Arquitetura de um *Web Service* para Busca de Preços de Produtos na Internet

Claudinei Rodrigues Junior[1], Simone Nasser Matos[1]

[1]Coordenação de Informática – Universidade Tecnológica Federal do Paraná (UTFPR)
84.016-210 – Ponta Grossa – PR – Brasil

claudineirdj@gmail.com, snasser@utfpr.edu.br

Abstract. *Currently there is not free software that implements several pricing methods for manager's use. Due to 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. Among those methods, one is known as pricing method based on the bidder decisions, which on depends on pricing method used in the market. There is not at the internet a free and open source service that searches selling prices of a product so, in order to help 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 Services, Pricing Methods, Information Systems, Software Engineering*

Resumo. *Atualmente não há um software que implemente diversos métodos de formação de preço de venda e que seja gratuito para que os gestores utilizarem. 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 diversos métodos de formação de preço de venda. Entre os métodos, há um conhecido como método baseado nas decisões das empresas concorrentes, que 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 Services, Métodos de Formação de Preço de Venda, Sistemas de Informação, Engenharia de Software.*

1. Introdução

A competitividade do mercado é intensa e 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 1991] e [Sardinha 1995], 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) [GPES 2010], 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.

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.

O presente artigo apresenta a arquitetura do *web service* desenvolvido. Vale ressaltar que todas as tecnologias empregadas para desenvolver o *web service* são gratuitas e *open-source*: Java [JAVA 2010], Hibernate [HIBERNATE 2010], MySQL [MYSQL 2010], Jericho HTML Parser [JERICHO 2010] e WSIT [WSIT 2010].

A seção 2 discute a importância da boa formação de preço de venda e quais são os aspectos que devem ser considerados no momento de tomar esta decisão. A seção 3 apresenta os conceitos básicos relacionados a tecnologia de *web services*.A seção 4 discorre sobre a arquitetura do *web service* desenvolvido para solucionar o problema do GPES. A seção 5 discute as características do *web service* desenvolvido. E, por fim, a seção 6 conclui o trabalho, discutindo os resultados obtidos, as restrições do serviço e fazendo algumas sugestões para trabalhos futuros.

2. 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 and 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 desde o crescimento da empresa até sua estabilidade econômico-financeira.

Segundo [Oliveira and 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 1991].

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* desenvolvido e apresentado neste artigo auxilia a análise da concorrência, obtendo os preços de mercados praticados pela concorrência, pois, 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 simultaneamente ou a realizarem os cálculos manualmente.

Devido a este problema, o GPES 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 and 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, entre outros parâmetros de entrada, os preços de venda praticados pelo mercado [SARDINHA 1995]. Por isso, surgiu a necessidade de se construir um *web service* para busca de preço de venda de um determinado produto.

3. *Web Services*

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 (*Single Object Acess Protocol*) *Technologies*). Estas tecnologias unidas a XML compõem a estrutura básica dos *Web Services*.

3.1 XML

A *Extensible Markup Language (XML)* é o padrão adotado para o transporte de dados [ABINADER and 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 heterogêneos permitida pelos *Web Services*.

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

3.2 WSDL

A *Web Services Descriptor Language* (WSDL) é a linguagem utilizada para criar uma auto-descrição pública do serviço [ABINADER and 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.

3.3 UDDI

O *Universal Description Discovery and Integration* (UDDI) é o diretório onde as informações sobre os *Web Services* podem ser armazenadas [ABINADER and 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.

3.4 SOAP

O *Single Object Acess Protocol* (SOAP) é o protocolo adotado pela maioria dos *Web Services* para troca de mensagens [ABINADER and 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.

4. Arquitetura do *Web Service*

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

4.1 Arquitetura de Alto Nível

A arquitetura de alto nível do *web service* desenvolvido é representada utilizando um diagrama de pacotes, apresentado na figura 1.

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*.

