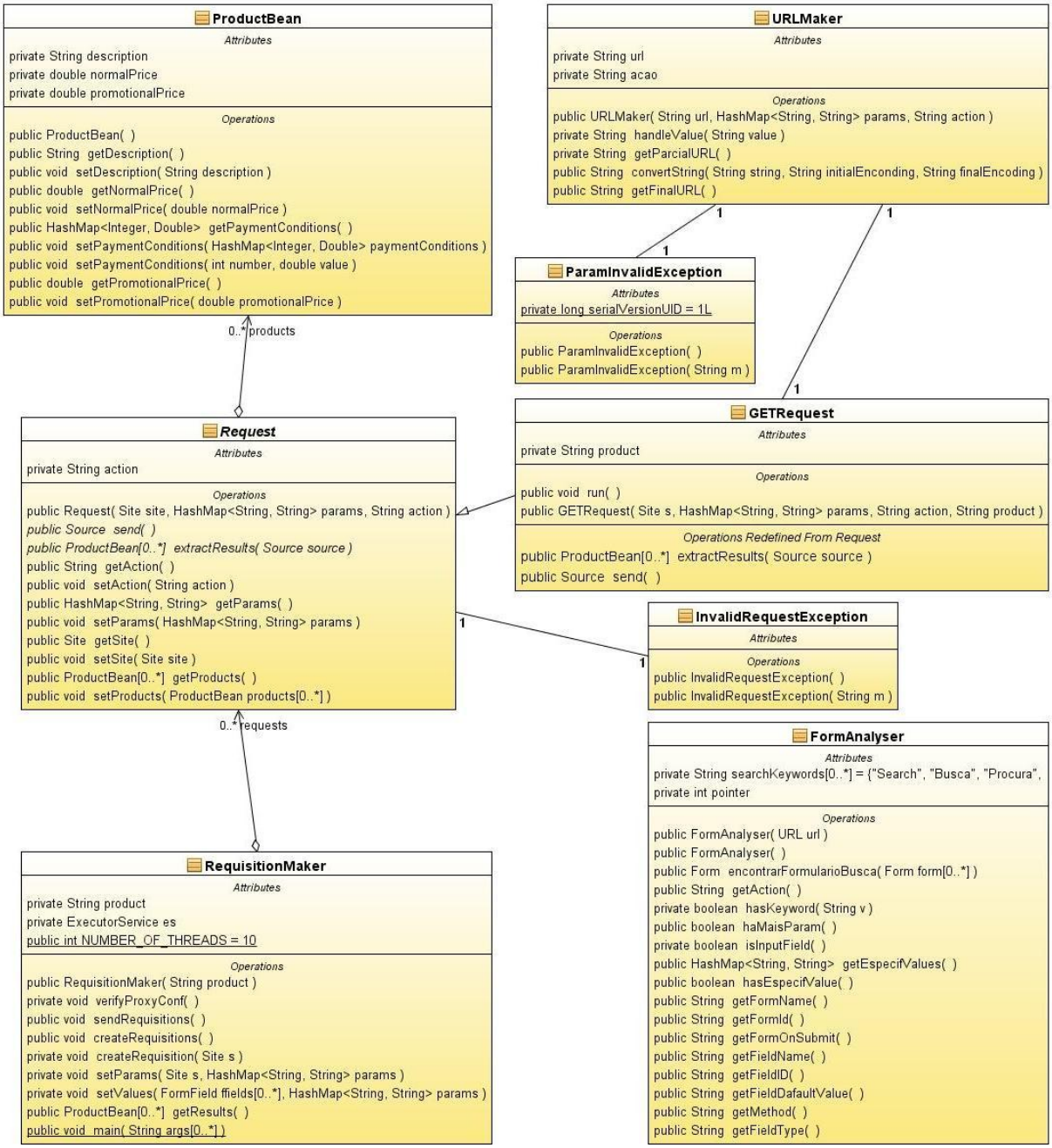

Figura 15 - Diagrama de Classes do Pacote *control.logic*
Fonte: Autoria Própria

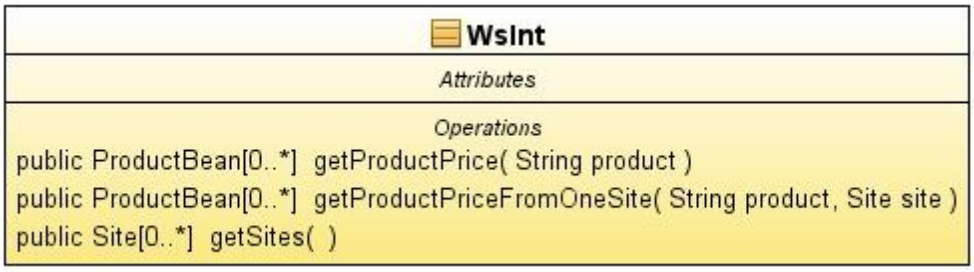

Figura 16 - Diagrama de Classes do Pacote *ws*
Fonte: Autoria Própria

Os métodos *getProductPrice, getProductPriceFromOneSite* e *getSites* são disponibilizados para os clientes que irão consumir o *web service*. O primeiro método realiza uma busca pelos preços do produto em todos os sítios cadastrados no banco de dados. O segundo realiza uma busca pelos preços do produto em um único sítio, que é passado como parâmetro para o método junto com o nome do produto. Um ponto a ser considerado sobre este método é o de que o *web service* aceita somente sítios cadastrados no banco de dados. Assim sendo, é disponibilizado o método *getSites*, que retorna todos os sítios cadastrados no banco dados. Desta forma, o cliente tem como descobrir quais são os sítios cadastrados e pode realizar uma busca em um sítio específico e ignorar os demais.

A especificação mais detalhada da interface de serviço criada a partir da classe *WsInt* é apresentada no apêndice A.

3.4.2.7 Diagrama de classes do pacote *model*

Este pacote contém uma única classe, *HibernateUtil,* que é responsável por fornecer uma fábrica de sessões do *Hibernate*. Esta classe está ilustrada na Figura 17.

Figura 17 - Diagrama de Classes do Pacote model
Fonte: Autoria Própria

3.4.2.8 Diagrama de classes do pacote *model.bean*

As classes deste pacote mapeiam as entidades do banco de dados. Elas são utilizadas pelo Hibernate (HIBERNATE,2010) para trabalhar com o banco de dados. A figura 18 mostra o diagrama entidade-relacionamento da base de dados.

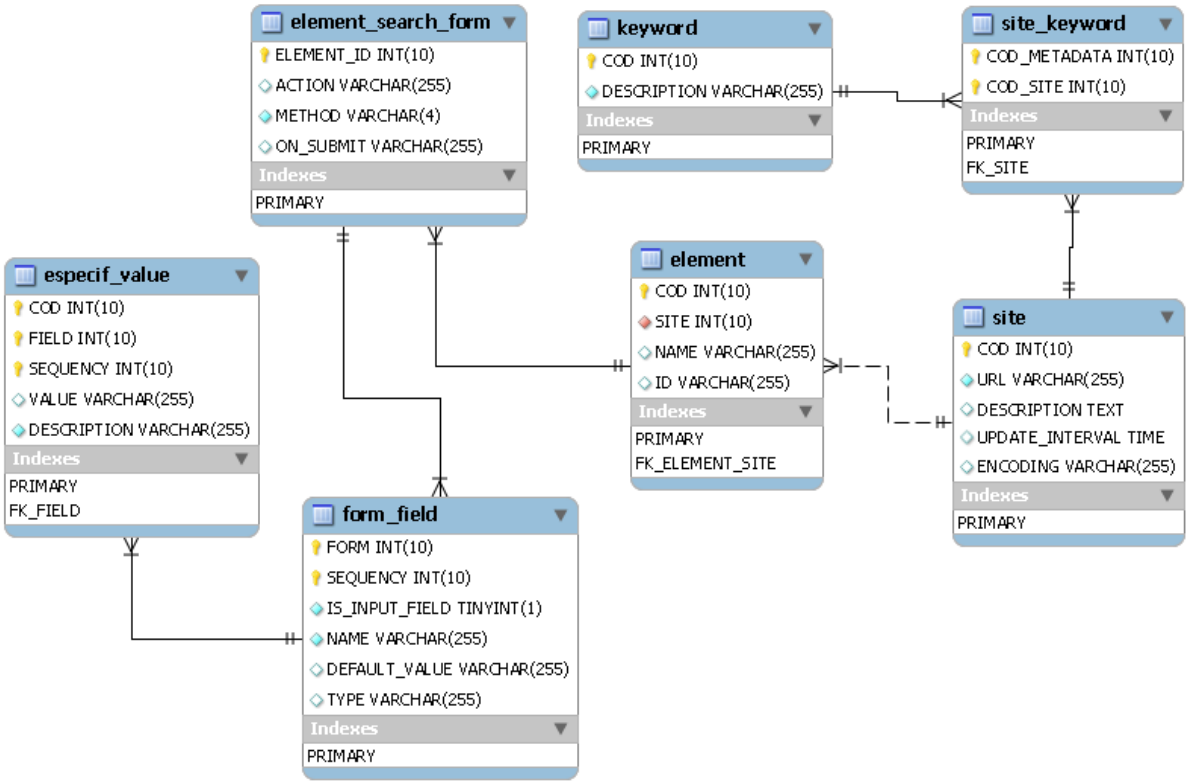

Figura 18 - Diagrama Entidade-Relacionamento do Banco de Dados
Fonte: Autoria Própria

A entidade *keyword* armazena as palavras-chaves dos sítios cadastrados. A entidade *site_keyword* relaciona as palavras chaves aos sítios cadastrados na entidade *site*.

A entidade *element* contém as propriedades comuns aos diversos elementos HTML, como *name* e *id.* As entidades *element_search_form*, *form_field* e *especific_value* armazenam, respectivamente, informações sobre o formulário de busca do sítio, seus campos e seus valores específicos.

A figura 19 mostra o diagrama de classes do pacote. Como é possível observar nos diagramas, para cada entidade do banco de dados foi criada uma classe correspondente. Estas classes são anotadas para serem utilizadas pelo Hibernate.

Para as tabelas que possuem chave composta, ou seja, mais de uma coluna da tabela compõem a chave primária, foram criadas classes dedicadas somente ao mapeamento das chaves. Estas classes possuem “Id” no nome, como é o caso de *EspecifValueId* e *FormFieldId*.

