Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

TAMIRES APARECIDA EVANGELISTA

SISTEMA DE MÚLTIPLOS CÁLCULOS PARA CORRETORAS DE SEGURO

Assis

2012

TAMIRES APARECIDA EVANGELISTA

SISTEMA DE MÚLTIPLOS CÁLCULOS PARA CORRETORAS DE SEGURO

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do Curso de Graduação.

ORIENTADOR: Profº Ms. Osmar Aparecida Machado

ÁREA DE CONCENTRAÇÃO: Informática

Assis

2012

FICHA CATALOGRÁFICA

EVANGELISTA, Tamires Aparecida

Sistema de Múltiplos Cálculos para Corretoras de Seguro/ Tamires Aparecida Evangelista. Fundação Educacional do Município de Assis – FEMA – Assis, 2012.

50 p.

Orientador: Profº Ms. Osmar Aparecido Machado

Trabalho de Conclusão de Curso – Instituto Municipal de Ensino Superior de Assis – IMESA.

1. Web Services. 2. Seguro. 3. SOAP.

CCD: 001.6

Biblioteca da FEMA

SISTEMA DE MÚLTIPLOS CÁLCULOS PARA CORRETORAS DE SEGURO

TAMIRES APARECIDA EVANGELISTA

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do Curso de Graduação, analisado pela seguinte comissão examinadora.

Orientador: Profº Ms. Osmar Aparecido Machado

Analisador (1): Profº Esp. Guilherme de Cleva Farto

Assis

2012

DEDICATÓRIA

Dedico este trabalho à minha família pelo incentivo, ao meu noivo pela paciência e aos meus amigos pela confiança depositada em meus sonhos.

AGRADECIMENTOS

Em primeiro lugar agradeço a Deus, pela saúde e pela oportunidade de realizar esse trabalho, e pelo amparo nos momentos difíceis.

À minha família, por me incentivarem, permanecerem sempre ao meu lado, e por me apoiarem durante minha caminhada nesses anos de faculdade e em toda minha vida.

Ao meu noivo por dedicar seu carinho e atenção nos momentos felizes e infelizes da minha vida, e por ser meu companheiro na minha vida acadêmica.

Ao meu orientador Profº Ms. Osmar Aparecido Machado, pelas sugestões e ensinamento durante todo esse trabalho.

A todos amigos que souberam compreender a minha ausência e por todo apoio que me deram.

RESUMO

Web Services são componentes do software capazes de oferecerem um tipo de serviço pela internet ou rede corporativa. Através do uso de padrões da internet como HTTP, XML e SOAP é possível acessar estes serviços por diferentes sistemas, independentes da plataforma em que foram desenvolvidos. A finalidade deste trabalho é apresentar a arquitetura de Web Services como um recurso disponível que facilita a interação entre empresas e seus serviços, além do desenvolvimento de um Web Service e um aplicativo que o consumirá para demonstração de sua eficácia.

Palavras Chaves: Web Services, Seguro, SOAP.

ABSTRACT

Web Services are software components able to offer a kind of service over the Internet or corporate network. Through the use of Internet standards like HTTP, XML and SOAP is possible to access these services through different programs, independent of the platform on which they were developed. The purpose of this paper is to present the architecture of web services as an available resource, which facilitates interaction between companies and their services, and the development of a Web service and an application that will to demonstrate of its effectiveness.

Key Words: Web Services, Security, SOAP.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CORBA	Commom Object Request Broker Architecture
DTD	Document Type Definition
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDL	Interface Definition Language
JMS	Java Message Service
RPC	Remote Procedure Call
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
UML	Unified Modeling Language
UDDI	Universal Description, Discovery and Integration
XML	Extensible Markup Language
WSDL	Web Service Description Language
W3C	World Wide Web Consortium

SUMÁRIO

1. INTRODUÇÃO

A realização de seguros se torna a cada dia uma prática mais acessível e necessária aos consumidores. Não há uma concordância unanima em relação ao surgimento do seguro, porém considera-se ser o instinto humano de cuidar de seus bens materiais ou físicos a forma mais primitiva de seguro.

O primeiro contrato de seguro registrado no mundo surgiu no Extremo Oriente, segundo MATTOS (1990, p.9)

> Os cameleiros da Babilônia atravessavam o deserto em caravanas para comercializar seus animais nas cidades vizinhas. Sentindo as dificuldades e os perigos da travessia, como a morte ou desaparecimento dos animais, estabeleceram um acordo: cada membro do grupo que perdia um camelo tinha garantia de receber um animal pago pelos demais cameleiros.

Com o passar dos anos novas formas de seguros foram surgindo, o ramo que mais rápido evoluiu em relação a seguros naquela época foram as navegações, principalmente por ser o meio mais usado para comercialização entre diferentes países.

> Povos da Antiguidade como os hebreus e os fenícios, grandes navegadores, enfrentavam riscos em suas contínuas travessias entre os mares Egeu e mediterrâneo. Por isso, procuraram uma forma de garantir-se contra possíveis prejuízos e firmaram um acordo entre si: quem perdia uma

> embarcação tinha garantida a construção de outra, paga pelos demais navegadores participantes da mesma viagem. (MATTOS, 1990, p. 9)

No Brasil, a primeira seguradora fundada foi em 1808, na Bahia, com o nome de “Companhia de Seguros Boa-Fé”, que operava somente seguro marítimo. (PULIDO, 2006 p.11). O surgimento de novas modalidades de seguro ocorreu de forma natural tornando assim o seguro um ramo extremamente concorrido; por isso as pequenas e médias corretoras encontram a necessidade de melhorar sua eficiência a fim de se tornarem mais competitivas.

Uma das maneiras de melhoria da competitividade é o uso eficiente de novas tecnologias, que possibilitam chegar aos consumidores de forma mais rápida e com menor custo. Atualmente existem várias tecnologias acessíveis aos corretores, que possibilitam melhor gerenciamento dos segurados de suas respectivas apólices, além de melhor gerenciamento dos produtos da corretora.

Entretanto, existem ainda muitas necessidades de serviços não satisfeitas para as operadoras, como por exemplo, múltiplos cálculos de seguro, já que atualmente são realizados por aplicativos individuais, disponibilizados pelas matrizes das seguradoras, cada um com suas características.

As corretoras geralmente atuam com várias empresas, como Porto Seguro, Sul América, Liberty, Allianz, Zurich, Bradesco, HDI Seguros, Marítima, Chubb, Bancos, dentre outras, o que dificulta a escolha da melhor alternativa para o cliente.

Neste sentido, uma das necessidades das corretoras é uma ferramenta que possibilite o cálculo do seguro a partir das várias seguradoras. Esta operação, atualmente, é realizada manualmente sendo necessário consultar cada um dos aplicativos, realizar o cálculo, imprimir e finalmente, analisar a melhor proposta para o cliente.

Dessa forma, a proposta deste trabalho é o desenvolvimento de uma ferramenta para múltiplos cálculos de seguro, que reverteria em um ganho de tempo essencial a esse ramo de atividade, já que poderiam utilizá-lo para atender melhor aos

segurados ou, até mesmo na prospecção de novos clientes, além de auxiliar na tomada de decisão em relação à melhor seguradora a ser contratada.

1.1 OBJETIVO

O objetivo deste trabalho é desenvolver um Web Service, que proverá o serviço de cálculo de seguros e um sistema gerenciador de corretoras, que fornecerá os dados para o cálculo dos seguros. Com isso, será demonstrado um considerável aumento na eficácia desses sistemas ao realizar múltiplos cálculos de seguro, que resulta na satisfação e maior produtividade do cliente final, os corretores.

1.2 PÚBLICO ALVO

O público alvo direto desse trabalho são os corretores de seguros e as seguradoras. Entretanto, por conta do tipo de negócio que atenderá, pode se incluir como publico alvo indireto, os clientes das corretoras, ou seja, os segurados.

1.3 JUSTIFICATIVA

Atualmente as tecnologias acessíveis aos corretores, são sistemas web, fornecidos pelas seguradoras, construídos sobre navegadores, portanto a interação dos corretores com o sistema é feita com o preenchimento de formulários e execução de ações enviadas através de protocolo HTTP para processamento e retorno de requisições.

Por isso neste trabalho é proposto um sistema de múltiplos cálculos de seguros, que utilizaria os cadastros dos segurados do sistema gerenciador da corretora para preenchimento dos formulários. Através da tecnologia dos Web Services, os formulários preenchidos seriam enviados, via protocolo SOAP, ao Web Service da seguradora, que retornaria como resposta o valor do seguro.

A implementação desta solução se reverteria em um ganho de tempo substancial na elaboração das propostas de seguro, essencial nesse ramo de atividade. Além disso, a aplicação poderá ser utilizada para melhorar a qualidade do atendimento aos segurados, ou até mesmo conseguir novos clientes, além de auxiliar os clientes na tomada de decisão sobre a melhor proposta a ser contratada.

Portanto, o desenvolvimento deste software apresenta-se como uma alternativa viável de ser utilizado pelas operadoras de seguros, proporcionando benefícios significativos para as mesmas e também para seus clientes.

Um fator relevante na proposta é que as tecnologias utilizadas para o desenvolvimento da aplicação são consideradas de baixo custo, pois fazem parte do conjunto de ferramentas classificadas como software livre, como Java e Apache Axis, uma aplicação livre do SOAP para implementação dos serviços Web Java. E, finalmente, a ferramenta de banco de dados que será utilizada é o PostgreSQL.

1.4 METODOLOGIA

O desenvolvimento deste projeto contempla uma série de etapas e atividades, que devem ser bem elaboradas a fim de garantir o sucesso do projeto. Dentre as principais etapas, destacam-se:

I. Levantamento bibliográfico: Serão realizados levantamentos de referencial teórico e pesquisas relacionadas às ferramentas utilizadas no projeto, bem como sobre os principais conceitos envolvidos no escopo do projeto. Para

tanto, serão utilizadas pesquisas em livros, internet, entrevistas, reuniões com o orientador, dentre outras;

II. Desenvolvimento do aplicativo. Esta etapa, considerada como o corpo principal do projeto, contempla:

- Investigar e entender a aplicabilidade das várias tecnologias que serão utilizadas no desenvolvimento do trabalho;

- Desenvolvimento: Levantamento de requisitos, análise, programação, testes e implementação;

III. Documentação: elaboração de documentação escrita dos aspectos teóricos e práticos do sistema, bem como confecção de material relativa às apresentações necessárias ao cumprimento dos critérios para os trabalhos de conclusão de curso.

1.5 CRONOGRAMA

O trabalho de conclusão de curso é uma das atividades do calendário acadêmico da FEMA – Fundação Educacional do Município de Assis. Desta forma, suas atividades precisam ser desenvolvidas em consonância com as datas definidas no cronograma de instituição.

Para tanto, o sucesso do projeto está diretamente relacionado com o cumprimento das atividades definidas no cronograma do projeto (Tabela 1) alinhadas com as atividades da instituição.

MESES	ATIVIDADE
FEVEREIRO	Escolha do Professor Orientador
MARÇO	Definição do Trabalho e desenvolvimento do Pré – Projeto
ABRIL	Entrega pré – projeto
MAIO, JUNHO E JULHO	- Levantamento e escrita dos aspectos teóricos referentes às tecnologias e ferramentas utilizadas no projeto; - Análise: Levantamento de requisitos, diagramação,
JULHO	Qualificação
AGOSTO, SETEMBRO, OUTUBRO	-Escrita: Revisão, complementação e documentação do projeto; - Desenvolvimento do projeto
NOVEMBRO	Entrega trabalho final e apresentação do Projeto à banca examinadora

Tabela 1 – Cronograma da Estrutura de Desenvolvimento

2. REFERENCIAL TEÓRICO

Para o desenvolvimento deste trabalho serão utilizadas diversas tecnologias e protocolos, como Java, SOAP, WSDL, que compõem um Web Services. A fim de explorar as vantagens e desvantagens de cada uma destas tecnologias, bem como para se obter melhor conhecimento e domínio de seus usos, elas serão descritas na sequencia, com suas especificidades.

2.1 WEB SERVICES

Segundo BECKER et al. (2008), em meados dos anos 60 o modelo de computação predominante eram os mainframes, computadores de grande porte físico que se caracterizavam pelo alto poder de processamento e pela estrutura mono-processada. Os sistemas eram multiusuários, ou seja, muitos usuários conectados por terminais (computadores sem processamento e armazenamento local) a um único processador e a uma única memória. Neste ambiente, as empresas desenvolviam seus próprios produtos e soluções. Os equipamentos geralmente não tinham compatibilidade com produtos de outros fabricantes.