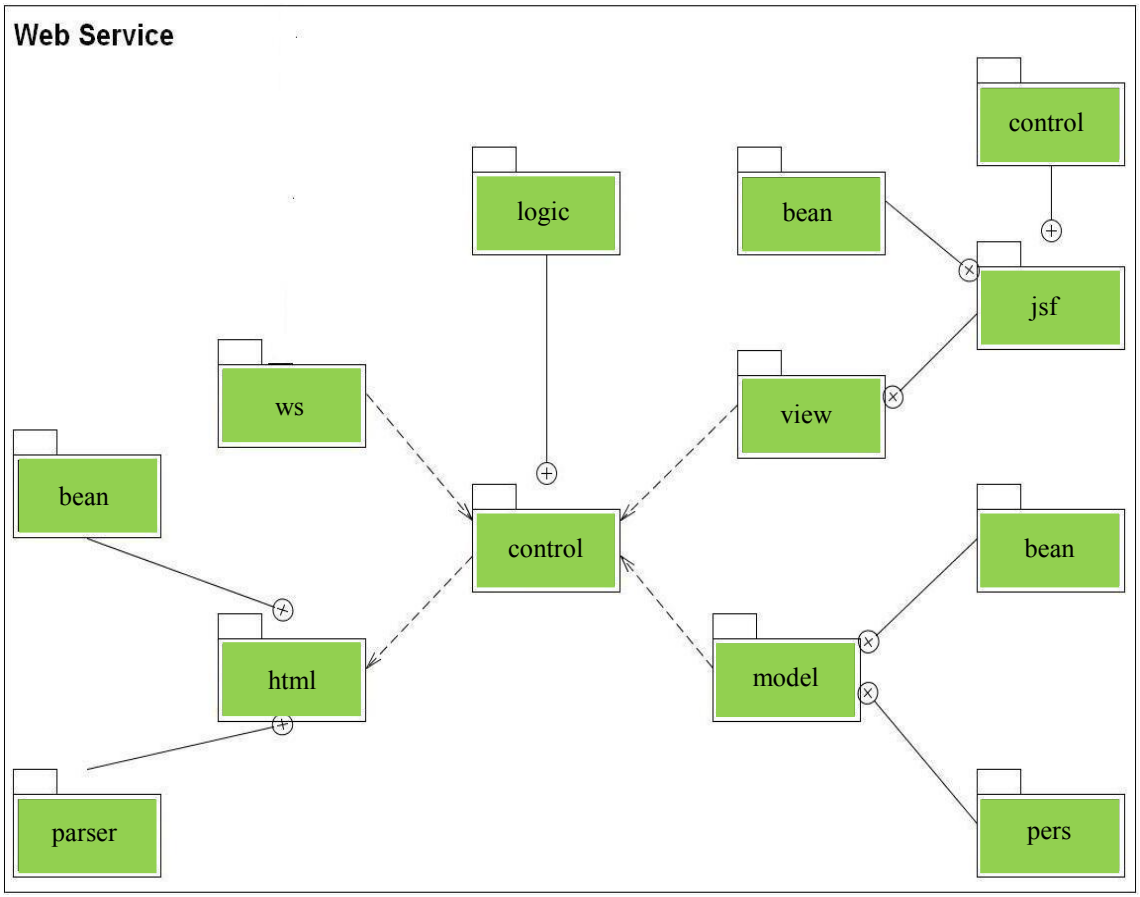

Figura 1 – Diagrama de Pacotes do *Web Service*

4.2 Arquitetura de Baixo Nível

A seguir será apresentada a arquitetura de baixo nível de alguns pacotes do *web service* desenvolvido.

4.2.1 Diagrama de Classes do Pacote *html*

O pacote *html* contém uma única classe, *EntityConverter*, e uma interface, *HTMLConstant*, ilustradas na Figura 2.

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*.

