UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE SISTEMAS DE INFORMAÇÃO – BACHARELADO

MÓDULOS INTEGRADOS DE REGISTRO DE ABASTECIMENTO DE VEÍCULOS PARA ÓRGÃOS PÚBLICOS

GABRIEL VIEIRA

BLUMENAU
2011

2011/2-14

GABRIEL VIEIRA

MÓDULOS INTEGRADOS DE REGISTRO DE ABASTECIMENTO DE VEÍCULOS PARA ÓRGÃOS PÚBLICOS

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

Prof. Jacques Robert Heckmann, Mestre - Orientador

BLUMENAU
2011

2011/2-14

MÓDULOS INTEGRADOS DE REGISTRO DE ABASTECIMENTO DE VEÍCULOS PARA ÓRGÃOS PÚBLICOS

Por

GABRIEL VIEIRA

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

	_____
Presidente:	Prof. Jacques Robert Heckmann, Mestre - Orientador, FURB
	_____
Membro:	Prof. Marcel Hugo, Mestre, FURB
	_____
Membro:	Prof. Wilson Pedro Carli, Mestre, FURB

Blumenau, 6 de dezembro de 2011.

Dedico este trabalho a todos os amigos, especialmente aqueles que me ajudaram diretamente na realização deste.

AGRADECIMENTOS

À minha mãe, que compreendeu minha ausência.

À minha namorada, pela paciência, empurrões e cobranças.

Aos amigos e colegas que contribuíram com conhecimentos.

À Pública Informática Ltda. e seus funcionários que me apoiaram e incentivaram durante a graduação.

Ao meu orientador, Jacques Robert Heckmann, por ter acreditado na conclusão deste trabalho.

Não há problema que não possa ser solucionado pela paciência.

Chico Xavier

RESUMO

Este trabalho apresenta a realização de um novo processo de abastecimentos de veículos em entidades públicas, utilizando a *internet* como forma de transferência de dados entre os envolvidos. O trabalho integra-se a dois sistemas da empresa Pública Informática Ltda. e tem como objetivo automatizar a comunicação entre o órgão público e o fornecedor de combustível, registrando todas as movimentações de abastecimento no sistema de gerenciamento de veículos. A dificuldade de elaborar o processo foi o isolamento das informações entre as entidades da cidade, desta maneira foi necessário criar um servidor de abastecimentos na prefeitura para centralizar as informações. A prefeitura encarrega-se de transmitir estes abastecimentos a outras entidades utilizando o módulo transmissor, e os órgãos recebem o abastecimento através do módulo *web service*. Ambos os módulos foram desenvolvidos em Java utilizando o *framework* Hibernate. O terceiro módulo desenvolvido foi o cadastro de abastecimento, que é integrado ao sistema de comunicação com o cliente e fornecedor utilizando a linguagem PHP.

Palavras-chave: *Web Service*. Abastecimento. Órgão público.

ABSTRACT

This paper presents the realization of a new process for vehicles supply in government entities, using the internet as a means of transfer data between those involved. The work integrates two systems of Pública Informática Ltda. company and aims to automate communication between the government agency and the fuel supplier, recording all movements in the vehicles supply management system. The difficulty of developing the process was the isolation of information between the entities of the city, thus it was necessary to create a server supplies the city to centralize information. The city hall is responsible for transmit these supplies to other entities using the transmitter module, and public agencies receive supplies through the web service module. Both modules have been developed in Java using the Hibernate framework. The third module developed was the supply register which is integrated into the communication system between customers and suppliers using the PHP language.