Em meados dos anos 70, começaram a padronização da ISO, que propôs uma arquitetura de conexão dividida em 7 camadas, então este modelo centralizado foi perdendo força com o surgimento e rápido crescimento das tecnologias de comunicação, como redes de computadores e a internet, assim que a internet foi aberta para exploração comercial, surgiu a necessidade de se padronizar a comunicação entre diferentes plataformas e linguagens de programação, para assim criar um modelo de computação distribuída, com a arquitetura cliente-servidor, em um ambiente descentralizado, onde aplicações distribuídas se tornariam serviços disponíveis na rede. BECKER (2008)

Há dois principais motivos para o surgimento da computação distribuída, o desenvolvimento dos processadores e o surgimento das redes de computadores. O principal objetivo é proporcionar o compartilhamento de recursos, seja hardware ou software. COULOURIS et al.(2007)

Goulart (2002, p. 12) afirmar:

> A interligação das máquinas em redes propiciou o surgimento do conceito de sistemas distribuídos, aproveitando o potencial das máquinas interligadas. De outro lado, tem-se os sistemas compostos de um único processador, memória e periféricos os quais são chamados de sistemas centralizados. Segundo Tanenbaum (1995), no passado era válido a Lei de Grosch, um especialista em computação, preconizando que "o poder computacional de um processador é proporcional ao quadrado de seu preço, ou seja, pagando duas vezes mais, pode-se obter o quádruplo da performance". Hoje em dia esta Lei de Grosch não é mais válida, pois com as novas tecnologias, em especial usando arquitetura INTEL a cada ano, pelo mesmo preço, obtemos o dobro ou o quádruplo de performance em relação ao equipamento anterior.

BECKER et al, (2008, p. 2) afirma "Com a evolução da computação distribuída surgiram novos padrões para o desenvolvimento de aplicações distribuídas orientadas ao uso de rede, como por exemplo, CORBA(Commom Object Request Broker Architeture)[CORBA 98]."

Segundo FREIRE (2002)

> Apesar de amplamente aceita, a arquitetura CORBA, não se tornou padrão – bem como outras diversas arquiteturas de distribuição de objetos, tais como padrão RMI da Sun. Argumenta-se que isso não ocorreu por alguns motivos simples: a primeira coisa que desmotivou a adoção desses padrões foi a necessidade de migração de código legado para as novas plataformas, reescrita de código e adaptação à forma de trabalho das novas arquiteturas. Além disso, alguns dos padrões mais promissores da arquitetura de objetos distribuídos pecam também pela falta de suporte por parte de linguagens de programação distintas [...]. Além disso, a adaptação a uma nova plataforma sempre traz custos e toma tempo – toda uma nova tecnologia tem que ser assimilada pelos usuários, programadores e clientes.

Neste cenário, as empresas precisavam integrar softwares legados com novas soluções levando à criação de novos produtos. Esta integração só é possível com a criação de um padrão para troca de dados independente de plataforma. Uma forma de alcançar isso é através da utilização de Web Services. (BECKER et al., 2008 p .2).

SAMPAIO (2006) define Web Service como um aplicativo servidor que disponibiliza um ou mais serviços para seus clientes, de maneira fracamente acoplada. Segundo GIRARDI (2004), pode-se entender Web Services como um aplicativo que possui suas funcionalidades disponíveis pela rede através de mensagens baseadas em XML, padrão também utilizado para a disponibilização das operações e descrição do serviço. No entanto, HENDRICKS (2002) define como um pedaço de negócio localizado em algum lugar da internet disponível através de protocolos da internet.

Percebe-se não haver uma única definição para Web Services, porém sua principal função é aceitar requisições de outros sistemas através da rede. Como é utilizado o padrão XML para troca de mensagens, tanto na implementação do serviço quanto na aplicação que acessará o web service, é possível a comunicação entre os sistemas de diferentes linguagens de programação, realizando assim a integração de aplicações que foram desenvolvidas, ou não, em diferentes linguagens, através

da Web, e precisam trocar informações, independentes de seus Sistemas Operacionais.

Neste projeto serão utilizadas algumas situações reais de transições entre empresas para demonstrar a vantagem de se usar aplicações baseadas em Web Services visando a reusabilidade de funcionalidades e a interoperabilidade com os outros sistemas.

2.1.1 Situação 1

Um primeiro exemplo é em relação a qualquer estabelecimento comercial que possua vários fornecedores, dos quais precisa sempre estar em contato tanto para saber os valores de produtos quanto para descobrir os dados da entrega após fechado o negócio.

Uma possível solução para esse cenário seria os que os fornecedores desenvolvessem um Web Service com as informações necessárias aos estabelecimentos como valores de produtos, validade, prazo de entrega e etc., mantendo-o sempre atualizado. Os estabelecimentos em contra partida acrescentariam a seus sistemas um consumidor de Web Services, acessando assim paralelamente os Web Services dos fornecedores, verificando os dados necessários para, no final da tarefa, gerar um relatório com os melhores fornecedores.

Para esse exemplo pode considerar como possíveis estabelecimentos uma padaria, lanchonete, livraria, loja de cosméticos dentre outros.

2.1.2 Situação 2

Outra situação para o uso de Web Service seria lojas de vendas on-line, onde supõe se que o cliente tenha escolhido como forma de pagamento o cartão de crédito. O

primeiro procedimento para a compra ser aprovada é o sistema da loja verificar com a administradora do cartão do cliente se há crédito suficiente para a compra. Para isso, é necessário o consumo do Web Service da administradora do cartão através do sistema da loja. Mas indo um pouco mais além, os processos de comunicação entre os sistemas, seria assim:

CONSUMIDOR – Sistema da Loja	**PRODUTOR – Web Service da Administradora de Cartão**
1 - Acessa o serviço do Web Service. 3 – Fornece as informações necessárias. 6 – Recebe as resposta e comunica ao cliente a aprovação ou recusa da compra.	2 – Solicita os dados necessários para a análise. 4 – Verifica o histórico de compras do cliente em questão e define o valor de crédito que há. 5 – Envia uma resposta.

Tabela 2 – Comunicação entre aplicação consumidora e Web Service consumido

Supondo que os sistemas sejam desenvolvidos em plataformas distintas, por exemplo, o consumidor utilizando Java e o produtor em C#, a comunicação entre eles será executada de forma normal, pois uma das principais características de um Web Service é a interoperabilidade entre plataformas, ou seja, as aplicações de diferentes plataformas podem conversar entre si trocando ou processando dados normalmente.

Como visto nos exemplos citados, Web Services foram criados para serem aplicações que aceitam solicitações de outros sistemas, sempre baseados em padrões, possibilitando assim a interoperabilidade entre sistemas e agilizando as transações.

2.2 VISÃO GERAL DOS WEB SERVICES

Segundo GIRARDI (2004, p. 28)

> Uma definição técnica de web services poderia ser como um serviço disponibilizado na Internet, descrito via WSDL (*Web Services Description Language*), registrado via UDDI (*Universal Description, Discover and Integration*), acessado utilizando SOAP (*Simple Object Access Protocol*) e com os dados transmitidos sendo representados em XML (*Extensible Markup Language*).

Para melhor compreensão destes componentes, citados por Girardi (2004), eles serão apresentados na sequencias, com suas características e funções.

2.2.1 Extensible Markup Language (XML)

Extensible Markup Language (XML) é um padrão world wide web consortium (W3C) um grupo responsável por diversos padrões da internet, é uma especificação proprietária, ou seja, não pertence a nenhuma empresa, supervisionada pela XML Working Group.

É um formato descritor de dados, que facilita na declaração mais precisa de conteúdos e resultados mais significativos de busca através de múltiplas plataformas. Por ser um arquivo texto há facilidade na leitura e processamento de seus arquivos, gerando assim menor incompatibilidade, tornando-o ideal para troca de dados entre aplicativos. Ele é um dos principais motivos do crescimento em

massa em relação ao uso de web services, pois através dessa linguagem os web services ficaram independentes de plataforma.

Segundo MENÉNDEZ (2002, p.21)

> XML é um padrão da indústria de informática que tem como maior objetivo o intercâmbio de dados. Ele permite que sistemas possam trocar informações de uma forma mais abrangente que arquivos texto, já que podemos dar semântica aos dados que estão sendo manipulados.

BECKER (2008, p.3) define XML como:

> XML (Extensible Markup Language) ou linguagem extensível de marcação é uma linguagem designada para descrever e estruturar informações. Como uma linguagem de marcação, XML se assemelha com a linguagem HTML, possuindo marcações para descrever os dados. Porém, estas marcações não são pré-definidas na linguagem, tornando possível a criação de marcações de acordo com necessidades específicas.

Todas as linguagens dão suporte à XML, por sua estruturada e hierárquica, o que o torna mais compreensivo ao ser humano. Devido a isso é utilizado para a substituição de arquivos de configuração e comunicação.

Seu uso como base para troca de informações é essencial aos web services, pois foi criado em um formato padrão, compatível com as diferentes plataformas e aplicações.

2.2.1.1 Documento XML

Um documento XML é um arquivo texto, composto por marcações aninhadas e delimitadas. Assim como o HTTP o XML também utiliza etiquetas, que indicam a estrutura e o conteúdo armazenado. Porém diferentemente do HTTP, onde as etiquetas já estão pré-definidas, no XML as etiquetas são extensíveis representando fielmente os dados armazenados pelo documento XML.

Para descrição da estrutura de documentos, o XML define marcadores que identificam e delimitam partes do documento, estes chamados de elementos. Na maioria das vezes os elementos são aninhados, onde o elemento mais externo é chamado de elemento raiz. Todo documento XML tem que ter pelo menos um elemento raiz. Entretanto elementos comunicam-se com outros elementos, podendo ter uma relação de pai ou filho, caso haja um elemento dentro do elemento raiz, considera-se esse elemento como filho, porém se houver outro elemento dentro deste, ele se torna pai desse novo elemento e ao mesmo tempo filho do elemento raiz. Além disso, elementos também podem conter atributos.

BECKER (2008, p.4) afirma,

> Um documento XML “bem formado” é um documento que está em conformidade com as regras sintáticas. Um documento XML “válido” é um documento bem formado e que está em conformidade com as regras de um DTD (Document Type Definition ou Definição do Tipo do Documento). Um DTD define os elementos permitidos em um documento XML. O propósito de um DTD é definir a estrutura do documento com uma lista de elementos

> possíveis. Com o uso de uma definição de documento, cada arquivo XML pode carregar uma descrição do seu próprio formato. Deste modo, grupos independentes podem concordar em usar um DTD comum para a troca de informação. Uma aplicação pode usar um DTD para verificar se os dados que recebeu são válidos.

O DTD é o que torna o XML portável, basta a aplicação que receberá um documento XML, processá-lo de acordo com as regras especificadas no DTD. Caso tenha elementos a mais, um dos elementos esteja fora de ordem ou um dos elementos não apareça, o documento torna-se inválido e aplicação fica ciente que um erro aconteceu.

Outra característica de portabilidade do XML é o fato do documento estar desvinculado com o formato de exibição dos dados, o que possibilita demonstrá-los em diferentes mídias.

Abaixo apresento um documento XML, demonstrando cada uma das características citadas acima.

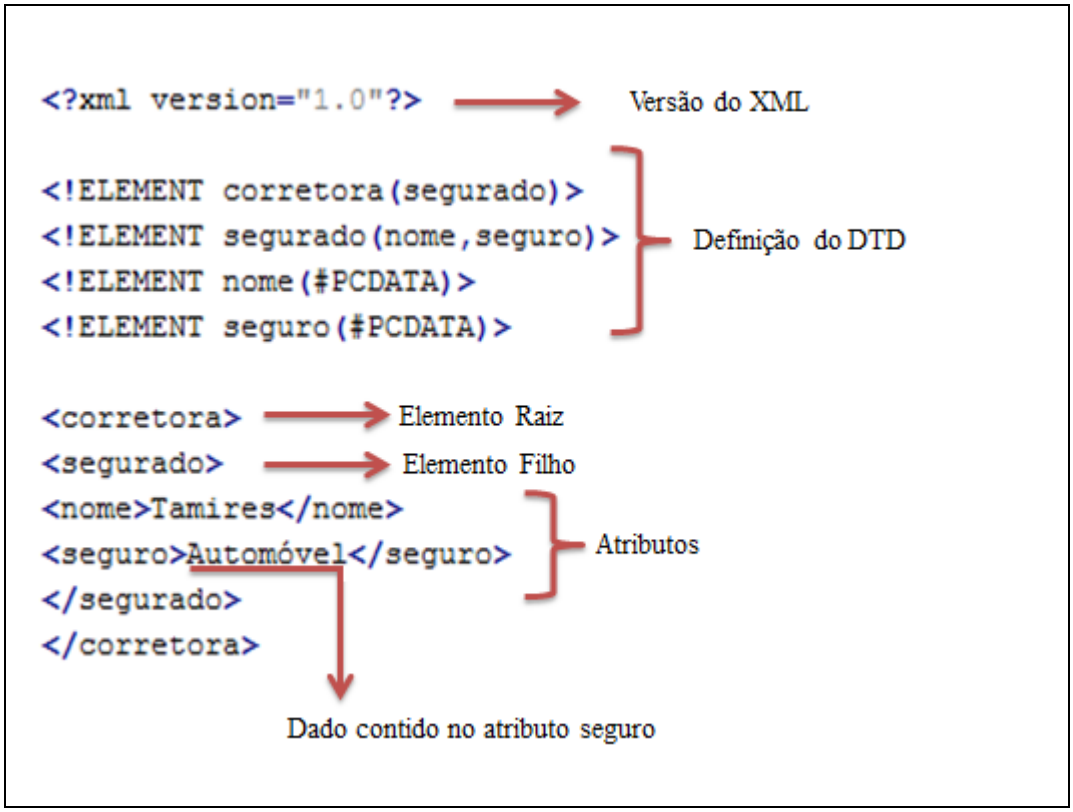

Figura 1 – Exemplo Documento XML

2.2.2 Simple Object Access Protocol (SOAP)

É um protocolo baseado em XML, usado para troca de informações em uma arquitetura orientada a serviços utilizando-se o HTTP, uma vez que esse protocolo é suportado por todos browsers, firewalls e servidores.