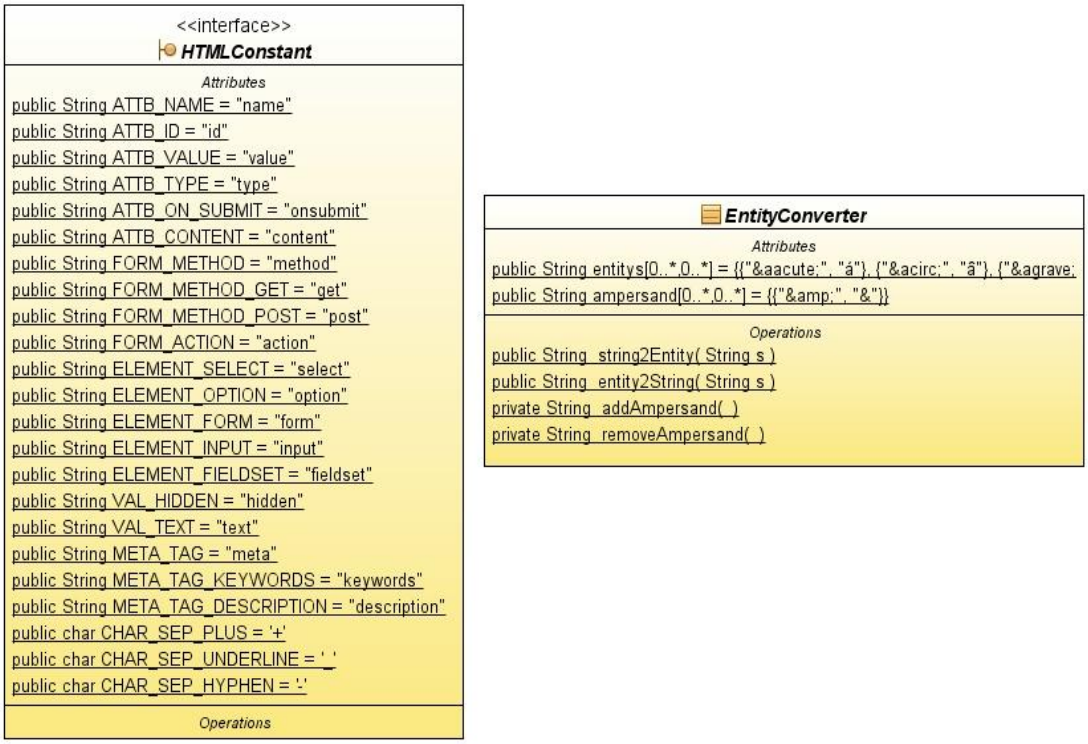

Figura 2 – Diagrama de Classes do Pacote *html*

4.2.2 Diagrama de Classes do Pacote *html.parser*

As classes do pacote *html.parser* estão ilustradas na Figura 3. 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.

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*.