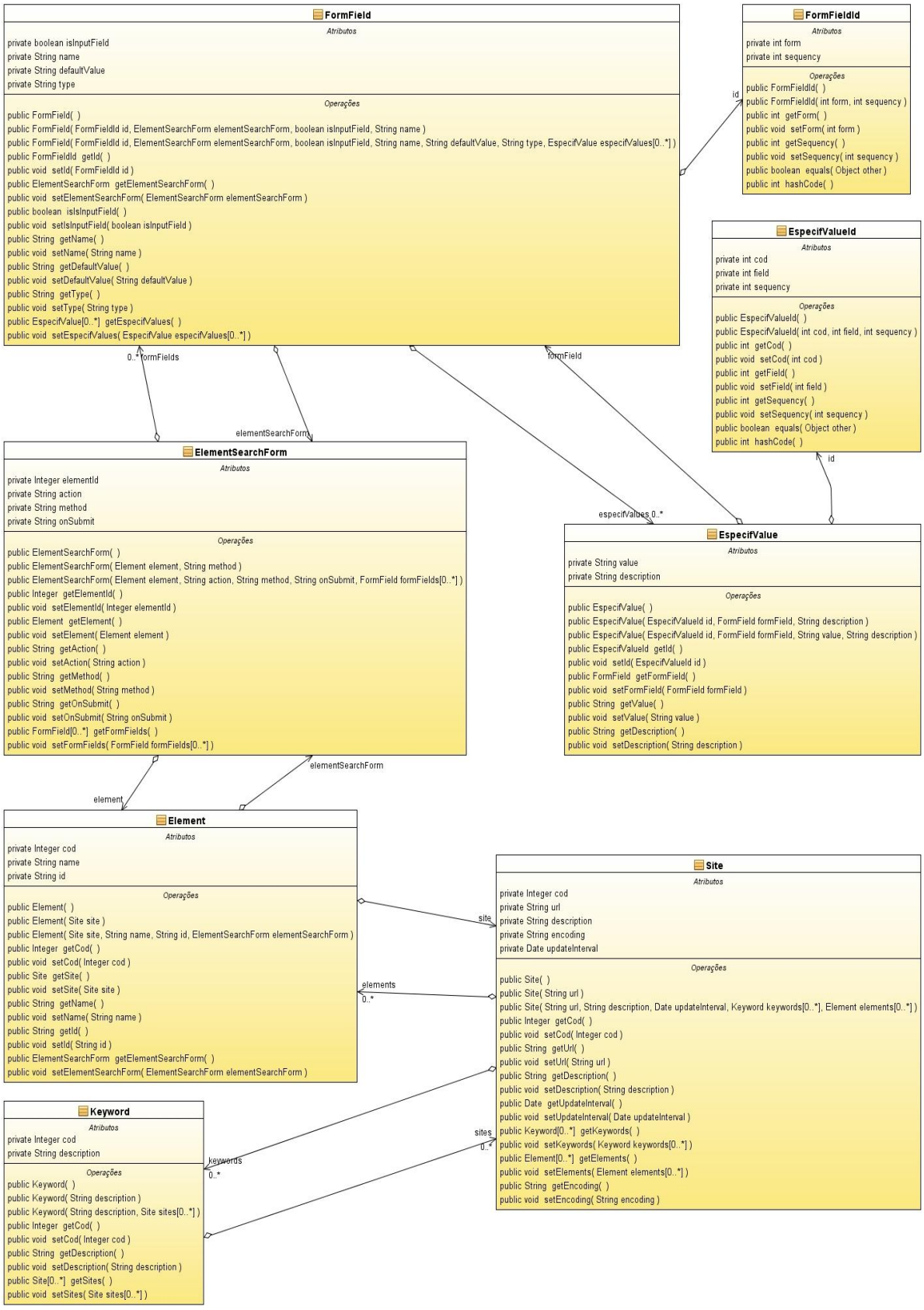

Figura 19 - Diagrama de Classes do Pacote *control*
Fonte: Autoria Própria

3.4.2.9 Diagrama de classes do pacote *model.pers*

Este pacote contém as classes responsáveis pela interação com o banco de dados e estão ilustradas na Figura 20.

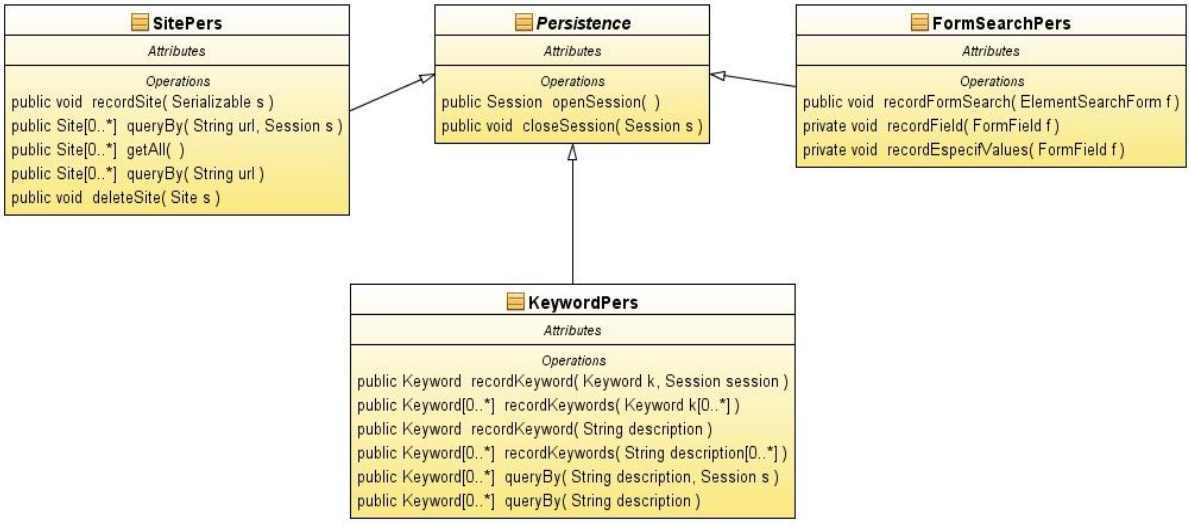

Figura 20 - Diagrama de Classes do Pacote *model.pers*
Fonte: Autoria Própria

A classe *Persistence* contém os métodos utilizados pelas demais classes de persistência, tais como: abrir e fechar sessão do Hibernate.

A classe *SitePers* possui os métodos para gravar, pesquisar e excluir os sítios do banco de dados. As classes *KeywordPers* e *FormSearchPers* realizam as mesmas operações para, as respectivas entidades *Keyword* e *FormSearch* e suas relacionadas.

3.4.2.10 Diagrama de classes do pacote *view.jsf.control*

As classes deste pacote são responsáveis por efetuar as alterações de configuração do *web service* realizadas através da interface. Estas classes estão mostradas na Figura 21.

A classe *ProxyConfControl* é a responsável por atualizar ou remover a configuração de *proxy* do serviço. Para manter a configuração do *proxy* quando o servidor é reiniciado, ela mantém um arquivo chamado *proxyConf.txt* com a configuração gravada.

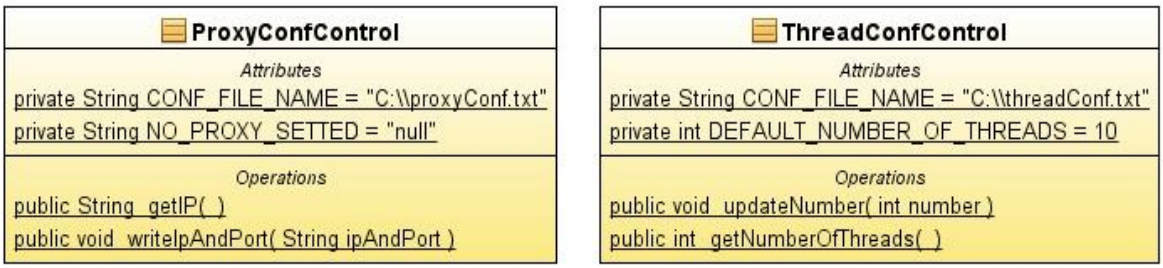

Figura 21 - Diagrama de Classes do Pacote *view.jsf.control*
Fonte: Autoria Própria

A classe *ThreadConfControl* é a responsável por definir o número de *threads* de requisição sendo executadas simultaneamente. Ela também utiliza um arquivo, chamado *threadConf.txt*, para manter a configuração do *pool* quando o servidor é reiniciado.

3.4.2.11 Diagrama de classes do pacote *view.jsf.bean*

Este pacote, mostrado na Figura 22, contém os *beans* gerenciados (*managed beans*) utilizados pela interface JSF.

A classe *ApplicationBean* é um *bean* de escopo de aplicação. Ele contém as configurações de *proxy* e *threads* em uso.

As classes *ThreadRequest* e *ProxyRequest* são *beans* de escopo de requisição e são utilizadas pelas páginas de configuração de *Threads* e *Proxy*, respectivamente.

A classe *SiteRequest* é um *bean* de escopo de requisição é usada pelas interfaces JSF responsáveis pela atualização e cadastro de sítios. A *SiteRequest* utiliza o *bean Site* para armazenar informações sobre os sítios que precisam ser atualizados.

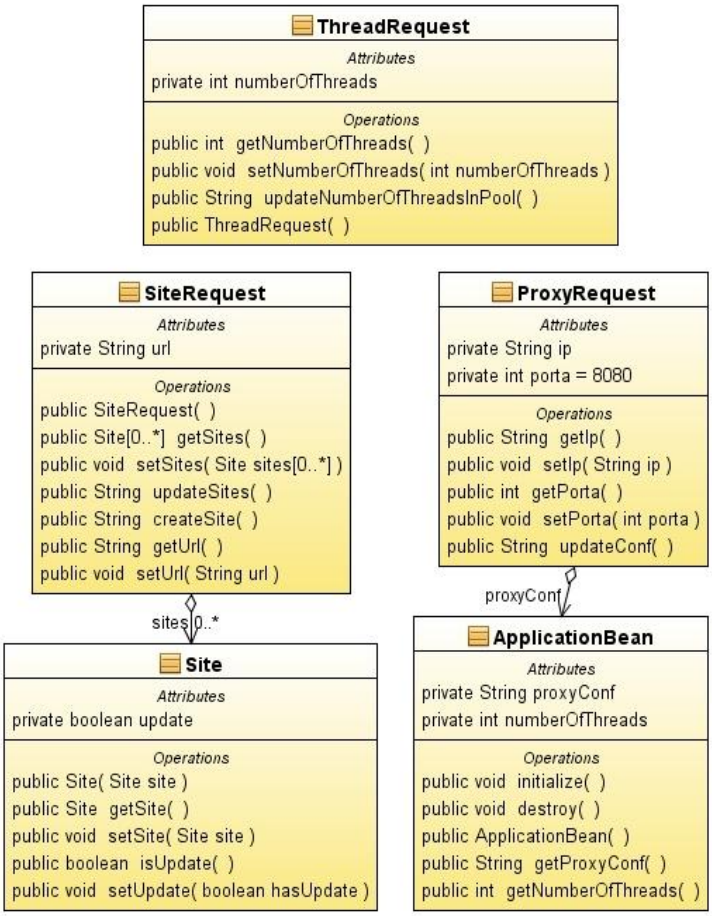

Figura 22 - Diagrama de Classes do Pacote *view.jsf.control*
Fonte: Autoria Própria

3.4.3 Arquitetura de Baixo Nível do Cliente

A fim de testar o *web service* proposto e demonstrar seu funcionamento, foi criado um cliente *desktop.*

As seções 3.4.3.1 e 3.4.3.2 apresentarão a arquitetura de baixo nível do cliente.

3.4.3.1 Diagrama de classes do pacote *ws* do cliente

As classes deste pacote são geradas automaticamente pelo NetBeans e são responsáveis pela interação com o *web service*.