BECKER (2008, p.7) afirma,

> SOAP (Simple Object Access Protocol) originou-se da ideia de um mecanismo de RPC baseado em XML originalmente proposto por Dave Winer em 1998. A ideia evoluiu e hoje SOAP é uma especificação da W3C

> proposta por organizações como Userland, Ariba, Microsoft, IBM, Compaq, HP, Lótus, SAP, entre outras.

HENDRICKS (2002; apud GOMES, 2005, p.42) define SOAP como:

> SOAP é um protocolo superficial que exporta informação de forma centralizada em um ambiente distribuído. Ele é um protocolo baseado em XML que consiste de três partes: um envelope que define um framework para descrever o que é uma mensagem e como processá-la; um conjunto de regras de codificação para expressar instâncias de tipos definidos pela aplicação; e uma convenção para representar chamadas remotas de procedimento e respostas.

SOAP utiliza XML para codificar todas as mensagens, porém essas não podem conter um DTD. Ela deve incluir XML namespaces em todos os elementos e atributos definidos pelo SOAP, isso proporciona a vantagem de todos os elementos definidos não entrarem em conflitos com os elementos padrões do SOAP. “Sendo assim, todos os elementos e atributos padrão do SOAP recebem um prefixo com o identificador do namespace SOAP-ENV¸ que está associado ao namespace http://schemas.xmlsoap.org/soap/envelope.” – GOMES (2005).

Seu uso juntamente com o XML e mecanismos de transporte padrão (HTTP) garantem a interoperabilidade e intercomunicação entre diferentes sistemas.

Por trafegar através da porta 80 de um servidor HTTP, onde apesar de ser fortemente monitorada, permite um melhor desempenho no tráfego de informações, atravessando com facilidade os firewalls, outras vantagens no uso de SOAP é o mapeamento satisfatório para o padrão solicitação e resposta do HTTP, além de

poder ser usado com outros protocolos de transporte além do HTTP como SMTP e JMS.

2.2.2.1 Mensagem SOAP

Uma mensagem SOAP é um documento XML “bem formado” que pode conter envelope, HEADER (cabeçalho) e BODY (corpo).
SOAP Envelope é o elemento raiz da mensagem SOAP, indica ao receptor o início e fim da mensagem, contendo dentro deste o HEADER caso haja necessidade, e o BODY obrigatoriamente. É representado pelo elemento SOAP-ENV: ENVELOPE.
Segundo MENÉNDEZ (2002, p.24)

> O elemento envelope está localizado no topo da hierarquia do protocolo SOAP. Ele é composto de dois elementos: XML namespace e encodingStyle. O XML namespace é um conjunto de nomes para tipos de elementos XML e nomes de atributos, isto é, um esquema XML [2]. Um dos esquemas mais usados nas mensagens SOAP encontra-se em http://schemas.xmlsoap.org/soap/envelope.
>
> O atributo encodingStyle identifica o tipo de dados reconhecido pelas mensagens SOAP, além de especificar como os dados devem ser serializados para o transporte através da web. É possível indicar mais de um URI (Uniform Resource Identifiers) para identificar as regras de serialização, onde a ordem de colocação é da mais específica para a menos específica.

SOAP HEADER é opcional, porém ao ser usado deve ser depois do envelope, sendo especificada pelo elemento SOAP-ENV: HEADER. Seu uso possibilita um

gerenciamento de instruções específicas para processamento da mensagem SOAP, como por exemplo, autenticação e gerenciamento de transação de serviços.

MACIEL (2007, p. 24) diz

> Apesar de não obrigatório, o cabeçalho constitui-se no mecanismo chave para a capacidade de expansão do protocolo SOAP, já que com a adição apropriada de entradas no cabeçalho, nós de processamento são instruídos a se comportarem de modo específico em determinados contextos como autenticação, autorização, roteamento, etc. Ao receber a mensagem SOAP com este atributo, o nó deverá executar o que determina a entrada ou, em caso de falha, enviar uma mensagem padrão de erro.

SOAP BODY é o elemento principal da mensagem SOAP, além de obrigatório. Nele estão contidas as informações que o destinatário irá processar, pode conter um RPC, uma resposta RPC, um documento XML ou um relatório de erro.

GOMES (2005, p. 49) afirma

> Caso um elemento de cabeçalho esteja presente, o elemento Body deve seguir imediatamente o elemento Header. O elemento Body pode conter elementos filhos que são denominados entradas do corpo. Uma entrada do corpo é identificada por um namespace. Os elementos filhos imediatos de uma entrada de corpo não precisam ser qualificados por um namespace.

“Um elemento denominado <Fault>, um mecanismo sofisticado e preciso para informar ao emissor a ocorrência de algum erro ou falha no processamento de mensagens, também pode aparecer no corpo da mensagem.” MACIEL (2007 p. 24-25).

Abaixo demonstro uma mensagem SOAP, para ilustrar tudo informado acima.

```
<?xml version=”1.0” enconding=”uft-8”?>
<SOAP-ENV:Envelope xmlns:xsi=“http://www.w3.org/2001/XMLSchema-
  instance” xmlns:xsd=”http://www.w3.org/2001/XMLSchema”
  xmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope”>
  <SOAP-ENV:Header>
    <a:authetication xmlns:a=”http://www.wrox.com/soap/authetication”
      SOAP-ENV:mustUnderstand=”1”>
      <a:username>usuario</a:username>
      <a:password>senha</a:password>
    <a:authetication>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <cmd:processReboot xmlns:cmd=”http://www.wrox.com/soap/cmd”>
      <ip xsi:type=”xsd:string”>192.168.1.3</ip>
    </cmd:processReboot>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 2 – Exemplo Mensagem SOAP (In: MACIEL, 2007).

2.2.3 Universal Description, Discover and Integration (UDDI)

Universal Description, Discovery and Integration (UDDI) é um protocolo proposto para permitir armazenamento e localização de um Web Service, ele dita como Web Services devem ser registrados e localizados na rede. (FREIRE, 2002)

UDDI é um registro do web service, através dele é possível encontrar empresas que forneçam determinado serviço, ler sobre o serviço e contatar alguém para mais

informações, caso necessário. Seu uso não é obrigatório, porém é uma prática muito comum entre as empresas que fornecem seus serviços através de web services, porque é uma forma de divulgar tal serviço.

“Estes registros funcionam sobre múltiplos sites operadores, que podem ser utilizados por qualquer um que gostaria de disponibilizar alguma informação sobre seus negócios ou entidades.” (GOMES, 2005 p. 56).

2.2.3.1 Registro UDDI

A implementação de um registro UDDI é semelhante à de uma lista telefônica, onde temos as páginas amarelas, contém os serviços e produtos oferecidos organizados por categoria específica ou regiões geográficas, as brancas informa os dados de contato da empresa que oferece o serviço e as verdes possui os dados técnicos do serviço como, por exemplo, informações sobre transações, descrições de serviço e invocação de aplicações.

A estrutura do registro é constituída por quatro elementos, que são explicados a seguir. Bussiness Entity: define a empresa que publica o serviço e suas definições técnicas, nele também pode ser feita uma breve descrição do serviço.

Bussiness Service: “A estrutura B*usiness Service* contém informações sobre cada um dos serviços oferecidos pela empresa. Para cada businessEntity podemos ter vários businessService; este relacionamento é feito através do campo businessKey.”. MENÉNDEZ (2002, p. 29);

Binding Template: “Contém informações que descrevem como obter acesso a um service, informando quais os pontos de acesso ao serviço chamado através de URLs.” – AMORIM (2004, p. 17).

tModels: “Contém informações que descrevem uma especificação técnica do serviço. Por exemplo, os protocolos de rede ou regras de sequência.” – AMORIM (2004, p. 17)

Abaixo apresento um exemplo de registro UDDI para exemplificar tudo que foi dito acima.

```
<element name = "businessEntity">
    <type content = "elementOnly">
        <group order = "seq">
            <element ref = "discoveryURLs" minOccurs = "0" maxOccurs = "1" />
            <element ref = "name" />
            <element ref = "description" minOccurs = "0" maxOccurs = "*" />
            <element ref = "contacts" minOccurs = "0" maxOccurs = "1" />
            <element ref = "businessServices" minOccurs = "0" maxOccurs = "1" />
            <element ref = "identifiedBag" minOccurs = "0" maxOccurs = "1" />
            <element ref = "categoryBag" minOccurs = "0" maxOccurs = "1" />
        </group>