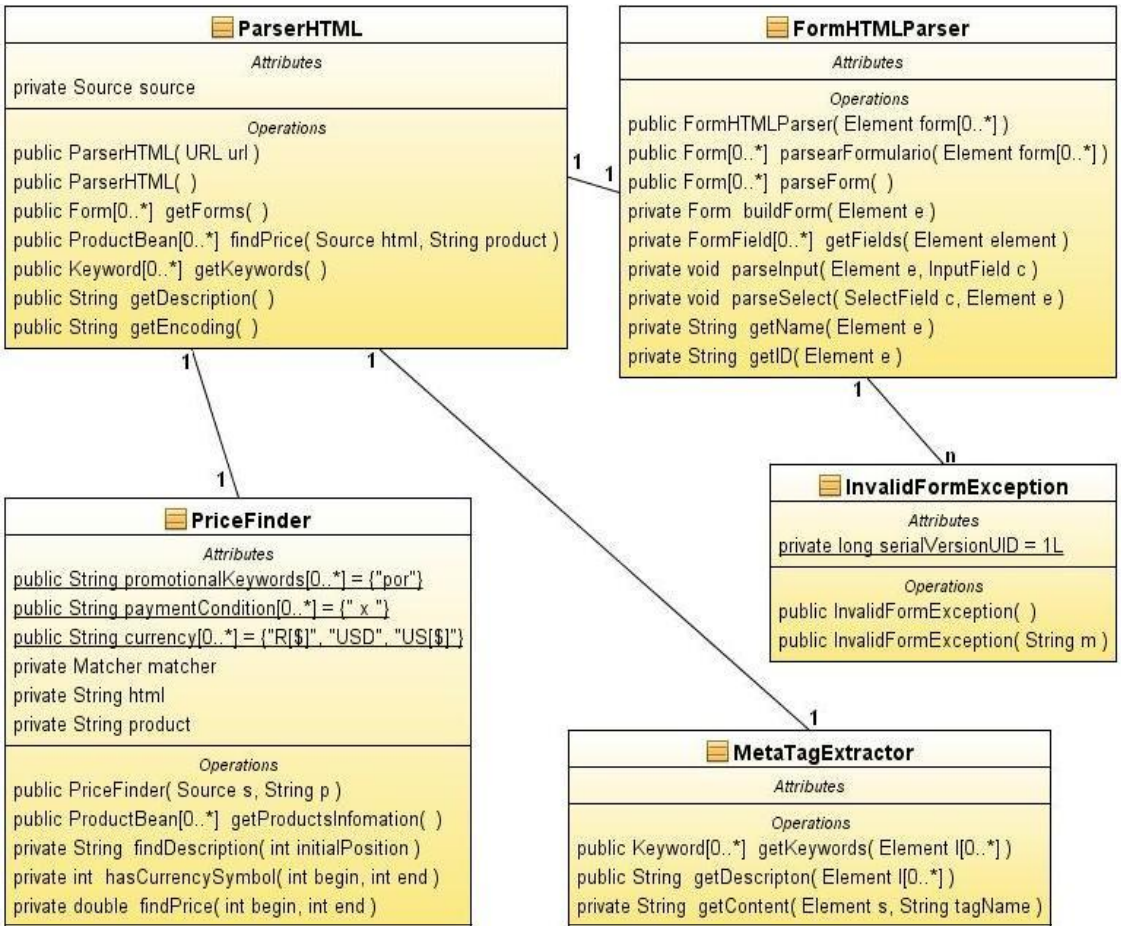

Figura 3 – Diagrama de Classes do Pacote *html.parser*

4.2.3 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 4.

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.