A figura 23 mostra o diagrama de classes do pacote *ws.*

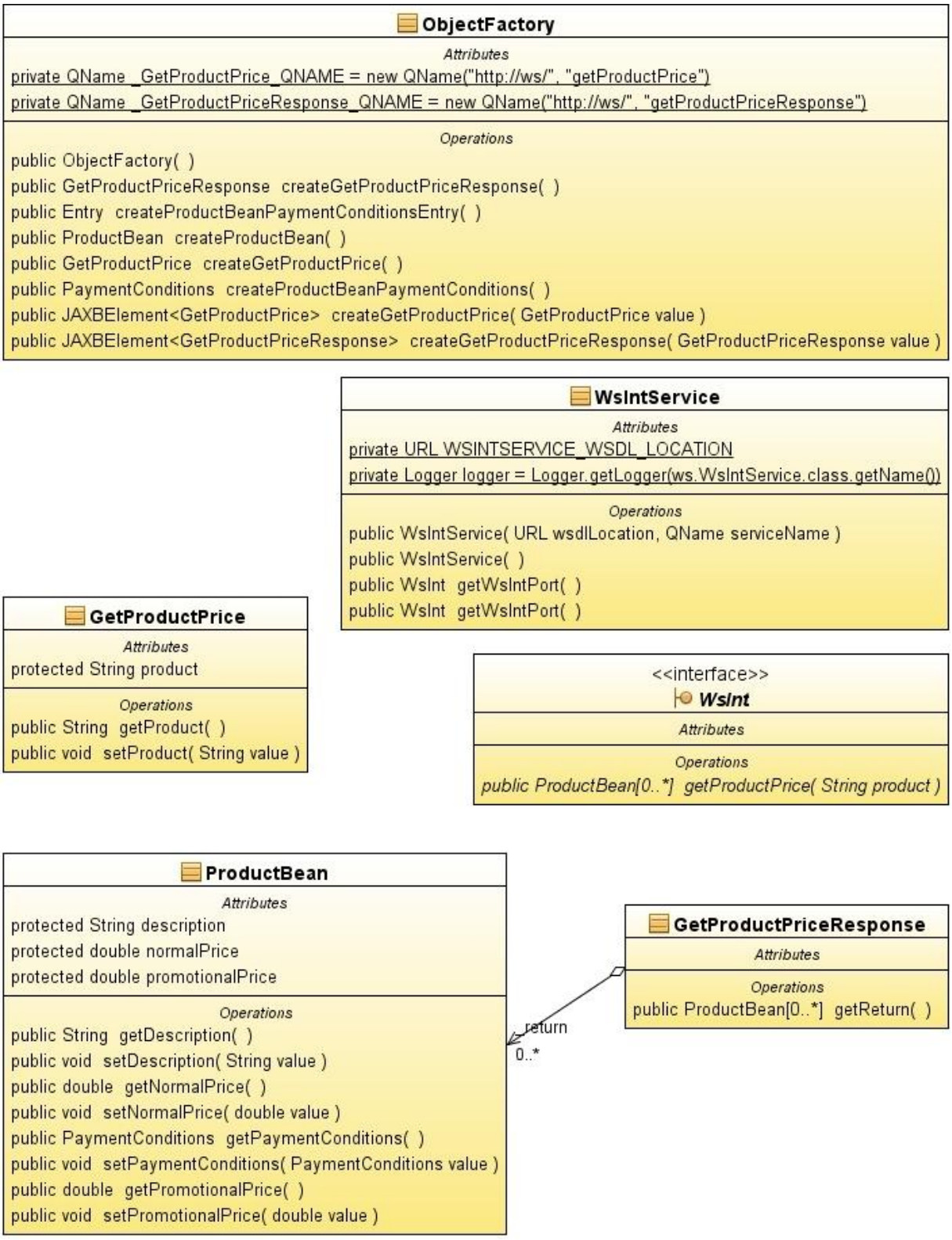

Figura 23 - Diagrama de Classes do Pacote *ws* do Cliente
Fonte: Autoria Própria

A classe *ObjectFactory* fornece diversos métodos para criação de instâncias de objetos utilizados para a interação com o *web service*. A classe *WsIntService* é quem cria a conexão com o *web service*. Quando uma conexão com o *container* do

serviço é estabelecida, um objeto do tipo *WsInt* é retornado. É através deste objeto que os métodos do serviço podem ser invocados.

O *web service* retorna um vetor de *ProductBean* como resposta. A classe *ProductBean* é a implementação do tipo do objeto retornado pelo *web service* enquanto a *GetProductPriceResponse* é a responsável por retornar o vetor de *ProductBean*.

3.4.3.2 Diagrama de classes do pacote *client*

A figura 24 apresenta o diagrama de classes do pacote *client*.

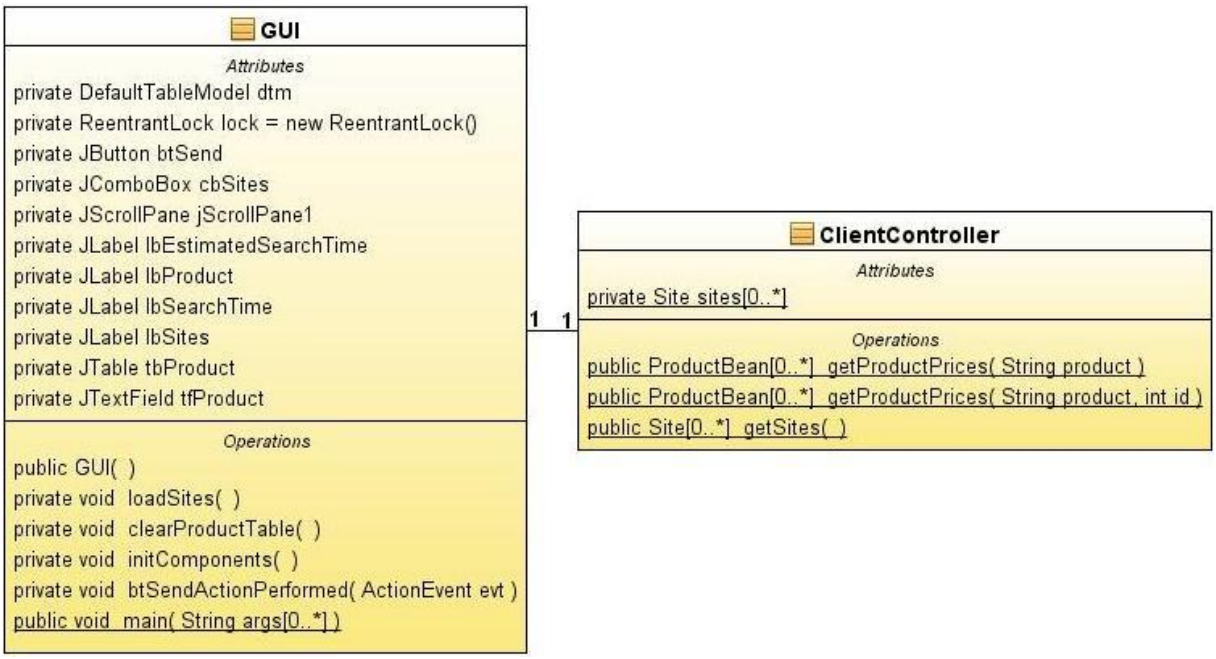

Figura 24 - Diagrama de Classes do Pacote *client*
Fonte: Autoria Própria

A classe *GUI* é responsável por gerar a interface gráfica do usuário. A *GUI* utiliza a classe *ClientController* para invocar os métodos do *web service* e obter os retornos do serviço.

No próximo capítulo serão apresentados diagramas de seqüência que ilustram a relação entre os objetos das classes apresentadas neste capítulo.

4 RESULTADOS

Este capítulo mostra os resultados atingidos por meio da elaboração deste trabalho, bem como suas limitações. A seção 4.1 exibe o funcionamento do *web service* proposto. A seção 4.2 faz uma pequena análise sobre o desempenho do *web service* desenvolvido. A seção 4.3 discute as vantagens de se utilizar o serviço criado. E, por fim, a seção 4.4 apresenta algumas restrições do serviço.

4.1 FUNCIONAMENTO DO *WEB SERVICE*

Esta seção está divida em duas subseções. A primeira apresenta as funções administrativas do *web service* desenvolvidas e explica como usá-las. A segunda relata como um cliente *desktop* utiliza o serviço de busca de preços.

4.1.1 Funções Administrativas

Inicialmente, o administrador do *web service* precisa definir configurações de *proxy* (se houver) e threads. A figura 25 mostra a interface para definição de configurações de *proxy*.

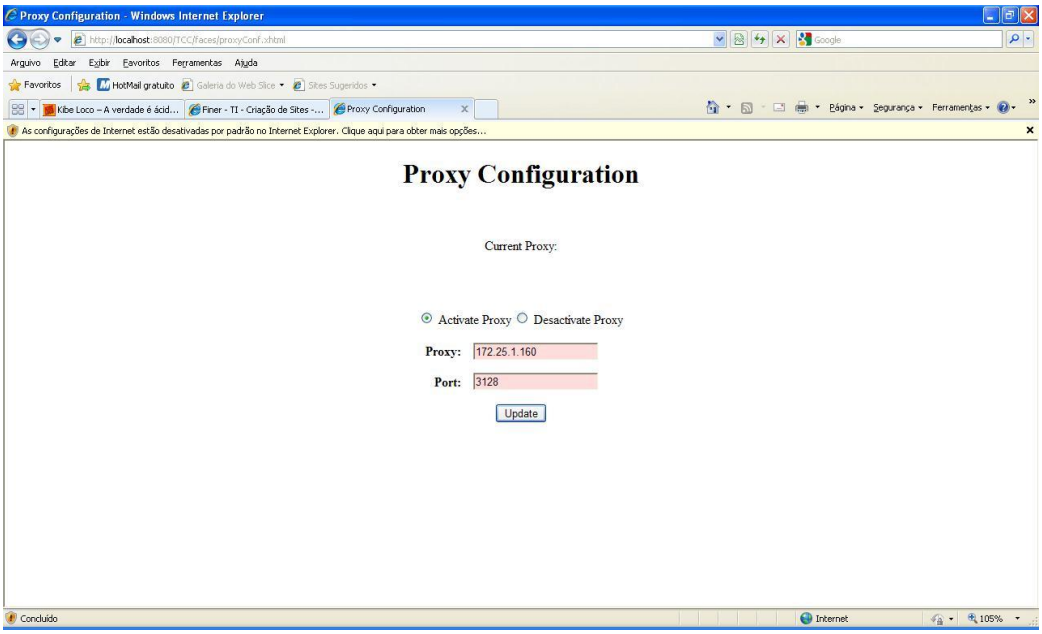

Figura 25 - Interface para definição de configurações de *proxy*

Fonte: Autoria Própria

Se for necessário definir configurações de *proxy,* deve-se selecionar a opção "*Activate Proxy*" e informar o endereço e a porta do *proxy*. Caso contrário, basta selecionar a opção "*Desactivate Proxy"*.

O diagrama de seqüência apresentado na figura 26 mostra quais objetos são utilizados para definir as configurações de *proxy*. Estes objetivos foram definidos na arquitetura do web *service* proposto no Capítulo 3.