            <attribute name = "businessKey" minOccurs = "1" type = "string" />
            <attribute name = "operator" type = "string" />
            <attribute name = "authorizedName" type = "string" />
        </type>
    </element>
    <element name = "businessService">
        <type content = "elementOnly">
            <group order = "seq">
                <element ref = "name" />
                <element ref = "description" minOccurs = "0" maxOccurs = "*" />
                <element ref = "contacts" minOccurs = "0" maxOccurs = "1" />
                <element ref = "bindingTemplates" />
                <element ref = "categoryBag" minOccurs = "0" maxOccurs = "1" />
            </group>
            <attribute name = "serviceKey" minOccurs = "1" type = "string" />
            <attribute name = "businessKey" type = "string" />
        </type>
    </element>
    <element name = "bindingTemplate">
        <type content = "elementOnly">
            <group order = "seq">
                <element ref = "description" minOccurs = "0" maxOccurs = "*" />
                    <group order = "choice">
                        <element ref = "accessPoint" minOccurs = "0" maxOccurs = "1" />
                        <element ref = "hostingRedirector" minOccurs = "0" maxOccurs = "1" />
                    </group>
                    <element ref = "tModelInstanceDetails" />
            </group>
            <attribute name = "bindingKey" minOccurs = "1" type = "string" />
            <attribute name = "serviceKey" type = "string" />
    </type>
    </element>
    <element name = "tModel">
        <type content = "elementOnly">
            <group order = "seq">
                <element ref = "name" />
                <element ref = "description" minOccurs = "0" maxOccurs = "*" />
                <element ref = "overviewDoc" minOccurs = "0" maxOccurs = "1" />
                <element ref = "identifierBag" minOccurs = "0" maxOccurs = "1" />
                <element ref = "categoryBag" minOccurs = "0" maxOccurs = "1" />
            </group>
            <attribute name = "tModelKey" minOccurs = "1" type = "string" />
            <attribute name = "operator" type = "string" />
            <attribute name = "autorizedName" type = "string" />
        </type>
    </element>
```

Figura 3 – Exemplo Registro UDDI (IN MENÉNDEZ, 2002)

2.2.4 Web Service Description Language (WSDL)

Web Service Definition Language (WSDL) ou Linguagem de definição dos Web Services, é um formato XML para descrição de serviços web. O provedor do Web Service constrói um documento à partir do WSDL para descrever detalhes necessários na invocação do serviço por um consumidor. Como por exemplo, a interface do serviço, quais métodos estão disponíveis, as assinaturas dos métodos e os valores retornados, o endereço de localização do serviço e o protocolo que o serviço é capaz de processar para ser feita a comunicação. (MACIEL, 2007)

O padrão WSDL foi o último dos três protocolos que compõem a arquitetura dos Web Services, e foi lançado algumas semanas depois do UDDI. Para descrever os objetos, parâmetros e dados de forma universal, foi criada a gramática descritiva de objetos e serviços baseada em XML. (FREIRE, 2002)

Segundo I-Web (2003 p. 2)

> Simplificadamente pode-se dizer que o arquivo WSDL é um documento XML que descreve um conjunto de mensagens SOAP e a forma como essas mensagens são trocadas. Em outras palavras, o WSDL é para o SOAP o que o IDL é para o CORBA ou COM. Como o WSDL é XML, ele é legível e editável, mas na maioria dos casos, ele é gerado e consumido pelo software.

Cabe ao WSDL a responsabilidade de prover as informações necessárias e corretas ao consumidor, para que haja uma comunicação eficaz, entre as mensagens SOAP com provedor de serviços. Ou seja, quando um cliente deseja consumir informações do web service, ele obtém a descrição do serviço da WSDL, após construída a

mensagem, passando os parâmetros necessários de acordo com a WSDL, a mensagem é enviada para o endereço fornecido. O web service ao receber a mensagem valida-a conforme as informações contidas na WSDL, passando então a mensagem adiante para processamento do SOAP.

Seu uso proporciona como vantagens a manutenção do serviço já que toda a definição da interface de Web Services está contida nele, facilita o uso do serviço, pois reduz a possibilidade de erros em potenciais, já que a partir dele está definido os tipos de dados aceitos além de reduzir a quantidade de código a ser implementada pelo sistema consumidor.

2.2.4.1 Documento WSDL

Um documento WSDL é formado por duas partes, a descrição abstrata, que é responsável por definir o comportamento do serviço em relação ao que consome e produz e, a definição concreta responsável por descrever especificamente o protocolo utilizado na camada de transporte de rede, ou seja, informar como e onde acessar o serviço.

“Arquivos WSDL iniciam com um elemento raiz <definitions>, que contém geralmente atributos definindo namespaces que descrevem quais XML Schemas são usados ao longo do documento WSDL.” – MACIEL (2007, p.28).

Os elementos de um documento WSDL que compõem a descrição abstrata são:

Tipos: Define quais os tipos de dados aceitos pelo Web Service. É representado em um documento WSDL pelo elemento <types>.

Mensagem: Lista todas as mensagens que podem ser trocadas entre Web Service e consumidor, também define as partes das mensagens e seus tipos. É representado em um documento WSDL pelo elemento <message>

Operações: MACIEL (2007, p. 29) define como:

> Uma operação é um agrupamento das mensagens que podem ser trocadas em uma interação particular com o serviço e é indicado pelo elemento <operation>. Quatro tipos de operações são suportados:
>
> - Unidirecional: uma mensagem chega ao serviço e não é produzida nada como resposta;
>
> - Solicitação/Resposta: uma mensagem chega ao serviço e é produzida uma mensagem como resposta;
>
> - Pedido/Resposta: o serviço envia uma mensagem e obtém uma resposta de volta;
>
> - Notificação: o serviço envia uma mensagem e não recebe nada como resposta.

Tipos de Porta: “É o agrupamento lógico de operações, podendo ser comparado com uma classe, que é um agrupamento lógico de métodos; representado em documento WSDL pelo elemento <portType>.” GOMES (2005, p. 53)

Os elementos de um documento WSDL que compõem a descrição concreta são:

Ligação: Especifica o protocolo usado para transportar as mensagens. É representado em um documento WSDL pelo elemento <binding>.

Serviço: Informa a localização do serviço, através de seu elemento filho <port>. É representado em um documento WSDL pelo elemento <service>.

Segue abaixo um exemplo, onde pode ser visto a maioria dos elementos citados acima.

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions name="WServiceService"
    targetNamespace="http://hello.org/wsdl"
        xmlns:tns="http://hello.org/wsdl"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <types/>
  <message name="olaMundo">
    <part name="String_1" type="xsd:string"/>
  </message>
  <message name="olaMundoResponse">
    <part name="result" type="xsd:string"/>
  </message>
  <portType name="olaIF">
    <operation name="olaMundo">
      <input message="tns:olaMundo"/>
      <output message="tns:olaMundoResponse"/>
    </operation>
  </portType>
  <binding name="olaIFBinding" type="tns:olaIF">
    <operation name="olaMundo">
      <input>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                   use="encoded" namespace="http://ola.org/wsdl"/>
      </input>
      <output>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                   use="encoded" namespace="http://ola.org/wsdl"/>
      </output>
      <soap:operation soapAction=""/>
    </operation>
    <soap:binding
          transport="http://schemas.xmlsoap.org/soap/http"
          style="rpc"/>
  </binding>
  <service name="WService">
    <port name="olaIFPort" binding="tns:olaIFBinding">
      <soap:address location="http://localhost:8080/wservice"/>
    </port>
  </service>
</definitions>
```

Figura 4 – Exemplo Documento WSDL (IN MENÉNDEZ, 2002)

2.3 ARQUITETURA DOS WEB SERVICES

Web Services é um serviço disponibilizado através da rede, um dos principais motivos do seu crescimento no mercado atual se deve ao fato que suas tecnologias

possibilitam a comunicação entre aplicações, independente de plataforma ou linguagem de programação, sem que haja intervenção humana. E o principal conceito que dá embasamento a esse ponto é do SOA (Service Oriented Architecture ou Arquitetura Orientada à Serviço).

Como Web Service é orientado a serviço sua arquitetura é baseada na interação de três entidades: Provedor de Serviço (Service Provider), Consumidor de Serviço (Consumer Service ou Service Requestor) e Registro do Serviço (Service Register ou Registry Provider). Dessa interação surgem as operações de publicação (publish), pesquisa (find) e ligação (bind).

O Provedor de Serviço é o responsável por criar o serviço, representa a plataforma que hospeda o web service. Ele tem duas funções, a de descrever o serviço em um formato padrão, que seja compreensível para qualquer um que precise utilizar o serviço, e também deve publicar o serviço, utilizando a operação de publicação, em um registro central que esteja publicamente disponível aos interessados.

O Consumidor de Serviços são todos os “personagens” que utilizam qualquer Web Service publicado por um provedor de serviços, ele tem duas funções, a de pesquisar sobre o registro onde o provedor publicou sua descrição, e através dela recuperar os detalhes que farão a descrição do serviço e a partir disto realizar a ligação com o serviço web.

O Registro de Serviço é a localização central onde o provedor de serviços publica seus serviços e o consumidor pesquise-os. É o arquivo UDDI, que contém informações sobre os serviços existentes, a descrição técnica deles e as empresas que o oferecem.

Segundo BECKER et al. (2008, p. 6)

> A interação entre Web Services pode ser feita estaticamente ou dinamicamente em tempo de execução. Um solicitante de um serviço

descreve as características do serviço procurado e utiliza o provedor de registro para localizar um serviço apropriado. Uma vez localizado o serviço, a informação na descrição do serviço é utilizada para a interação entre cliente e servidor. A descoberta, a invocação dinâmica de serviços (publish, find, bind) e uma colaboração baseada em mensagens permite o desenvolvimento de aplicações distribuídas fracamente acopladas com um enorme grau de interoperabilidade.

A Figura 5 abaixo demonstra o que foi descrito acima.

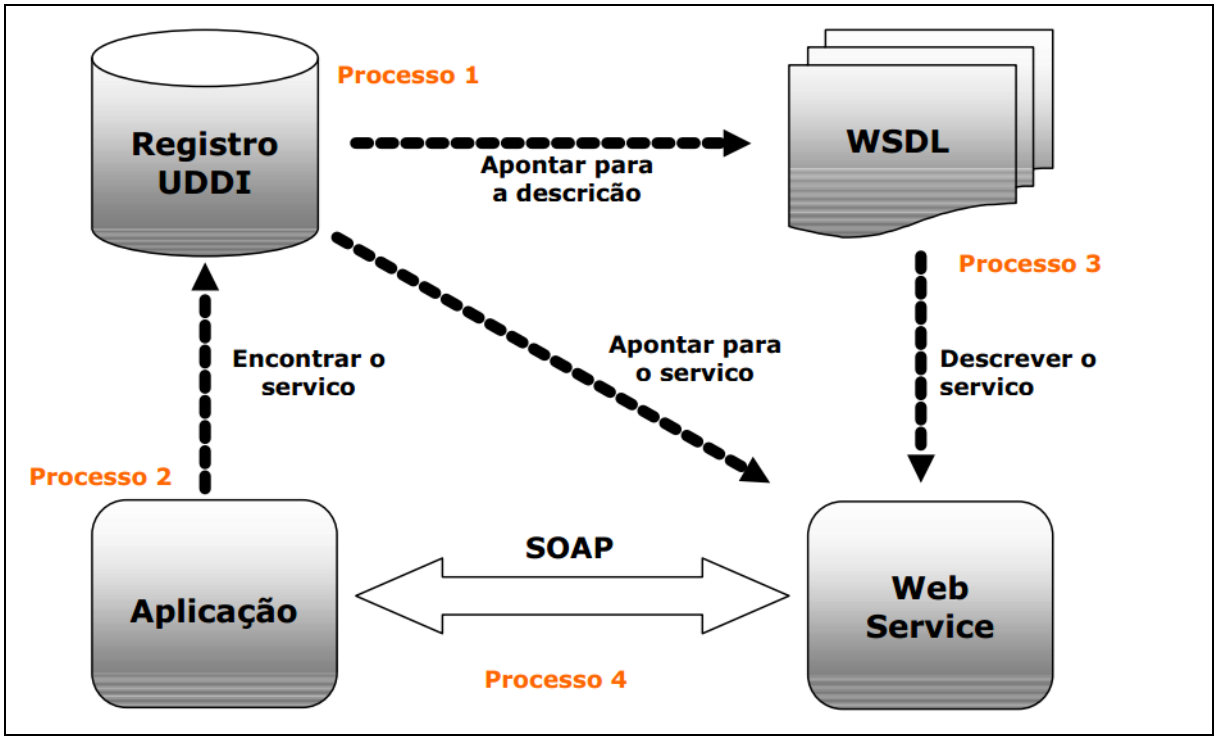

Figura 5 – Arquitetura de Web Services Fonte: MENÉNDEZ (2002)

3. DESENVOLVIMENTO DO PROJETO

O trabalho está estruturado em duas partes, sendo: uma escrita, composta pela revisão da literatura relacionada aos web services; e uma parte prática, composta pelo desenvolvimento de um web service simulando o ambiente de cálculo de seguros de uma seguradora e, ainda, um sistema para consumi-lo possibilitando a execução de múltiplos cálculos, simultaneamente, e auxiliando na tomada de decisão à melhor seguradora para o contrato do seguro calculado.

Foi realizado o levantamento de requisitos junto a dois corretores de seguros de Assis – SP, após uma explanação sobre o projeto, ambos concordaram ser uma excelente opção o uso de Web Services e logo pensaram em outros cenários, no mesmo sistema onde se pode usar da tecnologia a favor deles. A Figura 6 mostra detalhadamente o levantamento de requisitos.

CENÁRIO ATUAL

Atualmente os sistemas de corretores gerenciam segurados e seus seguros em geral.
Nos bancos há o uso de múltiplos cálculos, porém o valor é inviável a pequenas e médias corretoras.
Se houvesse uma solução mais barata com certeza usariam.

PROBLEMAS

Grande problema hoje é a perca de tempo nas cotações, e comissões.
Aviso dos segurados a vencer, atualmente faz a pesquisa pelo arquivo, porém se não estiver guardado certo perde o prazo de renovação que pode resultar em perda de bônus ao segurado, além da falta de cobertura.
Para realizar o cálculo das comissões por produtor na corretora, vai de seguradora em seguradora imprimindo o extrato de comissões, isso gasta até dois dias de serviço dependendo da quantidade de produtores e segurado.

POSSÍVEIS SOLUÇÕES

Guardar Apólice Atual, para economizar em papel que trará menos gasto à empresa.
Qualquer endosso (alteração, cancelamento) gera uma nova apólice de endosso.
Consulta apólices em atraso, emitindo aviso no programa caso houver.
Realizar o download dos extratos de comissões automaticamente.

Figura 6 – Levantamento de Requisitos

Segundo BOOCH et al.() “A UML((Unified Modeling Language), é adequada para a modelagem de sistemas, cuja abrangência poderá incluir sistema de informação corporativos a serem distribuídos a aplicações baseadas em Web e até sistemas complexos embutidos de tempo real”.

Diante disso, para a modelagem do aplicativo será utilizado a UML, para auxiliar no entendimento, ou melhor, na visualização de como ficará o aplicativo final. Apesar de a metodologia ser composta por 9 diagramas, neste projeto serão utilizados apenas o diagrama de classes e o de casos de uso, que serão demonstrados a seguir.