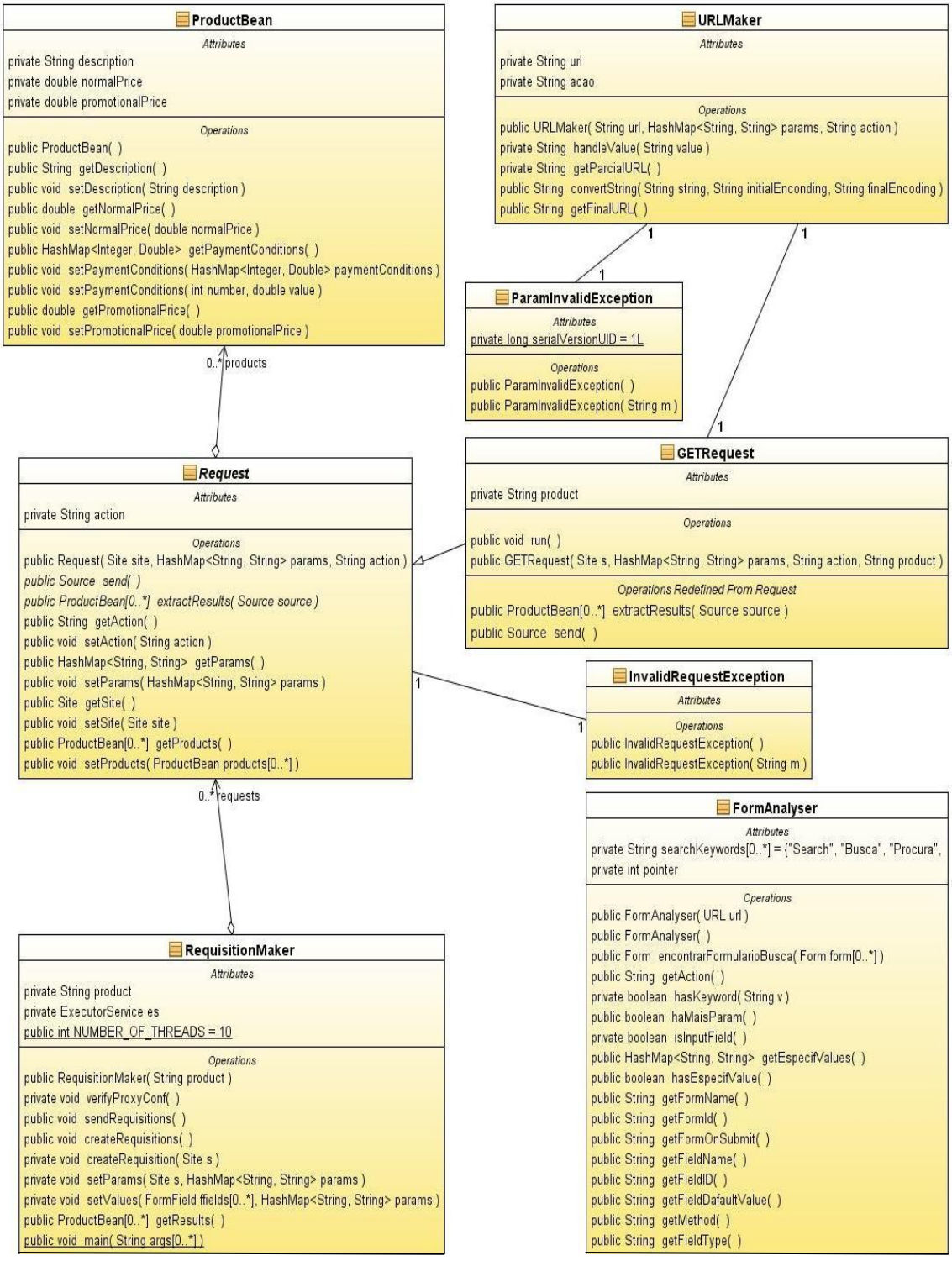

Figura 4 – Diagrama de Classes do Pacote *control.logic*

4.2.4 Diagrama de Classes do Pacote *ws*

A figura 5 apresenta a classe *WsInt*, localiza no pacote *ws*. É a partir desta classe que é gerada a interface do serviço.

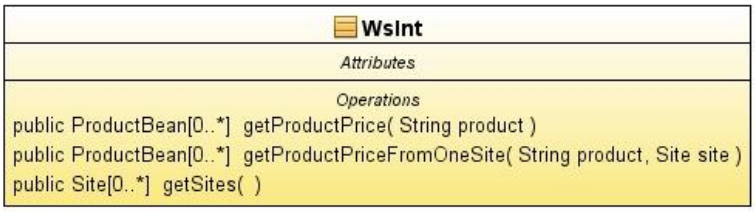

Figura 5 – Diagrama de Classes do Pacote *ws*

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.

4.2.5 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 6.