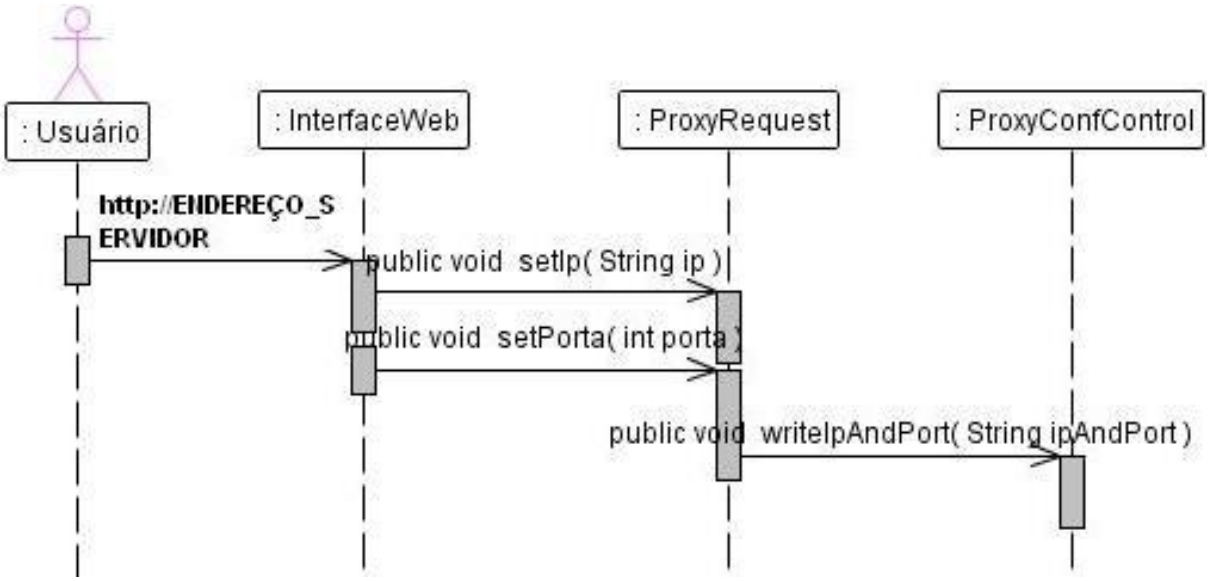

Figura 26 - Diagrama de Seqüência da Definição de Configuração de Proxy
Fonte: Autoria Própria

É possível observar pelo diagrama apresentado na figura 26 que, após o usuário informar as configurações de *proxy* através da interface Web, é criado um objeto *ProxyRequest* que contém os dados da configuração e é através do método *writeIpAndPort* da classe *ProxyConfControl* esses dados são escritos no arquivo de configuração.

Depois que as configurações de *proxy* forem definidas, pode-se alterar a configuração de *threads* da aplicação. A figura 27 mostra a interface para configuração de *threads*.

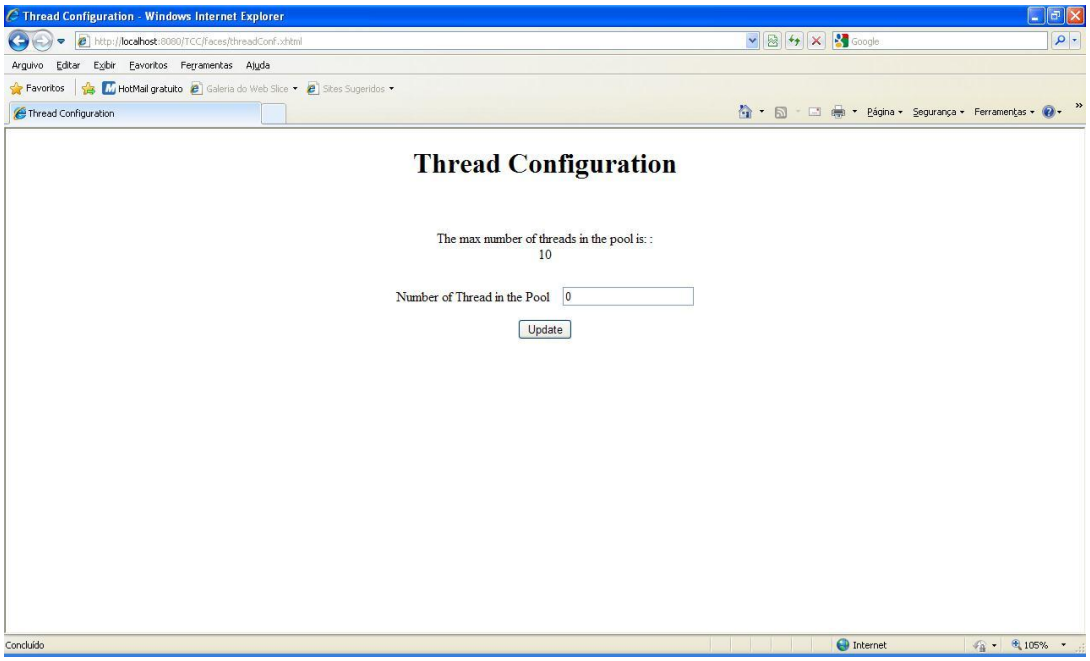

Figura 27 - Interface para definição da configuração de threads
Fonte: Autoria Própria

O usuário deve informar o número máximo de *threads* que podem ser executadas simultaneamente através da interface apresentada na figura 27. Para isso, basta ele informar o número desejado no campo de texto. Se nenhum valor for explicitado, será adotado como padrão o valor 10. A figura 28 mostra quais objetos são utilizados para definir as configurações de *threads*.

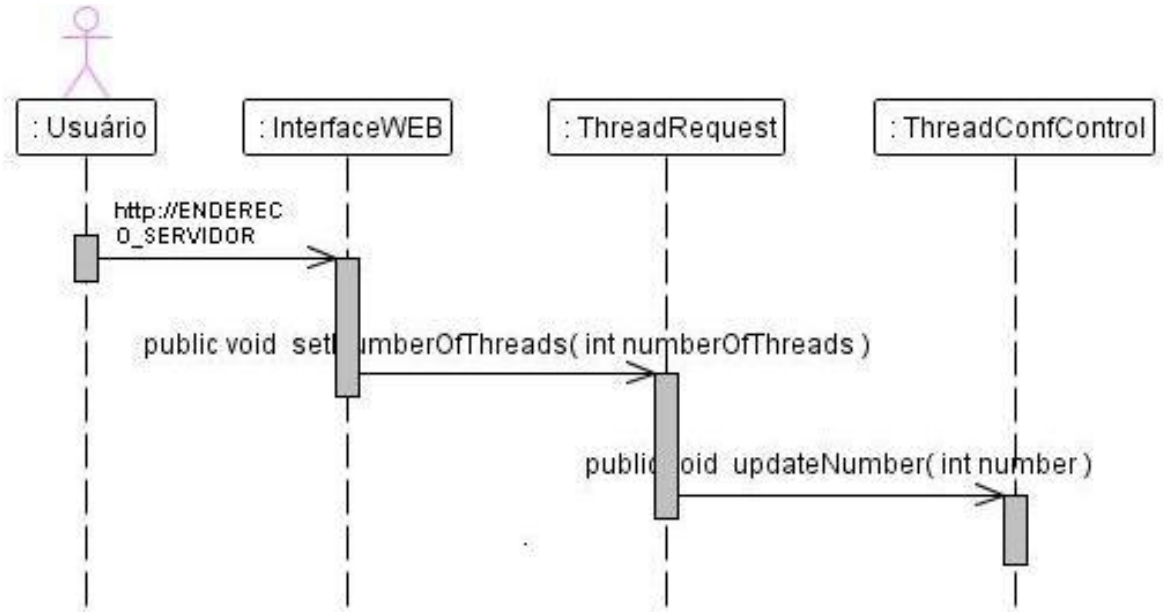

Figura 28 – Diagrama de Sequencia da Definição de Configuração de *Threads*
Fonte: Autoria Própria

Quando o usuário informa o número de *threads* através da interface Web, é criado um objeto do tipo *ThreadRequest* no qual é armazenado o número informado. Em seguida, o método *updateNumber* da classe *ThreadConfControl* é invocado para gravar o valor informado no arquivo de configuração.

Com todas as configurações de *proxy* e *threads* definidas, é possível cadastrar sítios de *e-commerce* através da interface apresentada na figura 29.

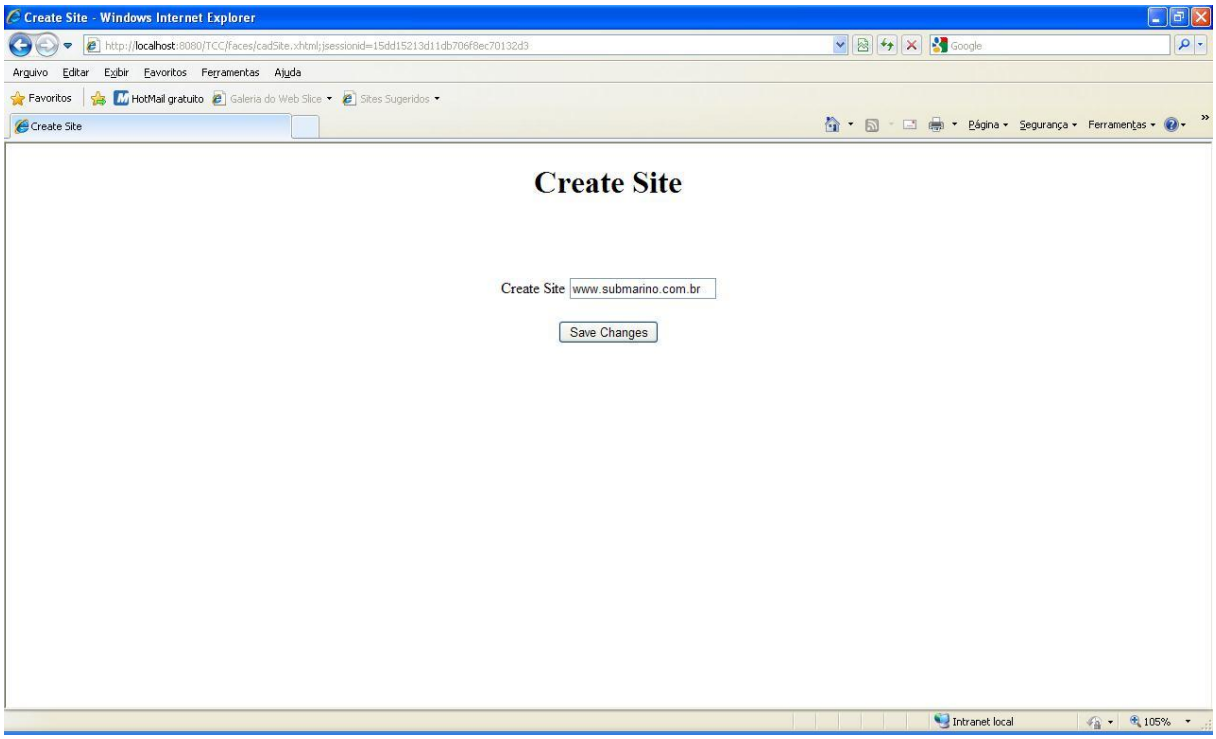

Figura 29 - Interface de cadastro de sítios de e-commerce
Fonte: Autoria Própria

Para cadastrar um novo sítio de *e-commerce*, basta digitar a URL no campo de texto e clicar no botão *Save Changes.*

As ações que acontecem quando o usuário clica no botão *Save Changes* são ilustradas na figura 30.