3.1 MODELAGEM SISTEMA CORRETORA

3.1.1 Diagrama Casos de Uso

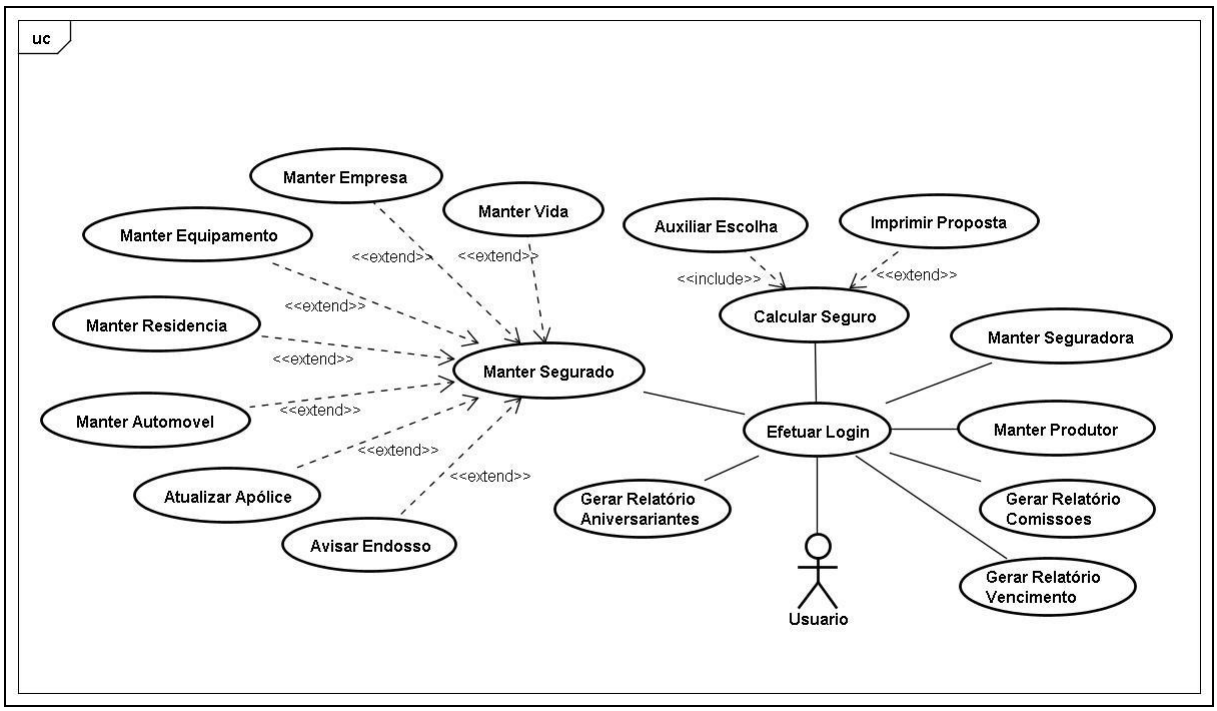

Figura 7 – Diagrama Caso de Uso Corretora

3.1.2 Narrativas Casos de Uso

3.1.2.1 Efetuar Login

1. Finalidade
    a. Login no Sistema
2. Atores
    a. Usuário
3. Pré-Requisito
    a. Login e Senha estarem cadastrados no sistema
4. Fluxo Principal
    a. Usuário acessa ao Sistema.
    b. O sistema solicita Login e senha.
    c. O sistema confirma Login e senha (7a).
    d. O Usuário terá acesso ao Sistema.
5. Fluxo Alternativo
6. Fluxo Exceção
7. Testes
    a. O sistema verifica Login e Senha.

3.1.2.2 Manter Produtor

1. Finalidade
    a. Efetuar cadastro, exclusão, alteração e listagem de produtor.
2. Atores
    a. Usuário

3. Pré-Requisito
    a. Usuário ter efetuado o Login.
4. Fluxo Principal
    a. O sistema solicita as informações necessárias.
    b. O Usuário informa os dados.(5a)(7a)
    c. O Sistema verifica se o Produtor já está cadastrado no sistema.(6a)(7b)
    d. O Sistema exibe a mensagem *Produtor Inserido com Sucesso.*
5. Fluxo Alternativo
    a. O Usuário cancela o cadastro.
6. Fluxo Exceção
    a. O Produtor já possui Cadastro.
7. Testes
    a. O sistema verifica se os dados foram preenchidos corretamente.
    b. O sistema verifica a existência do cadastro no sistema.

3.1.2.3 Manter Seguradora

1. Finalidade
    a. Cadastrar Seguradoras
2. Atores
    a. Usuário
3. Pré-Requisito
    a. Usuário ter efetuado Login no Sistema.
4. Fluxo Principal
    a. O Sistema solicita as informações necessárias.

b. O Usuário informa os dados. (5a)(7a)

c. O Sistema verifica se a seguradora já existe (6a) (7b).

d. O Sistema exibe a mensagem *Seguradora Cadastrada com Sucesso.*

5. Fluxo Alternativo

    a. O Usuário cancela o cadastro.

6. Fluxo Exceção

    a. A Seguradora já possui cadastro.

7. Teste

    a. O Sistema verifica se os dados foram inseridos corretamente.

    b. O Sistema verifica a existência da Seguradora no sistema.

3.1.2.4 Calcular Seguro

1. Finalidade

    a. Exibir valor do seguro em todas seguradoras que possuem Web Service.

2. Atores

    a. Usuário

3. Pré-Requisitos

    a. O Usuário ter efetuado Login no sistema.

4. Fluxo Principal

    a. O Sistema solicita as informações necessárias.

    b. O Usuário informa os dados. (5a) (7a)

    c. O Sistema envia os dados inseridos ao Web Service das Seguradoras.

    d. O Web Service retorna os valores. (6a)

    e. O Sistema exibe os valores dos seguros.

5. Fluxo Alternativo
    a. O Usuário cancela o cálculo.
6. Fluxo Exceção
    a. Os dados enviados não estavam corretos.
7. Testes
    a. O Sistema verifica se os dados foram informados corretamente.

3.1.2.5 Manter Segurado

1. Finalidade
    a. Cadastrar Segurados.
2. Atores
    a. Usuário.
3. Pré-Requisitos
    a. Usuário ter efetuado Login no sistema.
4. Fluxo Principal
    a. O Sistema solicita as informações necessárias
    b. O Usuário informa os dados. (5a) (7a).
    c. O Sistema verifica se o Segurado já existe. (6a) (7b)
    d. O Sistema exibe a mensagem *Segurado Cadastrado com Sucesso*.
5. Fluxo Alternativo
    a. O Usuário cancela o cadastro.
6. Fluxo Exceção
    a. Segurado já existente no Sistema.
7. Testes

a. O Sistema verifica os dados inseridos.

b. O Sistema verifica existência de Segurados.

3.1.2.6 Avisar Endosso

1. Finalidade
    a. Avisar ao Usuário quando ouve endosso nas apólices.
2. Atores
    a. Sistema
3. Pré-Requisitos
    a. Usuário ter efetuado Login no Sistema.
4. Fluxo Principal
    a. Sistema enviar dados ao Web Service das Seguradoras.
    b. Web Service retorna os dados. (6a)
    c. Sistema exibe os endossos ocorridos. (7a)
5. Fluxo Alternativo
6. Fluxo Exceção
    a. Dados enviados ao Web Service estão errados.
7. Testes
    a. Sistema verifica se houve endossos.

3.1.2.7 Atualizar Apólice

1. Finalidade
    a. Atualizar Apólice a cada alteração ocorrida.
2. Atores

a. Sistema

3. Pré-Requisitos
    a. Usuário ter efetuado Login no Sistema.
4. Fluxo Principal
    a. Sistema envia dados necessários aos Web Services das Seguradoras.
    b. Web Services retornam as apólices alteradas. (6a)
    c. Sistema atualiza as apólices. (7a)
5. Fluxo Alternativo
6. Fluxo Exceção
    a. Dados enviados incorretamente aos Web Services.
7. Teste
    a. Sistema verifica se houve alteração em alguma apólice.

3.1.2.8 Manter Automóvel

1. Finalidade
    a. Cadastrar Automóvel
2. Atores
    a. Usuário
3. Pré-Requisitos
    a. Usuário ter efetuado Login no Sistema.
4. Fluxo Principal
    a. Sistema solicita as informações necessárias.
    b. Usuário informa os dados. (5a) (7a)
    c. Sistema verifica se o Automóvel já existe. (6a) (7b)

d. Sistema exibe a mensagem *Automóvel Cadastrado com Sucesso.*

5. Fluxo Alternativo
    a. O Usuário cancela o cadastro.
6. Fluxo Exceção
    a. Automóvel já existe.
7. Teste
    a. Sistema verifica se os dados foram informados corretamente.
    b. Sistema verifica a existência do cadastro.

3.1.2.9 Manter Residência

1. Finalidade
    a. Cadastrar Residências.
2. Atores
    a. Usuário.
3. Pré-Requisitos
    a. Usuário ter efetuado Login no Sistema.
4. Fluxo Principal
    a. Sistema solicita as informações necessárias.
    b. Usuário informa os dados. (5a) (7a)
    c. Sistema verifica se a Residência já existe. (6a) (7b)
    d. Sistema exibe a mensagem *Residência Cadastrada com Sucesso.*
5. Fluxo Alternativo
    a. O Usuário cancela o cadastro.
6. Fluxo Exceção

a. Residência já existente.

7. Teste
    a. O sistema verifica se os dados foram informados corretamente.
    b. O sistema verifica a existência do cadastro no sistema

3.1.2.10 Manter Equipamento

1. Finalidade
    a. Cadastrar Equipamentos.
2. Atores
    a. Usuário
3. Pré-Requisitos
    a. Usuário ter efetuado Login no Sistema.
4. Fluxo Principal
    a. Sistema solicita as informações necessárias.
    b. Usuário informa os dados. (5a) (7a)
    c. Sistema verifica se o Equipamento já existe. (6a) (7b)
    d. Sistema exibe a mensagem *Equipamento Cadastrado com Sucesso.*
5. Fluxo Alternativo
    a. O Usuário cancela o cadastro
6. Fluxo Exceção
    a. Equipamento já existente.
7. Teste
    a. O sistema verifica se os dados foram informados corretamente.
    b. O sistema verifica a existência do cadastro no sistema

3.1.2.11 Manter Empresa

1. Finalidade
    a. Cadastrar Empresas.
2. Atores
    a. Usuário
3. Pré-Requisitos
    a. Usuário ter efetuado Login no Sistema.
4. Fluxo Principal
    a. Sistema solicita as informações necessárias.
    b. Usuário informa os dados. (5a) (7a)
    c. Sistema verifica se a Empresa já existe. (6a) (7b)
    d. Sistema exibe a mensagem *Empresa Cadastrada com Sucesso.*
5. Fluxo Alternativo
    a. O Usuário cancela o cadastro.
6. Fluxo Exceção
    a. Empresa já existente.
7. Teste
    a. O sistema verifica se os dados foram informados corretamente.
    b. O sistema verifica a existência do cadastro no sistema

3.1.2.12 Manter Vida

1. Finalidade
    a. Efetuar Cadastro de Seguros de Vida.

2. Atores
    a. Usuário
3. Pré-Requisitos
    a. Usuário ter efetuado Login no Sistema.
4. Fluxo Principal
    a. Sistema solicita as informações necessárias.
    b. Usuário informa os dados. (5a) (7a)
    c. Sistema verifica se o Seguro de Vida já existe. (6a) (7b)
    d. Sistema exibe a mensagem Seguro de Vida *Cadastrado com Sucesso.*
5. Fluxo Alternativo
    a. Usuário cancela o cadastro.
6. Fluxo Exceção
    a. Vida já existente.
7. Teste
    a. O sistema verifica se os dados foram informados corretamente.
    b. O sistema verifica a existência do cadastro no sistema

3.1.2.13 Auxiliar na Escolha

1. Finalidade
    a. Demonstrar ao Usuário qual o melhor seguro a ser feito.
2. Atores
    a. Sistema
3. Pré-Requisitos
    a. Usuário ter efetuado o cálculo de seguro.

4. Fluxo Principal
    a. O Sistema verifica qual o menor valor de Seguro recebido.
    b. O Sistema exibe a melhor escolha a ser feita.
5. Fluxo Alternativo
6. Fluxo Exceção
7. Teste

3.1.2.14 Imprimir Proposta

1. Finalidade
    a. Possibilita a impressão de propostas de seguro.
2. Atores
    a. Usuário
3. Pré-Requisitos
    a. Usuário ter realizado o cálculo de seguro.
4. Fluxo Principal
    a. O Usuário clica em imprimir proposta.
    b. O Sistema imprime a proposta.
5. Fluxo Alternativo
6. Fluxo Exceção
7. Testes

3.1.2.15 Gerar Relatório Aniversariante

1. Finalidade

a. O Usuário poderá quais segurados farão aniversário no período escolhido.

2. Atores
    a. Usuário
3. Pré-Requisitos
    a. Ter Segurados cadastrados no sistema.
4. Fluxo Principal
    a. O Usuário seleciona o período de pesquisa.(7a)
    b. O Sistema retorna os Segurados que farão aniversários neste período.
5. Fluxo Alternativo
6. Fluxo Exceção
7. Testes
    a. O Sistema verifica a existência de Aniversariantes no período.