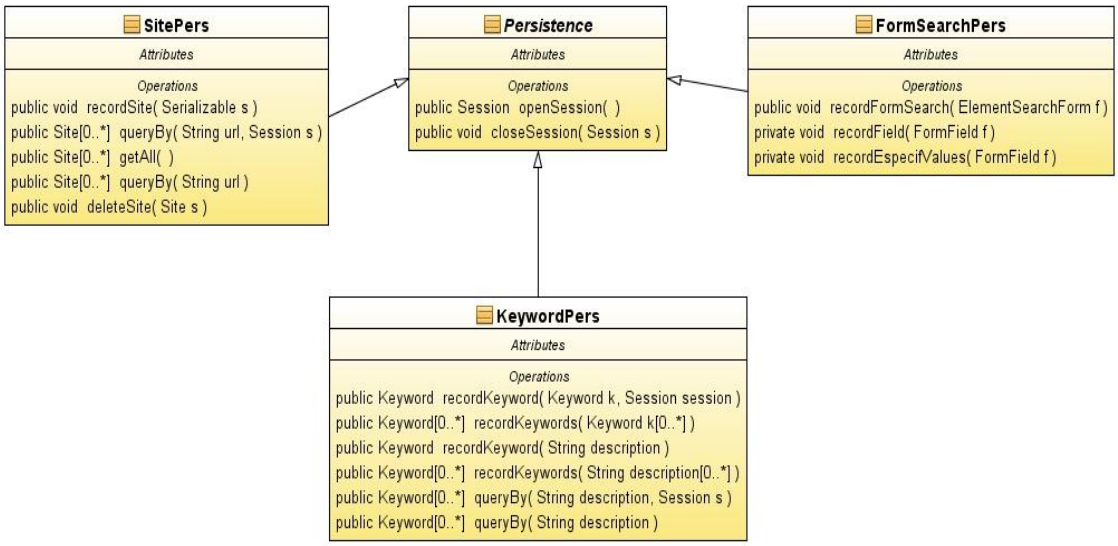

Figura 6 – Diagrama de Classes do Pacote *model.pers*

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.

5. Características do *Web Service* Desenvolvido

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 API WSIT [WSIT 2010]. A API JAX-WS [JAX-WS 2010] é o núcleo da WSIT e fornece os métodos básicos para a criação e disponibilização de *web services* enquanto as extensões providas pela WSIT oferecem 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 7.

Observando a figura 7, 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.

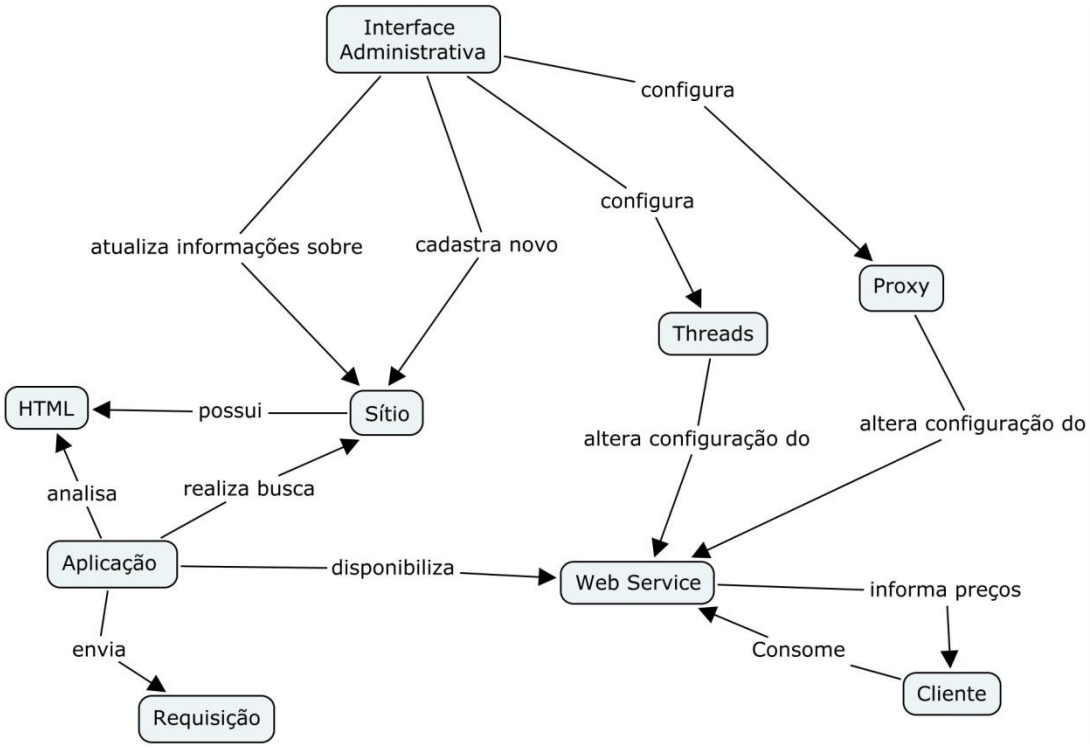

Figura 7 – Modelo de Funcionamento do *Web Service*

6. Conclusões

A figura 8 mostra um cliente *desktop* consumindo o serviço.

Como é possível observar pela figura 8, o *web service* desenvolvido é capaz de encontrar os preços de produtos nos sítios de *e-commerce* cadastrados na base de dados. Desta forma, ele supriu 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 e BuscaPe.