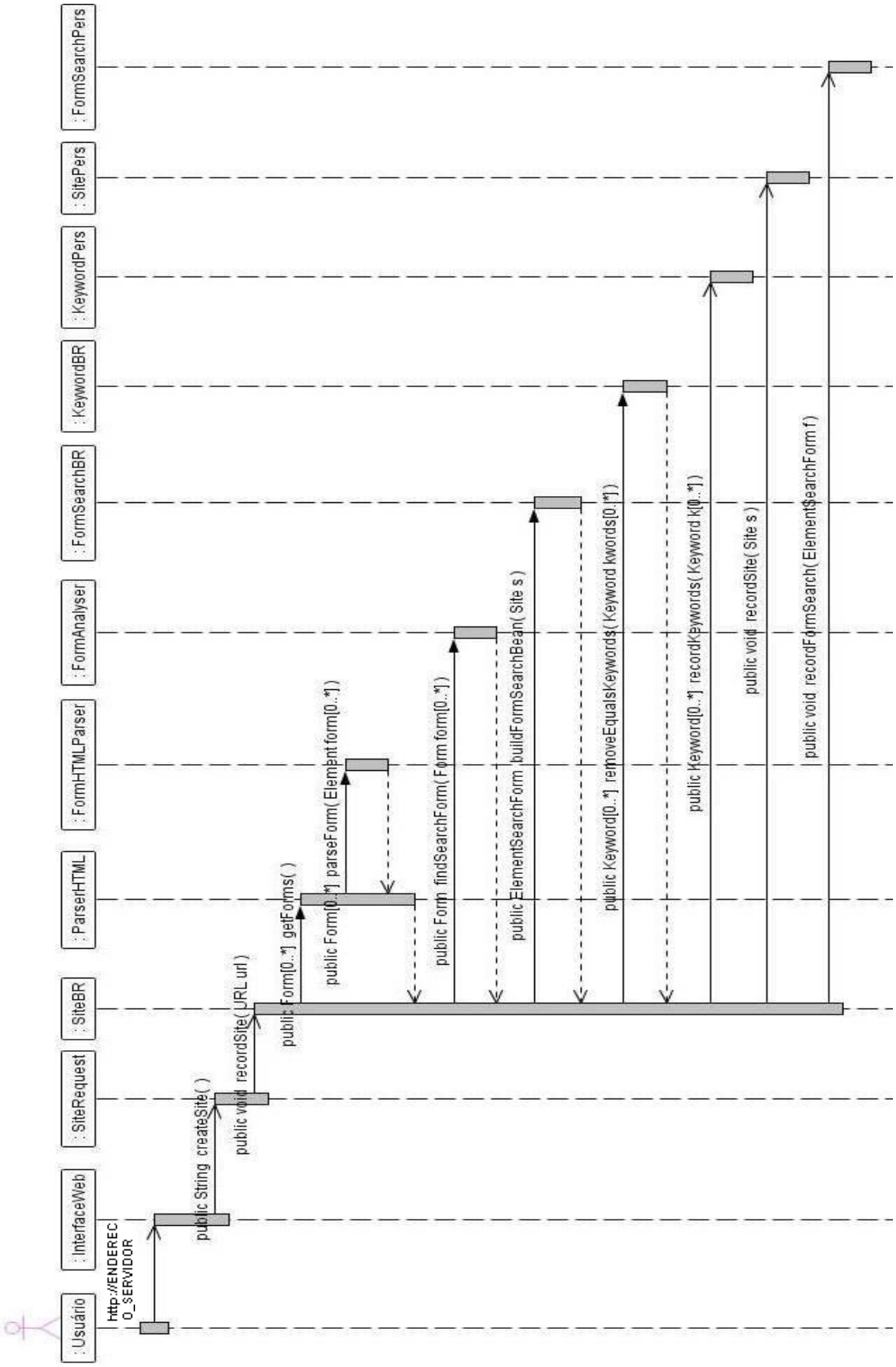

Figura 30 – Diagrama de Seqüencia do Cadastro de um Sítio
Fonte: Autoria Própria

Como é possível observar pelo diagrama, após o usuário informar qual sítio deseja cadastrar, um objeto do tipo *SiteRequest* é criado. Este objeto contém a URL do sítio informado pelo usuário e esta URL é passada como argumento para o método *recordSite* de um objeto do tipo *SiteBR*. Este objeto, utilizando o método *getForms* fornecido por um objeto do tipo *ParserHTML*, obtém os formulários do sítio.

Depois de ter obtido os formulários do sítio, o objeto do tipo *SiteBR*, usando o método *findSearchForm* de um objeto do tipo *FormAnalyzer*, encontra o formulário de busca. Após isso, o objeto do tipo *SiteBR* utiliza o método *buildFormSearchBean* de um objeto do tipo *FormSearchBR* para montar o *bean* do formulário de busca do sítio.

Muitas vezes um sítio utiliza uma mesma palavra-chave (*keyword*) mais de uma vez em sua meta-tag. Para evitar que haja valores repetidos gravados no banco de dados, o método *removeEqualKeywords* de um objeto do tipo *KeywordBR* é invocado para remover as repetições de palavras-chaves do sítio.

E, por fim, o sítio é gravado utilizando os métodos *recordKeywords*, *recordSite* e *recordFormSearch* de objetos dos tipos *KeywordPers, SitePers* e *FormSearchPers*, respectivamente.

4.1.2 Um Cliente *Desktop* Para o *Web Service*

Neste trabalho foi desenvolvido um cliente *desktop* para consumir o serviço disponibilizado através do *web service*. A interface do cliente é apresentada na figura 31.

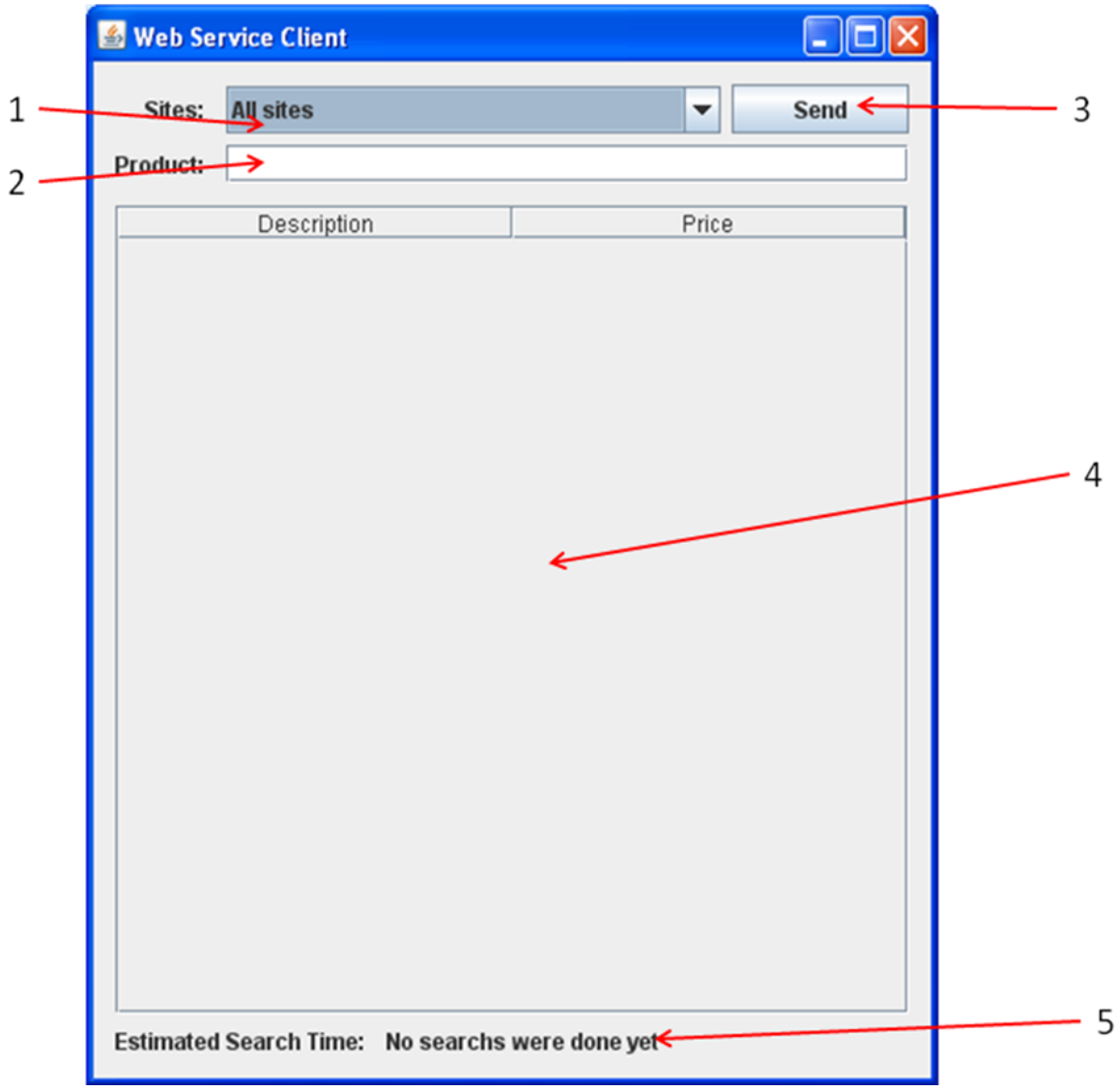

Figura 31 - Interface do Cliente
Fonte: Autoria Própria

O número 1 indica o componente no qual o usuário pode selecionar o sítio no qual deseja realizar a busca. A opção padrão é *All Sites* (todos os sítios). O número 2 permite ao usuário informar o nome do produto desejado. O 3 indica o botão que deve ser pressionado depois que o usuário informar o nome do produto e selecionar o sítio de *e-commerce* desejado. O 4 indica o local onde os resultados serão exibidos. E, por fim, o 5 indica o local onde são exibidos duas informações relevantes sobre a busca realizada: o tempo que demorou para a conclusão da busca e retorno do *web service* e o número de produtos encontrados.

A figura 32 mostra uma busca pelo produto Eric Clapton realizada através do *web service* no sítio do Submarino (SUBMARINO, 2010).

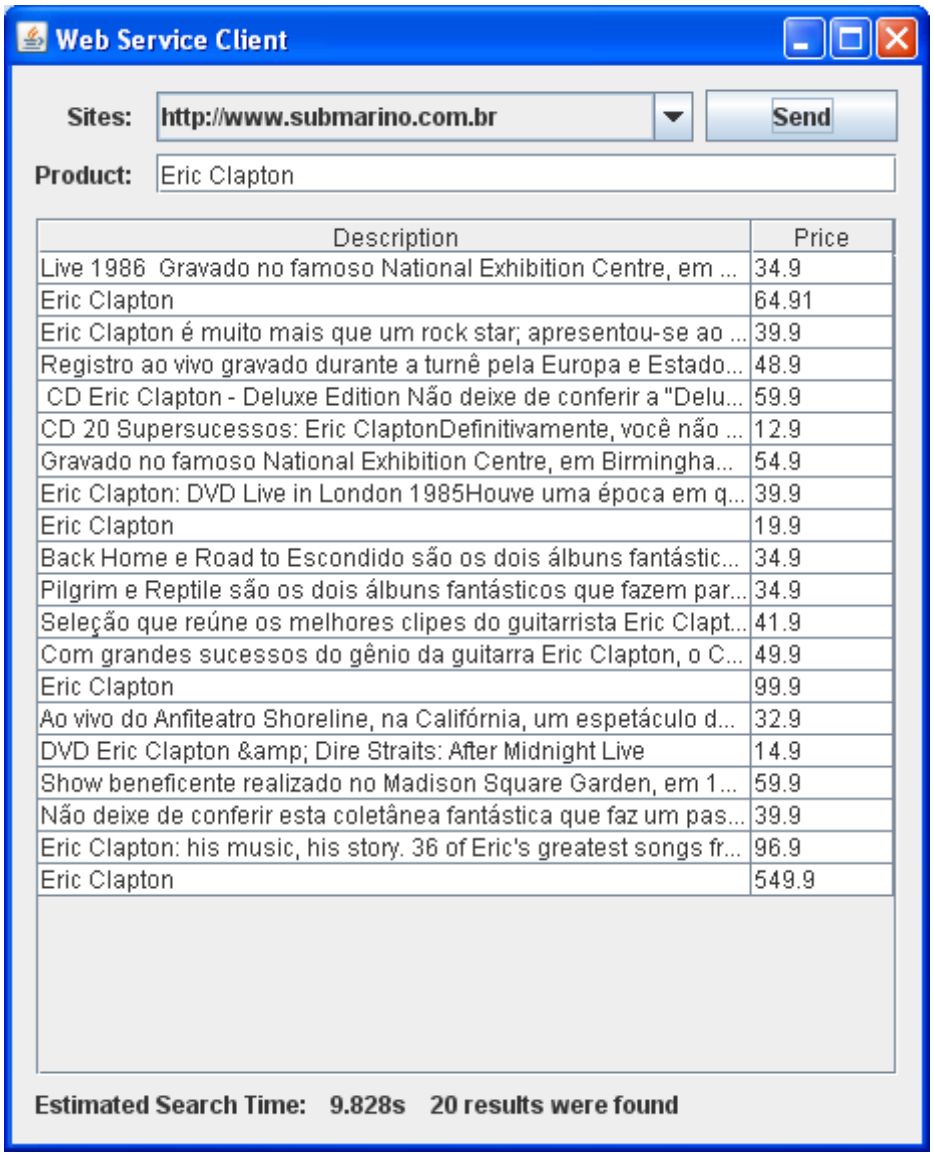

Figura 32 - Resultados retornados pelo web service em uma busca por Eric Clapton no Submarino
Fonte: Autoria Própria

O *web service* encontrou 20 produtos com o nome “Eric Clapton” no título ou na descrição do produto. Esta busca demorou 9.828s utilizando uma banda de 256 kbps.