3.1.2.16 Gerar Relatório Comissões

1. Finalidade
    a. O Usuário poderão ver todas as comissões a receber.
2. Atores
    a. Usuário
3. Pré-Requisitos
    a. Ter Segurados cadastrados no sistema.
4. Fluxo Principal
    a. O Usuário seleciona o período de pesquisa.
    b. O Sistema exibe as comissões do período escolhido. (7a)
5. Fluxo Alternativo
6. Fluxo Exceção

7. Testes
    a. O Sistema verifica se houve comissões emitidas no período escolhido.

3.1.2.17 Gerar Relatório Vencimento

1. Finalidade
    a. Possibilitar ao Usuário ver as apólices a vencer.
2. Atores
    a. Usuário.
3. Pré-Requisitos
    a. Ter Segurados cadastrados no sistema.
4. Fluxo Principal
    a. O Usuário seleciona o período.
    b. O Sistema exibe as apólices a vencer no período escolhido. (7a)
5. Fluxo Alternativo
6. Fluxo Exceção
7. Testes
    a. O Sistema verifica se há vencimentos no período desejado.

3.1.3 Diagrama de Classes

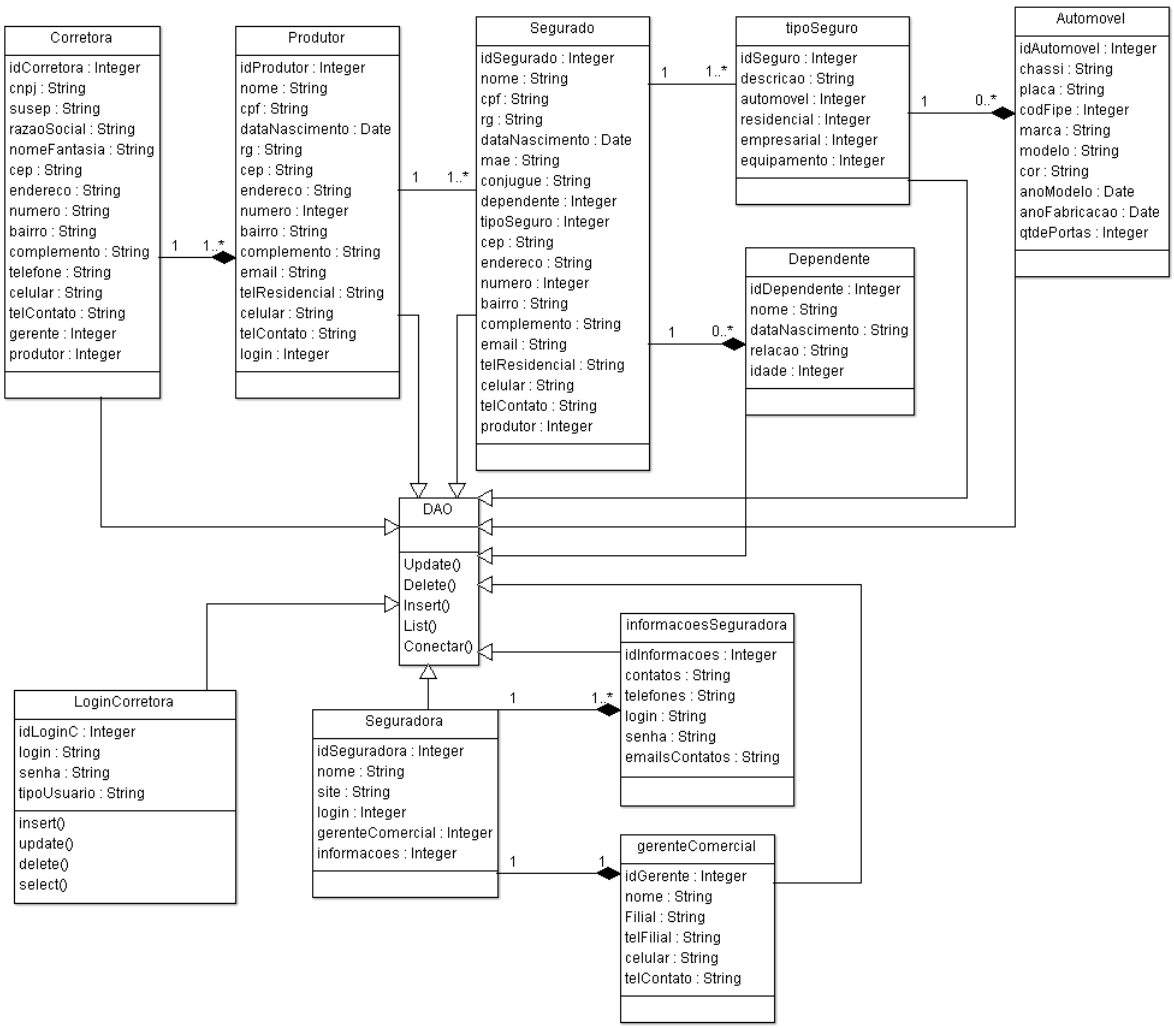

Figura 8 – Diagrama Classes Geral

3.1.3.1 Relação Corretora – Produtor – Segurado

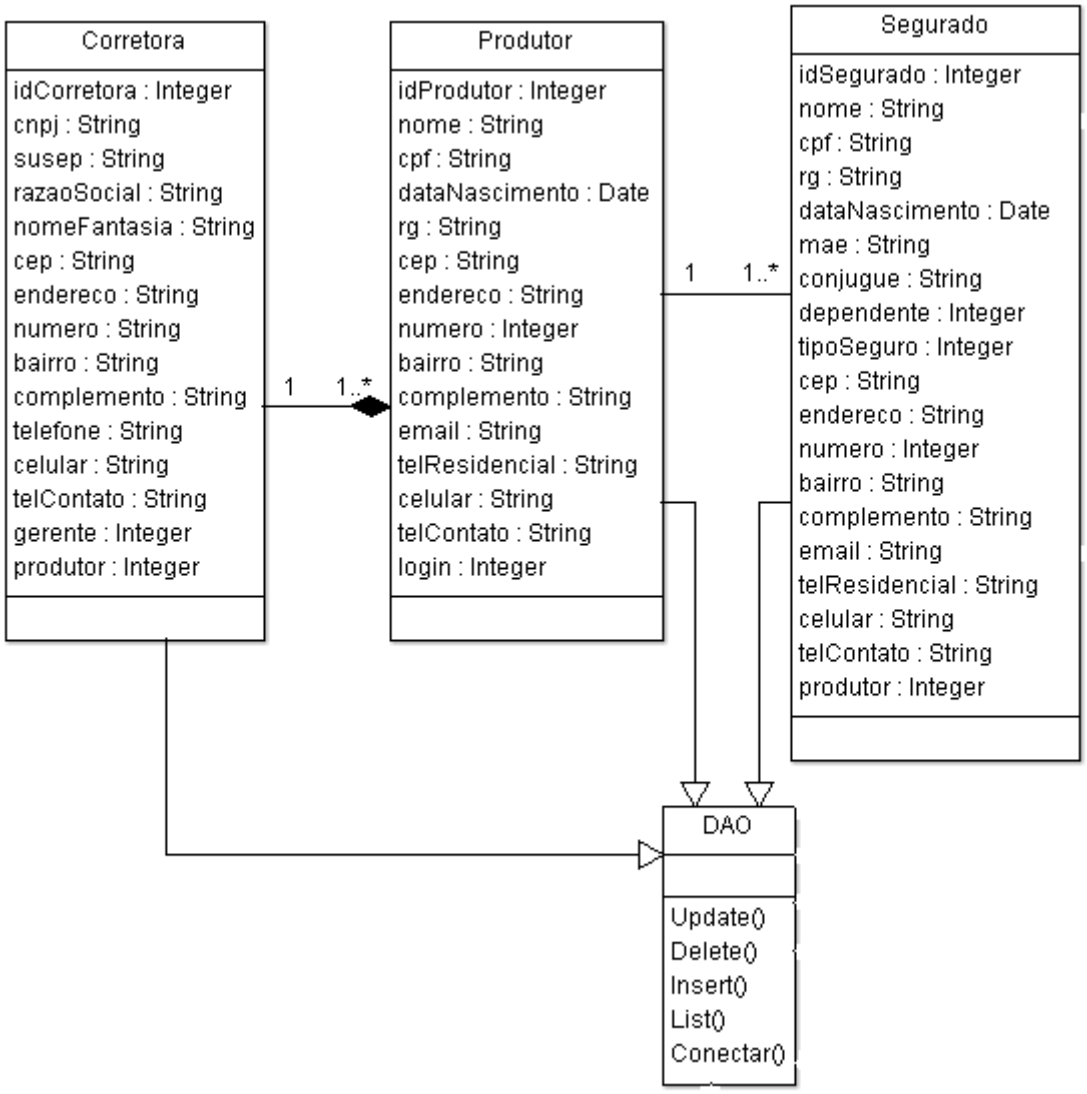

Figura 9 – Diagrama Classes Relação Corretora - Produtor - Segurado

3.1.3.2 Relação Segurado – Dependente - Tipo Seguro - Automóvel

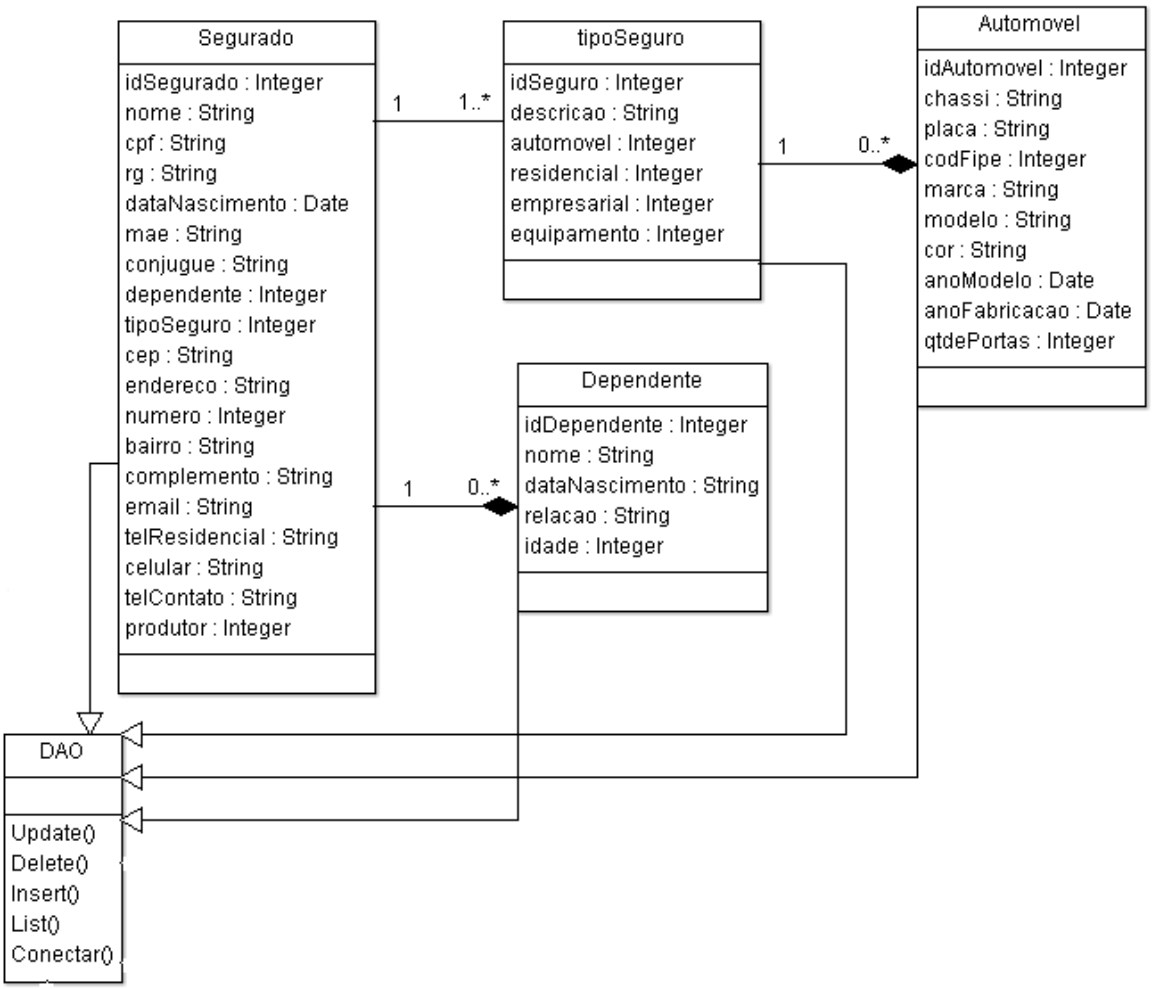

Figura 10 - Diagrama Classes Relação Segurado - Dependente – Tipo Seguro - Automóvel