As tecnologias empregadas no desenvolvimento do *web service* facilitaram consideravelmente a implementação do serviço e reduziram drasticamente o tempo de desenvolvimento. Vale ressaltar que foi fácil empregar estas ferramentas, não demandando muito tempo o aprendizado das mesmas.

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

O *web service* e o código fonte do serviço podem ser baixados gratuitamente do sítio do GPES (2010).

6.1 Trabalhos Futuros

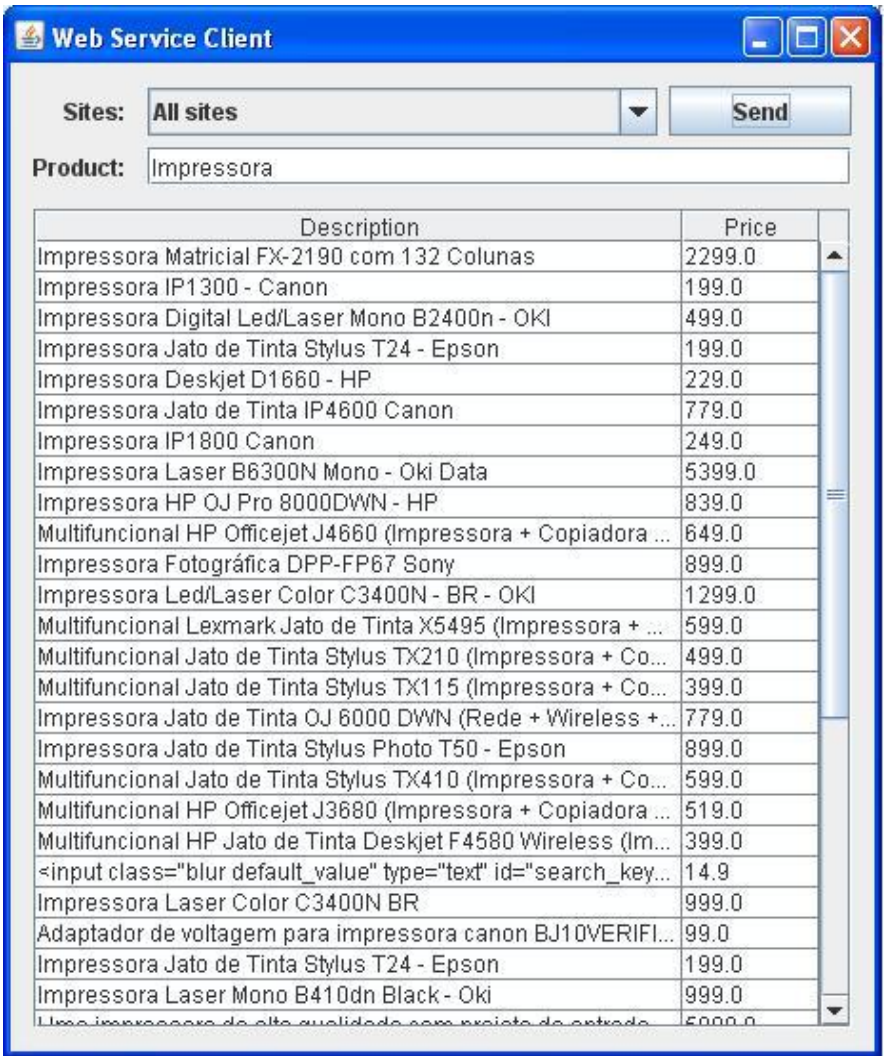

Figura 8 – Um cliente *desktop* consumindo o serviço

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.

7. Referências

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

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

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

GPES. Disponível em http://200.134.81.19:8080. Acesso em fev-2010.

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

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

JAVA. Disponível em http://java.sun.com. Acesso em jan-2010.

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

JERICHO. Disponível em http://jerichohtml.sourceforge.net. Acesso em jan-2010.

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

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

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

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. (1991), Formação de preços e do lucro, Atlas, 3th edição.

SARDINHA, J. C. (1995), Formação de Preços: A Arte do Negócio, Makron Books.

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

WSIT. Disponível em http://wsit.dev.java.net. Acesso em jan-2010.