Para fins de comparação, a figura 33 mostra alguns resultados retornados pelo Submarino quando é feita uma busca por Eric Clapton diretamente no sítio através de um navegador web.

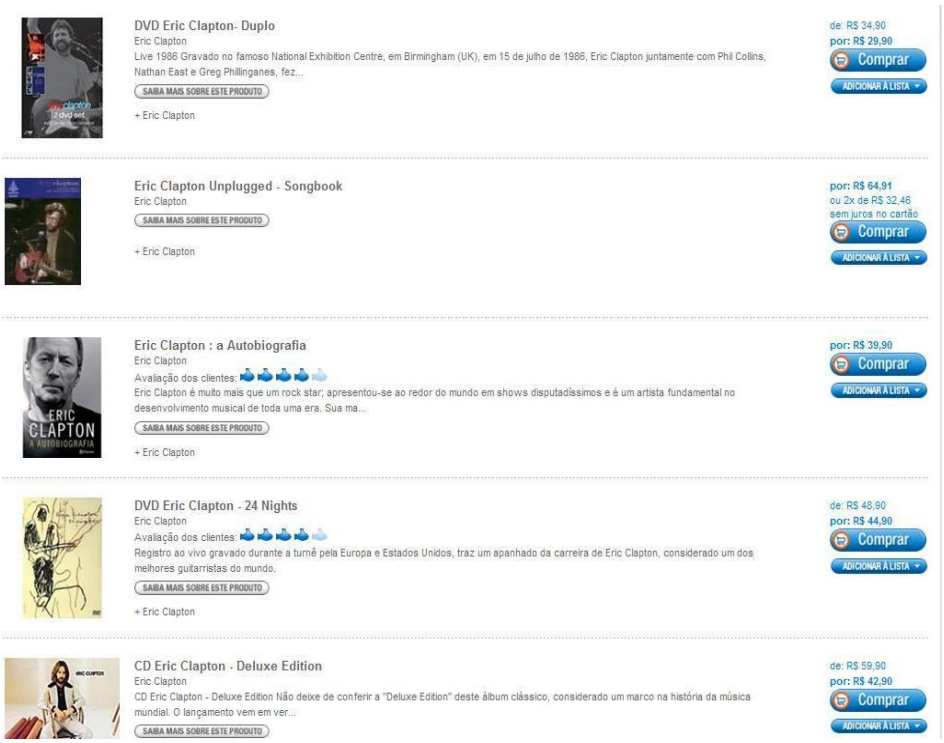

Figura 33 - Alguns resultados de uma busca por Eric Clapton no Submarino
Fonte: Autoria Própria

Como é possível observar, todos os produtos e preços exibidos na figura 33 estão entre os valores retornados pelo *web service*. O diagrama apresentado na figura 34 mostra a seqüência de ações que ocorrem quando o cliente realiza um busca no *web service*.

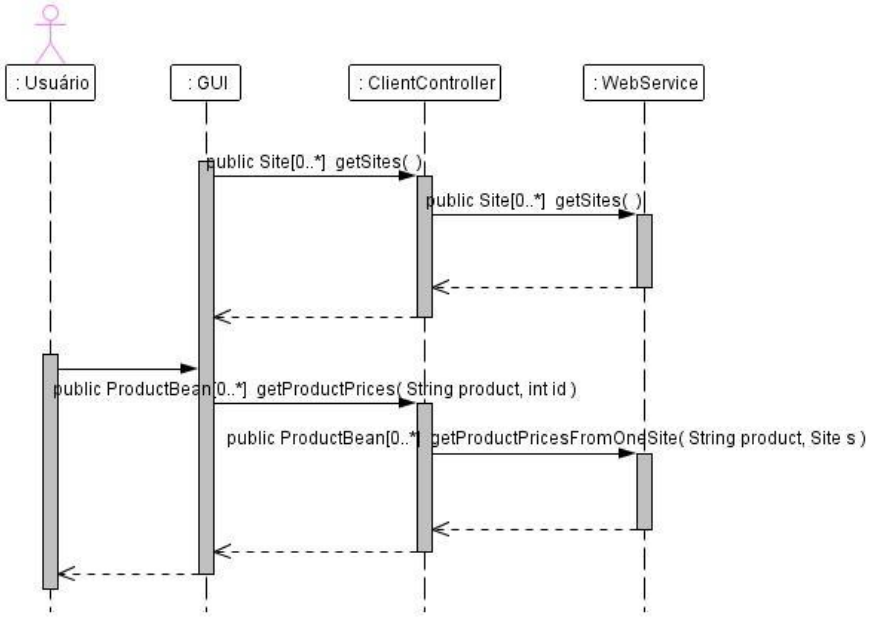

Figura 34 - Diagrama de seqüência do cliente
Fonte: Autoria Própria

Inicialmente o cliente invoca o método *getSites* do *web service* para obter os sítios cadastrados no banco de dados do serviço. Após isso, o usuário informa o nome do produto, seleciona o sítio no qual ele deseja que a busca seja realizada (submarino, no caso do exemplo citado anteriormente) e pressiona o botão *Send*. O cliente invoca o método *getProductPricesFromOneSite* do *web service* passando como parâmetro o nome do produto buscado e o sítio no qual o cliente deseja que a busca seja realizada.

O diagrama da figura 35 apresenta a seqüência de ações que o *web service* realiza quando o cliente faz a interação apresentada na figura 34.

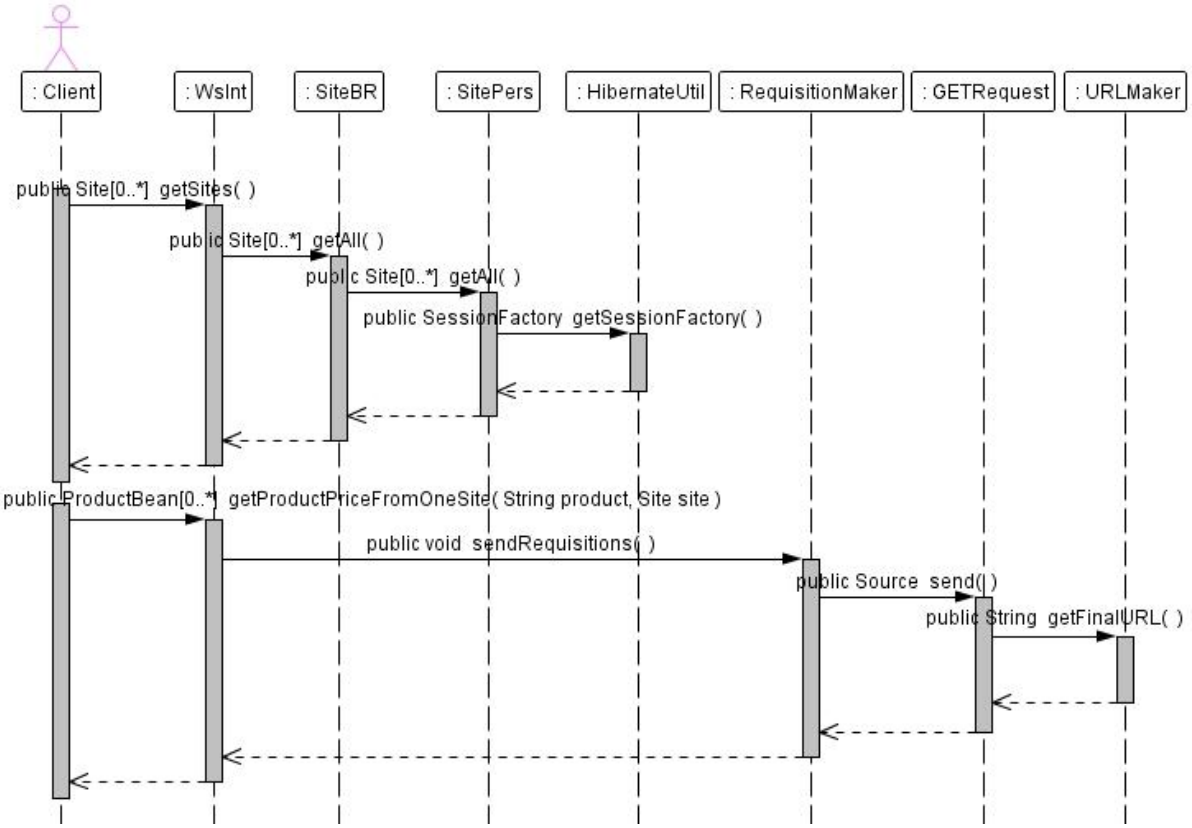

Figura 35 - Diagrama de seqüência do *web service*
Fonte: Autoria Própria

A primeira requisição que o cliente faz para o *web service* é a *getSites*. O *web service* recupera todos os sítios de *e-commerce* cadastrados no banco de dados através de objetos das classes *SiteBR e SitePers*.

Depois que o cliente obtém os sítios cadastrados no banco de dados, ele invoca o método *getProductPricesFromOneSite* do *web service*. Após isso, o método *sendRequisitions* de um objeto do tipo *RequisitionMaker* é chamado pelo *web service.*

O objeto do tipo *RequisitionMaker* envia requisições do tipo GET para o sítio de *e-commerce* especificado pelo cliente utilizando o método *send* de um objeto do tipo *GetRequest*. Objetos deste tipo utilizam o método *getFinalURL* de objetos do tipo *URLMaker* para obter o endereço para o qual a requisição deve ser enviada.

4.2 ANÁLISE DE DESEMPENHO

Esta seção faz uma pequena análise de desempenho do *web service* em função da banda disponível. Uma análise mais completa e elaborada depende de conhecimentos mais avançados sobre redes de computadores e matemática estatística, fugindo do escopo deste trabalho. Entretanto, uma análise de desempenho mais completa está entre as sugestões de trabalhos futuros discutidas na seção 5.1.

As buscas para análise foram realizadas em seis sítios, com o número máximo de *threads* de requisição definido para cinco e “O Senhor dos Anéis” como sendo o produto buscado. A figura 36 apresenta um gráfico do tempo estimado de busca em função da largura de banda disponível.