3.1.3.3 Login Corretora

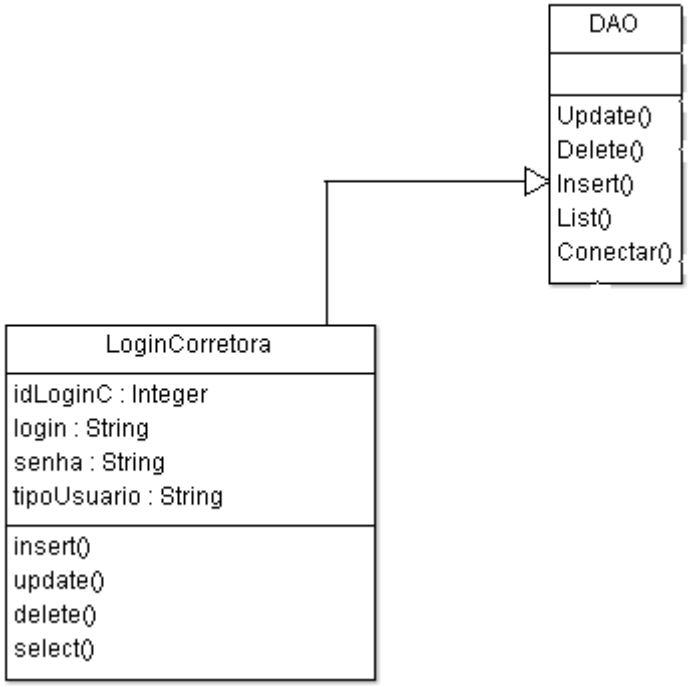

Figura 11 – Diagrama Classes Login Corretora

3.1.3.4 Relação Seguradora – Gerente Comercial – Informações Seguradora

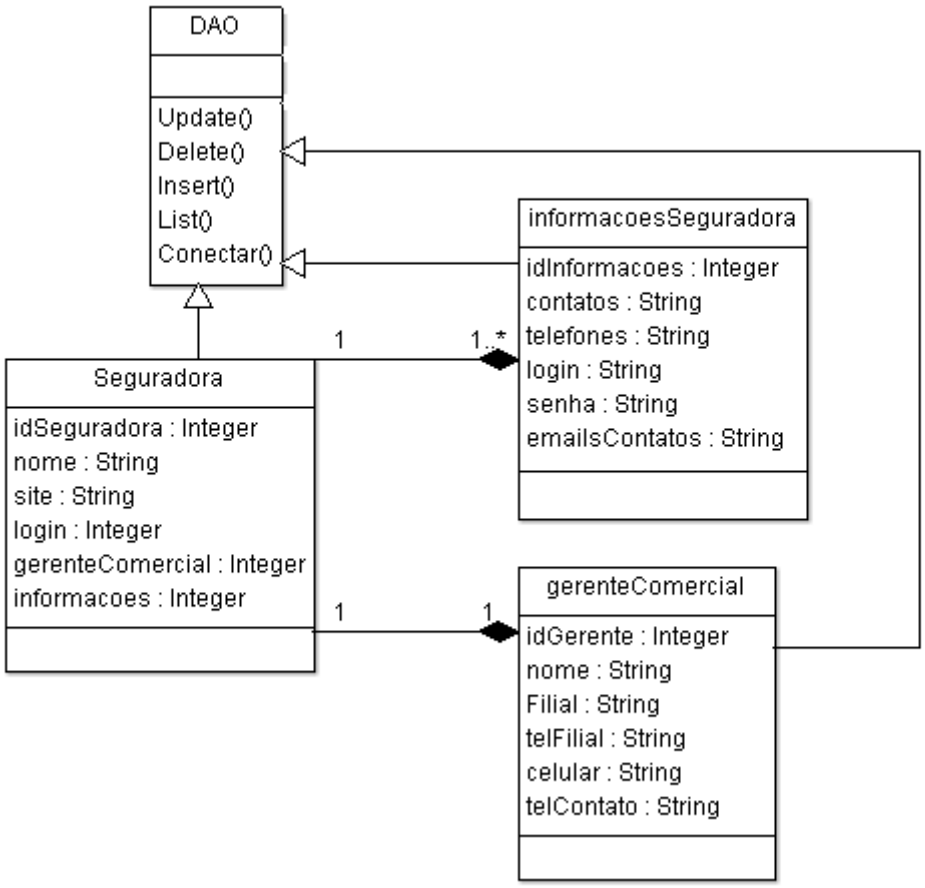

Figura 12 – Diagrama Classes Relação Seguradora – Gerente Comercial – Informações Seguradora

3.2 MODELAGEM WEB SERVICE

3.2.1 Diagrama Casos de Uso

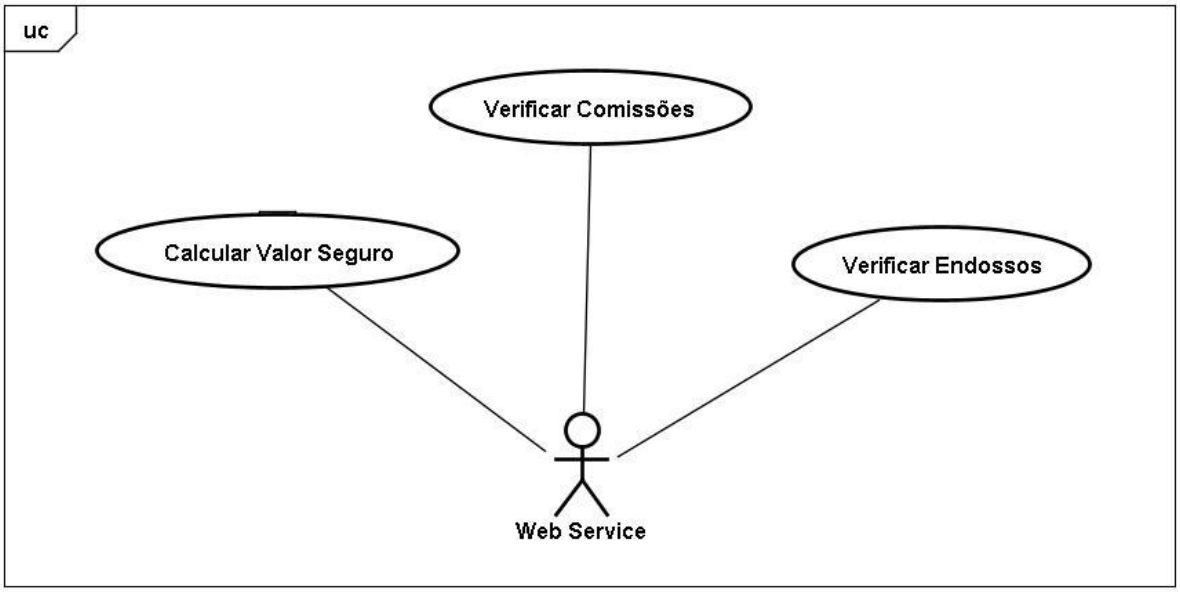

Figura 13 – Diagrama Casos de Uso Seguradora

3.2.2 Narrativas Casos de Uso

3.2.2.1 Calcular Valor Seguro

1. Finalidade
    a. Retornar ao Sistema Gerenciador de Corretoras o valor do Seguro.
2. Atores
    a. Web Service Seguradora
3. Pré-Requisito

a. Ter recebido dados do Sistema Gerenciador de Corretoras

4. Fluxo Principal
    a. O sistema calcula o valor do seguro de acordo com os dados fornecidos. (6a) (7a)
    b. O sistema retorna o valor do seguro.
5. Fluxo Alternativo
6. Fluxo de Exceção
    a. Os dados recebidos estão errados.
7. Testes
    a. O sistema verifica os parâmetros recebidos.

3.2.2.2 Verificar Comissões

1. Finalidade
    a. Retorna ao Sistema Gerenciador de Corretoras as comissões emitidas.
2. Atores
    a. Web Service.
3. Pré-Requisito
    a. Ter recebido os dados do Sistema Gerenciado de Corretoras.
4. Fluxo Principal
    a. O sistema procura por comissões emitidas. (6a) (7a)
    b. O sistema retorna as comissões emitidas ao Sistema Gerenciador de Corretoras.
5. Fluxo Alternativo
6. Fluxo de Exceção

a. Os dados recebidos estão errados.

7. Testes

a. O Sistema verifica os dados recebidos.

3.2.2.3 Verificar Endossos

1. Finalidade

a. Retorna ao Sistema Gerenciador de Corretoras os endossos emitidos.

2. Atores

a. Web Service.

3. Pré-Requisito

a. Ter recebido os dados do Sistema Gerenciado de Corretoras.

4. Fluxo Principal

a. O sistema procura por endossos emitidos. (6a) (7a)

b. O sistema retorna os endossos emitidos ao Sistema Gerenciador de Corretoras.

5. Fluxo Alternativo

6. Fluxo de Exceção

a. Os dados recebidos estão errados.

7. Testes

a. O Sistema verifica os dados recebidos.

4. RESULTADOS

Aqui irei mostrar alguns códigos do sistema gerenciador de corretora e dos Web Services desenvolvidos. Logo em seguida colocarei algumas imagens da aplicação final.

4.1 CLASSE AUTOMÓVEL

No código abaixo mostro a classe automóvel do pacote model, responsável por representar as entidades.

```
package Classe;
/** @author Tamires */
//declara todos atributos da classe automovel
public class Automovel {
    private int codautomovel;    private String marca;
    private String modelo;    private String chassi;
    private String placa;    private String anomodelo;
    private String anofabricacao;    private String portas;
    private String zerokm;    private String utilizacao;
    private String principalcondutor;    private String ceppernoite;
    private Proprietario proprietario =  new Proprietario();
    private Segurado segurado =  new Segurado();
    public Automovel(){}
//Getters e Setters
}
```

4.2 CLASSE AUTOMÓVEL DAO

No código abaixo mostro a classe responsável por realizar as ações no banco de dados.

```
package Dao;

//imports

//Classe responsável por realizar as operações de inserir, alterar, excluir e listar acessando //ao banco de dados.

/** @author Tamires */

public class AutomovelDAO {

   private Statement sql;    private Conexao con;

   public AutomovelDAO(){

      con =  new Conexao();

      try{

         sql =  con.getCon().createStatement();

      }catch(Exception e){

         e.printStackTrace();

      }

public void Inserir(Automovel a){

     try{

        sql.execute("insert into tabautomovel(modelo, chassi, placa, anomodelo, anofabricacao, portas,zerokm,utilizacao,principalcondutor,ceppernoite,proprietario,segurado)values('"+a.getModelo()+"','"+a.getChassi()+"','"+a.getPlaca()+"','"+a.getAnomodelo()+"','"+a.getAnofabricacao()+"','"+a.getPortas()+"','"+a.getZerokm()+"',''"+a.getUtilizacao()+"','"+a.getPrincipalcondutor()+"','"+a.getCeppernoite()+"','"+a.getProprietario().getCodproprietario()+"',''"+a.getSegurado().getCodsegurado()+"')");
FacesContext.getCurrentInstance().addMessage(null,newFacesMessage(FacesMessage.SEVERITY_INFO,"Sucesso","Automovel Inserido"));

      }catch(Exception e){

         e.printStackTrace();

      }finally{      con.fecharConexao();

       }}
```

```
public void Alterar(Automovel a){
    try{
      sql.execute("update tabautomovel set modelo='"+a.getModelo()+"', chassi='"+a.getChassi()+"',
"placa='"+a.getPlaca()+"', anomodelo='"+a.getModelo()+"', anofabricacao='"+a.getAnofabricacao()+"',
"portas='"+a.getPortas()+"', zerokm='"+a.getZerokm()+"', utilizacao='"+a.getUtilizacao()+"',
"principalcondutor='"+a.getPrincipalcondutor()+"', ceppernoite='"+a.getCeppernoite()+"',
"proprietario='"+a.getProprietario().getCodproprietario()+"',
segurado='"+a.getSegurado().getCodsegurado()+"' where codautomovel="+a.getCodautomovel());
      FacesContext.getCurrentInstance().addMessage(null, new
FacesMessage(FacesMessage.SEVERITY_INFO, "Sucesso", "Automovel Alterado"));
    }catch(Exception e){
      e.printStackTrace();
    }finally{
      con.fecharConexao();
    }
  }

  public void Deletar(Automovel a){
    try{
      sql.execute("delete from tabautomovel where codautomovel="+a.getCodautomovel());
      FacesContext.getCurrentInstance().addMessage(null, new
FacesMessage(FacesMessage.SEVERITY_INFO, "Sucesso", "Automovel Deletado"));
    }catch(Exception e){
      e.printStackTrace();
    }finally{
      con.fecharConexao();
    }
  }
```

4.3 CLASSE AUTOMOVELBEAN

```
package Bean;

//imports

/** @author Tamires */

@ManagedBean

@RequestScoped

//Classe responsável por controlar todo o fluxo de informação que passa pelo sistema.

public class AutomovelBean {

   private Automovel a =  new Automovel();    private AutomovelDAO adao =  new AutomovelDAO();

   private List<Automovel> alist = new ArrayList<Automovel>();

   public AutomovelBean(){

                adao =  new AutomovelDAO();       alist =  adao.listautomovel();

   }

   public void inserir(){

                adao =  new AutomovelDAO();       adao.Inserir(a);

  }

  public void alterar(){

                adao =  new AutomovelDAO();       adao.Alterar(a);

  }

  public void deletar(){

                adao =  new AutomovelDAO();       adao.Deletar(a);

 }

          //Getters e Setters

}
```

4.4 CLASSE COTACAOSEGURADORA

```
package ws.business;

/** @author Tamires */

//Classe responsável por calcular o valor do seguro de acordo com os dados recebidos

public class CotacaoSeguradora {

   public Integer valorSeguro(String automovel, String zerokm, String utilizacao, String ceppernoite, String sexo,
   int idadeprincipalcondutor, int fatorajuste){

      int vautomovel=0, vceppernoite=0, vsexo=0,vidade=0,vfator=0,vzerokm=0,vutilizacao=0;

      String a = automovel;

       if(automovel.equals("Camaro")){

          vautomovel = 2 * 2003;

      }else if(automovel.equals("Gol")){

          vautomovel = 2 * 380;

      }else if(automovel.equals("Novo Fusca")){

          vautomovel = 2 * 800;

      }else if(automovel.equals("Cross Fox")){

          vautomovel = 2 * 432;

      }else if(automovel.equals("Ferrari")){

          vautomovel = 2 * 10000;

      }

      if(zerokm.equals("Sim")){

          vzerokm = 10;

      }else vzerokm = 20;

      if(utilizacao.equals("Particular")){

          vutilizacao = 10;

      }else if(utilizacao.equals("Taxi")){

          vutilizacao = 50;

       }else vutilizacao = 20;
```

```
      if(ceppernoite.equals("19800000")){
          vceppernoite = 10;
       }else vceppernoite=20;
       if(sexo.equals("Feminino")){
          vsexo = 10;
       }else if(sexo.equals("Masculino")){
          vsexo = 20;
       }
       if(idadeprincipalcondutor>25){
          vidade = 10;
       }else vidade = 20;
       if(fatorajuste == 105){
          vfator = 100;
       }else if(fatorajuste == 110){
          vfator = 200;
       }else vfator = 50;
      return vautomovel+vceppernoite+vfator+vidade+vsexo+vutilizacao+vzerokm;
   }
}
```

4.5 WEB SERVICE SEGURADORA

```
package ws;

//Imports


/** @author TAMIRES */

@WebService(serviceName = "WebServiceSeguradora")

//Classe responsável por passar os dados recebidos para a classe CotacaoSeguradora

public class WebServiceSeguradora {


    /**

     * Operação de serviço web

     */

    @WebMethod(operationName = "cotacao")

    public Integer cotacao(@WebParam(name = "automovel") String automovel, @WebParam(name = "zerokm") String zerokm, @WebParam(name = "utilizacao") String utilizacao, @WebParam(name = "ceppernoite") String ceppernoite, @WebParam(name = "sexo") String sexo, @WebParam(name = "idadeprincipalcondutor") int idadeprincipalcondutor, @WebParam(name = "fatorajuste") int fatorajuste) {

        CotacaoSeguradora c = new CotacaoSeguradora();


System.out.println(automovel+"/n"+ceppernoite+"/n"+sexo+"/n"+idadeprincipalcondutor+"/n"+fatorajuste+"/n");

        return c.valorSeguro(automovel, zerokm, utilizacao, ceppernoite, sexo, idadeprincipalcondutor, fatorajuste);

    }

}
```

4.6 APLICAÇÃO FINAL

Abaixo colocarei algumas imagens demonstrando como ficou a aplicaçação final deste trabalho.

Figura 14 – Tela Inicial do Sistema

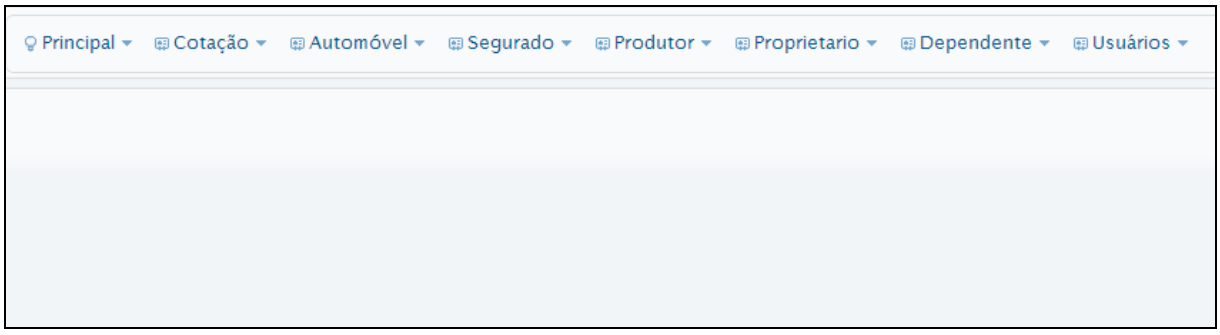

Figura 15 – Tela Menu Principal do Sistema

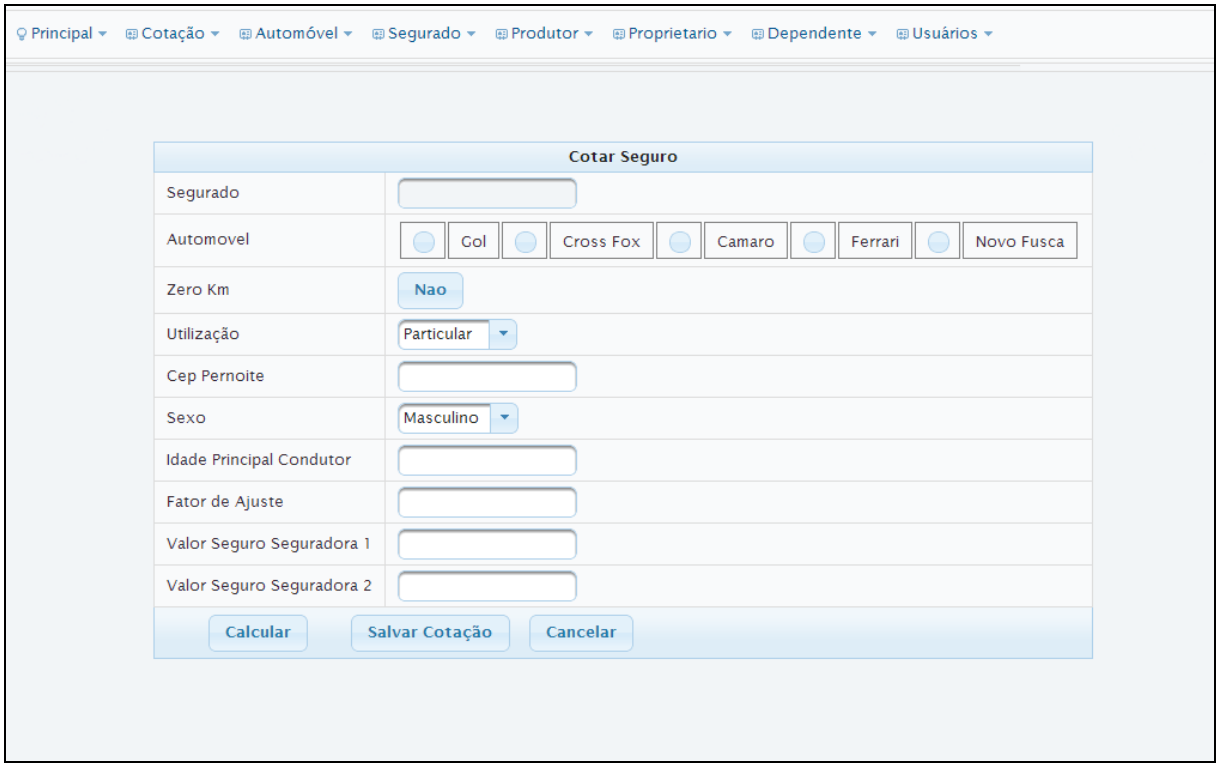

Figura 16 – Tela em que ocorre o múltiplo cálculo de seguro

5. CONCLUSÃO

Ao final do trabalho espera-se como resultado a obtenção de uma ferramenta capaz de auxiliar na tomada de decisão de segurado e corretor em relação à melhor companhia para o seguro de tal.

Cumprido este objetivo, considerado como principal neste trabalho, existe a expectativa de crescimento e desenvolvimento pessoal, obtida por intermédio do uso das ferramentas descritas nas seções anteriores, em um ambiente real, que certamente propiciará experiência e desafios úteis em minha trajetória profissional.

5.1 TRABALHOS FUTUROS

Como possíveis trabalhos futuros, pode-se apontar:

- Criação e implementação de um Web Service em um dispositivo Móvel.
- Implementar outros sistemas demonstrando a versatilidade do uso de Web Services.

6. REFERÊNCIAS

AMORIM, Simone da Silva. **A Tecnologia Web Services e sua Aplicação num Sistema de Gerência de Telecomunicações.** 2004. 91 f. Dissertação (Mestrado) - Universidade Estadual de Campinas, São Luís, 2004.

BECKER, Aleksader Knabben; CLARO, Daniela Barreiro; SOBRAL, João Bosco. **Web Services e XML:** Um Novo Paradigma da Computação Distribuída. Universidade Federal de Santa Catarina. Disponível em: <http://homes.dcc.ufba.br/~dclaro/download/ArtigoWebServices.pdf>. Acesso em: 01 out. 2012.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML: guia do usuário.** 2. ed. Rio de Janeiro: Elsevier, 2005. Disponível em: <http://books.google.com.br/books?hl=pt-BR&lr=&id=ddWqxcDKGF8C&oi=fnd&pg=PR13&dq=UML&ots=fcAHnchMJQ&sig=Q5sWuzx9cpxndFYTR2aLYhzsAdU>. Acesso em: 28 out. 2012.

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. **Sistemas Distribuídos:** Conceitos e Projetos. 4. ed. Porto Alegre: Bookman, 2007. 784 p. Disponível em: <http://books.google.com.br/books?id=KSZ1rIRWmUoC&pg=PA40&dq=sistemas+distribuidos&hl=pt-BR&sa=X&ei=XRiNUMCeDIjY8gTk8IHIBA&ved=0CDAQ6AEwAA#v=onepage&q=sistemas%20distribuidos&f=false>. Acesso em: 28 out. 2012.

FREIRE, Herval. **Web Services: A Nova Arquitetura da Internet:** Artigo publicado na revista Developer's Magazine número 73, Setembro de 2002. Disponível em: <http://www.cnnt.com.br/files/webservices.pdf>. Acesso em: 20 set. 2012.

GIRARDI, Reubem Alexandre D'almeida.**Framework para coordenação e mediação de Web Services modelado como Learning Objects para ambientes de aprendizado na Web.** 2004. 111 f. Dissertação (Mestrado) - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2004.

GOMES, Lucas de Souza Reis. **Desenvolvimento de Aplicações de Dispositivos Móveis com J2ME e Integração com Web Services.** 2005. 138 f. Trabalho (Bacharelado) - Universidade Federal de Santa Catarina, Florianópolis, 2005.

GOULART, Ademir. **Sistemas Distribuídos e Comunicação em Grupo.** Disponível em: <http://www.goulart.pro.br/sd/arquivos/APOSTILA_AdemirGoulart_SD_CGv10.PDF>. Acesso em: 28 out. 2012.

HENDRICKS, Mack et al. Profissional Java Web Services. Rio de Janeiro: Alta Books, 2002.

I-WEB. **Web Services.** Disponível em: <http://iweb.com.br/iweb/pdfs/20031008-webservices-01.pdf>. Acesso em: 15 set. 2012.

MACIEL, Diego Rabelo. **Implementação de um Disco Virtual Seguro Baseado em Web Services.** 2007. 77 f. Monografia (Bacharelado) - Departamento de Centro de Ciências Exatas e Tecnologia, Universidade Federal do Maranhão, São Luís, 2007.

MATTOS, Lucy Moreira; RAMOS, Maria Heliete Alves. **Teoria geral de seguro**. Rio de Janeiro: FUNENSEG, 1990.

MENÉNDEZ, Andrés Ignácio Martínez. **Uma Ferrramenta de Apoio ao desenvolvimento de Web Services.** 2002. 97 f. Dissertação (Pós Graduação) - Departamento de Centro de Ciências e Tecnologia, Universidade Federal de Campina Grande, Campina Grande, 2002.

PULIDO, Aline Fernanda Santos. **Da Responsabilidade Civil do Corretor de Seguros**. 2006. 53p. Trabalho de Conclusão de Curso – Faculdades Integradas "Antônio Eufrásio de Toledo", São Paulo, Presidente Prudente, 2006.

QUEIROZ, Gilberto Ribeiro de. **UML: Visão Geral.** Disponível em: <http://www.dpi.inpe.br/~gribeiro/apresentacoes/uml_2008_02_29.pdf>. Acesso em: 24 jun. 2012.

SAMPAIO, Cleuton. **SOA e Web Services em Java**. 1. ed. Rio de Janeiro: Editora Brasport,2006