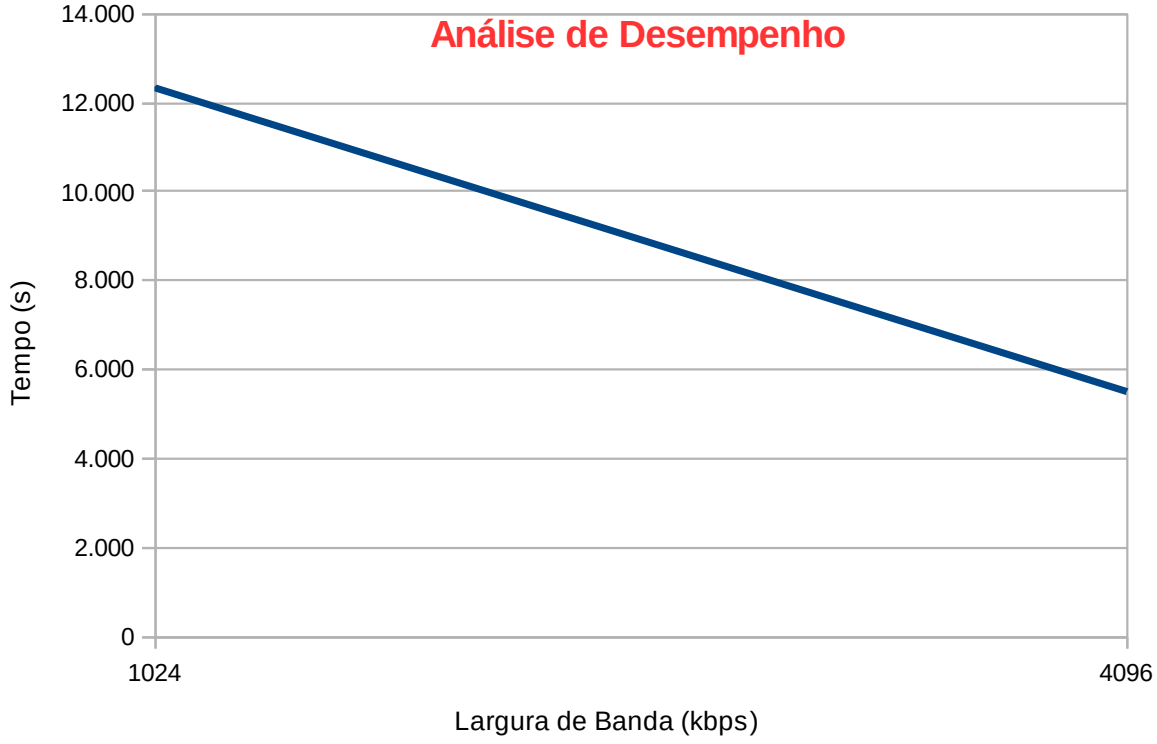

Figura 36 - Gráfico da Análise de Desempenho
Fonte: Autoria Própria

Os testes foram realizados em redes com bandas de 256, 1024 e 4098 kbps e os tempos estimados foram de 15.015, 12.328 e 5.458 segundos, respectivamente. E, conforme é possível observar pelo gráfico, quanto maior for a largura de banda da conexão com a internet disponível para o serviço, menor é o tempo que o *web service* demorará para obter os preços.

4.3 VANTAGENS DE UTILIZAR O *WEB SERVICE*

As principais vantagens de utilizar o *web service* desenvolvido neste trabalho são:

- Facilidade para encontrar os preços de um produto, pois não é necessário entrar em cada sítio e realizar a busca manualmente
- Promover a interoperabilidade entre plataformas heterogêneas. Assim sendo, o serviço desenvolvido neste trabalho pode ser utilizado por aplicações que rodem sobre qualquer plataforma, como, por exemplo, celulares e aplicações desenvolvidas com a plataforma .NET da Microsoft.
- A aplicação é capaz de realizar buscas em sítios internacionais, facilitando para aqueles que não dominam o idioma do sítio.
- Sítios de comparação de preços tradicionais, tais como BuscaPe (BUSCAPE, 2010), JaCotei (JACOTEI, 2010), entre outros, relacionam busca de produtos e preços somente de sítios de *e-commerce* com os quais tenham sido realizados acordos comerciais. Sendo assim, a busca de preços realizadas em sítios de comparação de preço é limitada. O *web service* desenvolvido possui a vantagem de não estar limitado a sítios com os quais tenham sido realizados qualquer tipo de acordo.

4.4 RESTRIÇÕES DO *WEB SERVICE* PROPOSTO

O *web service* proposto é capaz de fazer requisições somente do tipo HTTP-GET. A aplicação não funciona com sítios que utilizam o método HTTP-POST ou fazem suas buscas utilizando *javascript.*

Quando um novo sítio é cadastrado, a aplicação procura no HTML do sítio por um formulário que possua uma das palavras-chaves de busca. Se o sítio possuir um formulário de busca, porém, não possuir uma das palavras-chaves, o sítio não será cadastrado.

Outra restrição do *web service* desenvolvido neste trabalho é a maneira simplificada de realizar a busca da relação preço/produto em uma página HTML. A aplicação não consegue distinguir entre o nome do produto e a descrição em alguns casos, como mostra a figura 37.

Como é possível observar na figura 37, o *web service* não apenas retornou algumas descrições ao invés do nome do produto como também interpretou alguns trechos erroneamente dos htmls, como é o caso de resultados como “SOFTWARE” e preço “2.22121212122”.

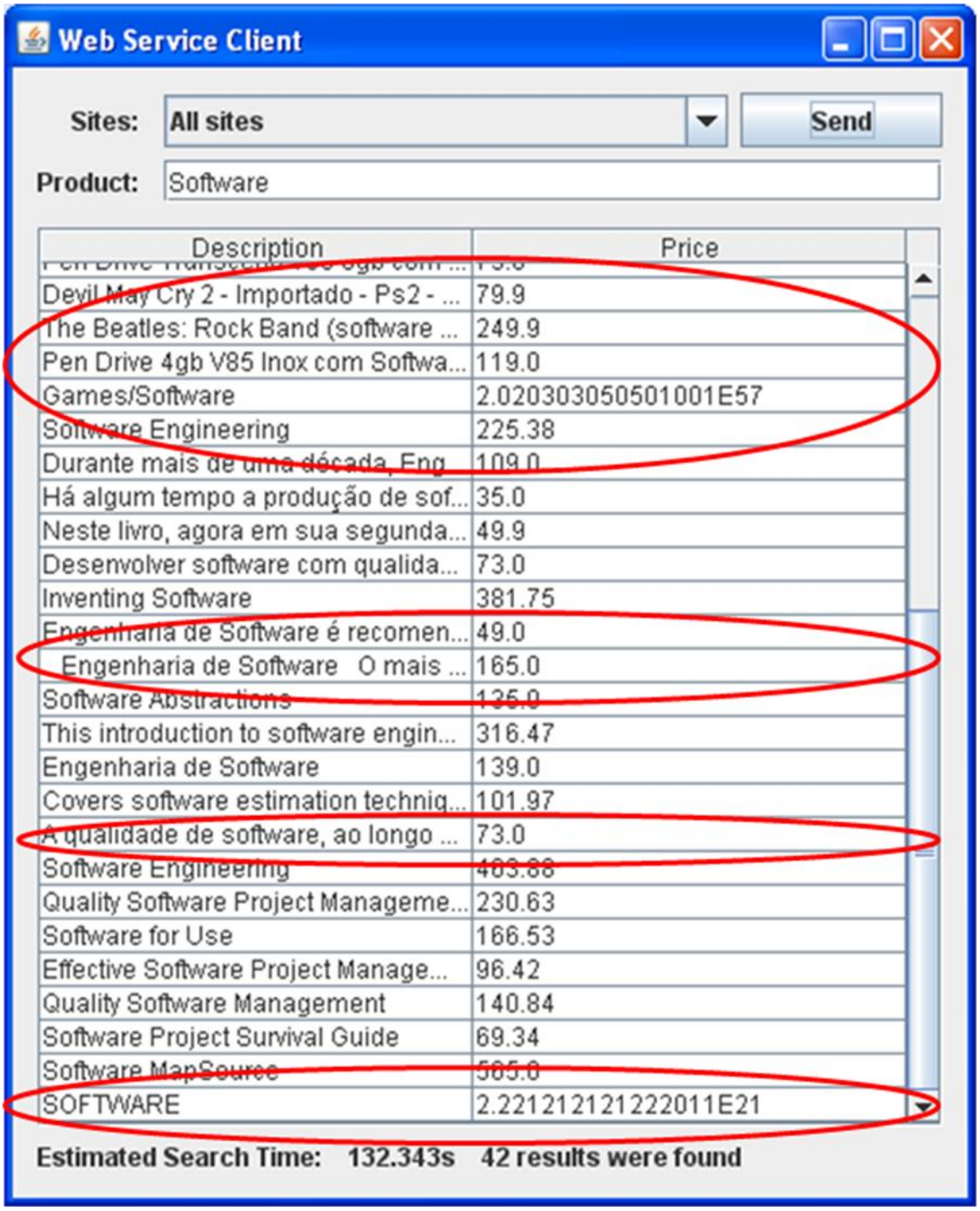

Figura 37 - Problemas no resultado
Fonte: Autoria Própria

5 CONCLUSÃO

O *web service* proposto é capaz de encontrar os preços de produtos a partir dos sítios de *e-commerce* cadastrados na base de dados. Desta forma, ele supri as necessidades do Grupo de Pesquisa em Engenharia de Software do Campus Ponta Grossa na implementação do método baseado nas decisões das empresas concorrentes.

Este *web service* contém código aberto e disponibiliza uma interface para acesso automatizado, diferente de sítios de comparação de preços tradicionais, tais como JaCotei (JACOTEI, 2010) e BuscaPe (BUSCAPE, 2010).

Quanto maior for a banda de conexão com a internet disponível para o serviço, menor será o tempo de busca do *web service* proposto para encontrar e retornar os resultados ao usuário.

Ele também possibilita que sejam definidas configurações de *proxy* e número máximo de *threads* no *pool*. Estas configurações são importantes para que o serviço possa ser utilizado dentro de um rede em que exista um servidor *proxy* e para um melhor desempenho de busca, respectivamente.

O serviço proposto não é capaz de realizar buscas em sítios que utilizem o método POST do HTTP e, também, não realiza busca baseado em funções *javascript*, sendo estes temas para trabalhos futuros.

Uma das limitações do trabalho proposto é a falta de precisão para encontrar as descrições corretas dos produtos pesquisados, como relatado na seção 4.4. Porém, isso não compromete seu uso pelo Grupo de Pesquisa.

5.1 TRABALHOS FUTUROS

A seguir são apresentadas algumas sugestões para trabalhos futuros:

- Uma análise de desempenho completa: esta analise permitiria mapear quais aspectos influenciam o desempenho do *web service,* viabilizando um estudo para otimização do serviço.
- Refatoração do código e aplicação de padrões de projeto: há diversas funcionalidades que podem desenvolvidas para o *web service.* Entretanto,

para facilitar a manutenção do código e a expansão para acomodar novas funcionalidades, é interessante realizar o *refactoring* do código.

- Aperfeiçoamento da lógica para encontrar os preços de um produto: apesar do *web service* encontrar as descrições dos produtos e seus preços com uma taxa de acertos bastante aceitável, é interessante aperfeiçoar a lógica empregada para realizar as buscas utilizando técnicas de inteligência artifical.
- Aplicação de mecanismos de segurança: o *web service* não realiza autenticação de clientes e nem envia mensagens criptografadas. O GPES não precisa que estes mecanismos sejam implementados, porém dependendo do contexto no qual o serviço for ser utilizado, a implementação destes mecanismos pode ser crucial.
- Criação de novas funcionalidades: há diversas funcionalidades que podem ser acrescidas ao *web service* desenvolvido. Entre elas, estão: encontrar, além dos preços, as condições de pagamento do produto, realizar buscas utilizando HTTP-POST, interpretar algumas funções *javascript* e manter um histórico das buscas realizadas e seus resultados.

REFERÊNCIAS

ABINADER, Jorge Abílio; LINS, Rafael Dueire. **Web Services em Java.** Brasport, 2006.

BONDFARO. Disponível em http://bondfaro.com.br. Acesso em jan-2010.

BOOTH, David; HAAS, Hugo; MCCABE, Francis; NEWCOMER, Eric; CHAMPION, Michael; FERRIS, Chris; ORCHARD, David. **Web Services Architecture.** Disponível em < http://www.w3.org/TR/ws-arch/ >. Acesso em 28 dez. 2009.

BUSCAPE. Disponível em http://buscape.com.br. Acesso em jan-2010.

CERAMI, Ethan. **Web Services Essentials Distributed Applications with XML-RPC, SOAP, UDDI & WSDL.** O'Reilly, 2002.

CHINNICI, Roberto; MOREAU, Jean-Jacques; RYMAN, Arthur; WEERAWARANA, Sanjiva. **Web Services Description Language (WSDL) Verison 2.0 Part I - Core Language**. Disponível em < http://www.w3.org/TR/WSDL20 >. Acesso em 10-jan-2010.

GLASSFISH. Disponível em http://glassfish.dev.java.net. Acesso em jan-2010.

HANSEN, Mark D. **SOA Using Java Web Services.** Prentice Hall, 2007.

HIBERNATE. Disponível em: http://hibernate.org. Acesso em jan-2010.

JACOTEI. Disponível em http://jacotei.com.br. Acesso em jan-2010.

JAX-WS. Disponível em: http://jax-ws.dev.java.net. Acesso em jan-2010.

JSF. Disponível em: http://java.sun.com/javaee/javaserverfaces. Acesso em jan-2010

MAHMOUD, Qusay H. Service Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI), **Sun Developer Network**, abril 2005. Disponível em < http://java.sun.com/developer/technicalArticles/WebServices/soa/ >. Acesso em 07 mar. 2010.

METRO. Disponível em: http://metro.dev.java.net. Acesso em jan-2010.

MYSQL. Disponível em: http://wb.mysql.com. Acesso em jan-2010.

NAKAGAWA, M. **ABC – Custeio baseado em atividades**. São Paulo: Atlas, 1995.

NETBEANS. Disponível em: http://netbeans.org. Acesso em jan-2010.

NASCIMENTO, D. T. do; VARTANIAN, G. H. **O método pleno** – uma abordagem conceitual. Revista CRC-SP nº09 Set/1999.

OLIVEIRA, Rafael R. de; CREMA, Rudy J. C. **Definição dos pontos de estabilidade, em nível de requisitos, no domínio de preço de venda.** 2009. 192 f. Trabalho de Conclusão de Curso (Graduação) – Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2009.

SANTOS, Joel José dos. **Formação de preços e do lucro**. 3 ed. São Paulo: Atlas, 1991.

SARDINHA, J. C. **Formação de Preços: A Arte do Negócio**. São Paulo: Ed. Makron Books, 1995.

SEBRAEPR. Disponível em http://sebraepr.com.br. Acesso em jan-2010.

STREETPRICE. Disponível em http://streetprice.com. Acesso em jan-2010.

SUBMARINO. Disponível em http://submarino.com.br. Acesso em jan-2010.

APÊNDICE A – Arquivo WSDL gerado

Não é necessário se preocupar com a elaboração manual ou interpretação do arquivo WSDL durante o desenvolvimento de um *web service* ou de um cliente porque as ferramentas de desenvolvimento e APIs geram automaticamente tanto o arquivo WSDL como as classes clientes que interagem com o serviço.

Sendo assim, o WSDL do *web service* proposto será apresentado para uma melhor compreensão da interface do serviço criada.

Neste apêndice, o WSDL foi divido em cinco partes e a primeira é exibida na figura 38.

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!--
3   Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is
4    JAX-WS RI 2.2-hudson-752-.
5    -->
6  <!--
7   Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is
8    JAX-WS RI 2.2-hudson-752-.
9    -->
10 <definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
11     200401-wss-wssecurity-utility-1.0.xsd"
12     xmlns:wsp="http://www.w3.org/ns/ws-policy"
13     xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
14     xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
15     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
16     xmlns:tns="http://ws/"
17     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
18     xmlns="http://schemas.xmlsoap.org/wsdl/"
19     targetNamespace="http://ws/" name="WsIntService">
```

Figura 38 - Primeira Parte do WSDL
Fonte: Autoria Própria

A linha 1 é uma declaração típica de arquivos XMLs. Os comentários das linhas 2 a 9 informam que o arquivo foi gerado e publicado pela API JAX-WS, que é a referencia de implementação (RI) da Sun. Nas linhas 10 a 19 são declarados alguns *namespaces* do arquivo.

O quadro 1 apresenta os principais *namespaces* utilizados em arquivos WSDL.

Prefixo	Namespace
wsdl	"http://www.w3.org/ns/wsdl"
wsdli	"http://www.w3.org/ns/wsdl-instance"
wsdlx	"http://www.w3.org/ns/wsdl-extensions"
wrpc	"http://www.w3.org/ns/wsdl/rpc"
wsoap	"http://www.w3.org/ns/wsdl/soap"
whttp	"http://www.w3.org/ns/wsdl/http"
xs	"http://www.w3.org/2001/XMLSchema"
xsi	"http://www.w3.org/2001/XMLSchema-Instance"

Quadro 1 - Principais namespaces utilizados
Fonte: Adaptado de Chinnici *et. al.* (2007)

A figura 39 apresenta a segunda parte do WSDL.

```
20 <types>
21 <xsd:schema>
22   <xsd:import namespace="http://ws/"
23     schemaLocation="http://localhost:8080/TCC/WsIntService?xsd=1" />
24     </xsd:schema>
25     </types>
```

Figura 39 - Segunda Parte do WSDL
Fonte: Autoria Própria

A linha 22 importa o *namespace* “http://ws/” declarado no XML Schema do WSDL e definido na classe *WsInt*.

A figura 40 apresenta a terceira parte do arquivo WSDL. Nesta parte são declaradas as mensagens que o *web service* pode receber ou enviar. Para cada método disponibilizado pelo serviço são declaradas duas mensagens no WSDL, uma para a requisição feita pelo cliente e outra para a resposta enviada pelo *web service*.

Na figura 41 são declaradas as operações que utilizam as mensagens apresentadas na figura 40.

```
26 <message name="getProductPrice">
27   <part name="parameters" element="tns:getProductPrice" />
28     </message>
29 <message name="getProductPriceResponse">
30   <part name="parameters" element="tns:getProductPriceResponse" />
31     </message>
32 <message name="getProductPriceFromOneSite">
33   <part name="parameters" element="tns:getProductPriceFromOneSite" />
34     </message>
35 <message name="getProductPriceFromOneSiteResponse">
36   <part name="parameters"
37     element="tns:getProductPriceFromOneSiteResponse" />
38     </message>
39 <message name="getSites">
40   <part name="parameters" element="tns:getSites" />
41     </message>
42 <message name="getSitesResponse">
43   <part name="parameters" element="tns:getSitesResponse" />
44     </message>
```

Figura 40 - Terceira Parte do WSDL
Fonte: Autoria Própria

```
45 <portType name="WsInt">
46 <operation name="getProductPrice">
47   <input wsam:Action="http://ws/WsInt/getProductPriceRequest"
48     message="tns:getProductPrice" />
49   <output wsam:Action="http://ws/WsInt/getProductPriceResponse"
50     message="tns:getProductPriceResponse" />
51     </operation>
52 <operation name="getProductPriceFromOneSite">
53   <input
54     wsam:Action="http://ws/WsInt/getProductPriceFromOneSiteRequest"
55     message="tns:getProductPriceFromOneSite" />
56   <output
57     wsam:Action="http://ws/WsInt/getProductPriceFromOneSiteResponse"
58     message="tns:getProductPriceFromOneSiteResponse" />
59     </operation>
60 <operation name="getSites">
61   <input wsam:Action="http://ws/WsInt/getSitesRequest"
62     message="tns:getSites" />
63   <output wsam:Action="http://ws/WsInt/getSitesResponse"
64     message="tns:getSitesResponse" />
65     </operation>
66     </portType>
```

Figura 41 - Quarta Parte do WSDL
Fonte: Autoria Própria

O elemento *portType* declara qual a classe que está disponível como serviço. Neste caso, é a *WsInt*. Em seguida, na linha 47, é declarada a operação *gerProductPrice*, que espera como entrada a mensagem "*tns:getProductPrice"* e

retorna a mensagem *“tns:getProductPriceResponse”* como resposta. As operações declaradas nas linhas 52 e 60 seguem a mesma idéia.

A figura 42 apresenta a quinta e última parte do WSDL.

```
67 <binding name="WsIntPortBinding" type="tns:WsInt">
68   <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
69      style="document" />
70 <operation name="getProductPrice">
71   <soap:operation soapAction="" />
72 <input>
73   <soap:body use="literal" />
74     </input>
75 <output>
76   <soap:body use="literal" />
77     </output>
78     </operation>
79 <operation name="getProductPriceFromOneSite">
80   <soap:operation soapAction="" />
81 <input>
82   <soap:body use="literal" />
83     </input>
84 <output>
85   <soap:body use="literal" />
86     </output>
87     </operation>
88 <operation name="getSites">
89   <soap:operation soapAction="" />
90 <input>
91   <soap:body use="literal" />
92     </input>
93 <output>
94   <soap:body use="literal" />
95     </output>
96     </operation>
97     </binding>
98 <service name="WsIntService">
99 <port name="WsIntPort" binding="tns:WsIntPortBinding">
100   <soap:address location="http://localhost:8080/TCC/WsIntService" />
101     </port>
102     </service>
103     </definitions>
```

Figura 42 - Quinta Parte do WSDL
Fonte: Autoria Própria

Nesta parte do WSDL são declaradas informações relacionadas à conexão entre servidor e cliente. Na linha 68 é definido o SOAP sobre HTTP como protocolos de transporte e nas linhas seguintes são especificados detalhes sobre a configuração do SOAP para cada operação disponibilizada. Os detalhes destas especificações fogem do escopo deste trabalho.

Nas linhas 99 e 100 são declarados, respectivamente, o nome do serviço e seu endereço.

ANEXO A – Email de Resposta do Sítio JaCotei

Abaixo é apresentado o email de resposta do JaCotei (JACOTEI, 2010) afirmando que não é permitido que aplicações obtenham e utilizem informações relacionadas no sítio.

Bom dia, Prezado Claudinei

Agradecemos a visita ao JáCotei e a informação enviada

Atualmente não temos nada automatico que possa ser utilizado e nosso site também não permiti rodar programas para capturar preços.

Mas temos interesse , e queremos conhecer o projeto , teria alguma possibilidade de marcarmos uma visita aqui na empresa.

atenciosamente.

Dircely Coelho
Atendimento
Tel./Fax.: (11) 3288-3151 | R. 226
Skype.: dircely.coelho.jacotei