Key-words: Web Service. Supplyment. Government entities.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

API - *Application Programming Interface*

EA - Enterprise Architect

HTTP - *HyperText Transfer Protocol*

Java EE - Java *Enterprise Edition*

Java SE - Java *Standard Edition*

LGPL - *Lesser General Public License*

QR Code - *Quick Response Code*

REST - *REpresentational State Tranfer*

RPC - *Remote Procedure Calls*

SOAP - *Simple Object Access Protocol*

UML - *Unified Modeling Language*

W3C - *World Wide Web Consortium*

WSDL - *Web Service Description Language*

XML - *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Segundo o Tribunal de Contas de Santa Catarina (2011), a administração de uma cidade é baseada em leis que determinam todos os processos que devem ocorrer dentro de um órgão público. A complexidade destas leis tornou-se um grande motivo para a informatização das áreas, como um meio de automatizar e fiscalizar a administração. O tribunal de contas é responsável pela fiscalização do cumprimento destas leis, e possui um software que integra todas as informações dos órgãos.

A empresa Pública Informática, que desenvolve softwares específicos para órgãos públicos, possui sistemas que foram desenvolvidos conforme essa legislação. Em específico, esses softwares controlam as compras públicas, frotas, contabilidade e comunicação com os fornecedores pela *internet*. O sistema de frotas é responsável pelo controle dos abastecimentos dos veículos de responsabilidade de cada um dos órgãos clientes.

Segundo uma pesquisa realizada pela própria empresa fabricante do software, o nível de ocorrências destes abastecimentos vem aumentando a cada dia e em decorrência deste nível de informação, não se consegue atualizar todos os abastecimentos no sistema de frotas.

Porém, ainda hoje, alguns clientes da Pública Informática autorizam abastecimentos de veículos por meio de um formulário em papel emitido pelo sistema de gerenciamento de veículos. O fornecedor faz o registro de todas as informações do abastecimento neste papel e, no final do mês, esse é devolvido ao órgão público.

1.1 OBJETIVOS DO TRABALHO

O objetivo principal do trabalho é apresentar um novo processo de controle no abastecimento de veículos com o auxilio de três módulos que permitam efetuar os registros dos abastecimentos utilizando a *internet*.

Os objetivos específicos do trabalho proposto são:

a) elaborar uma interface *on-line* de registro de abastecimento de veículos;
b) desenvolver estrutura que se utilize de *web services* para comunicação entre a prefeitura, que centraliza todas as informações, e todas as entidades municipais;

c) elaborar um algoritmo de geração de informação por código de barras para compor a requisição de abastecimento;
d) elaborar uma consulta para o fornecedor consultar todos os abastecimentos realizados;
e) criar mecanismos para registrar (*log*) as ações entre o módulo de transmissão e o módulo receptor *web service*;
f) criar um relatório das mensagens de retorno do *web service* ao transmissor.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está disposto em quatro capítulos. No primeiro capítulo, é apresentada a introdução do assunto, os objetivos a serem alcançados com o desenvolvimento e a estrutura do trabalho.

O segundo capítulo apresenta a fundamentação teórica sobre conceitos de *web service*, Hibernate, Quartz, Axis, QR Code, sistema atual e os trabalhos correlatos.

No terceiro capítulo têm-se a descrição do ciclo de desenvolvimento do sistema, detalhes sobre a especificação e modelagem, técnicas e ferramentas utilizadas e a operacionalidade do sistema com os resultados e discussões.

No quarto capítulo apresenta-se a conclusão sobre os objetivos alcançados e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda assuntos como *web service*, Hibernate, Quartz, Axis, QR Code, sistema atual, além de trabalhos correlatos.

2.1 WEB SERVICE

O *web service* é uma ferramenta que disponibiliza serviços para clientes através de uma rede, utilizando um protocolo para troca de mensagens. Ele se comunica com outras aplicações utilizando um documento conhecido como *Web Service Description Language* (WSDL), em formato *Extensible Markup Language* (XML) e utiliza um protocolo para troca de mensagens chamado *Simple Object Access Protocol* (SOAP). A comunicação com o *web service* deve ser realizada por um *software* cliente, que possui o endereço e as configurações do *web service* (SAMPAIO, 2006).

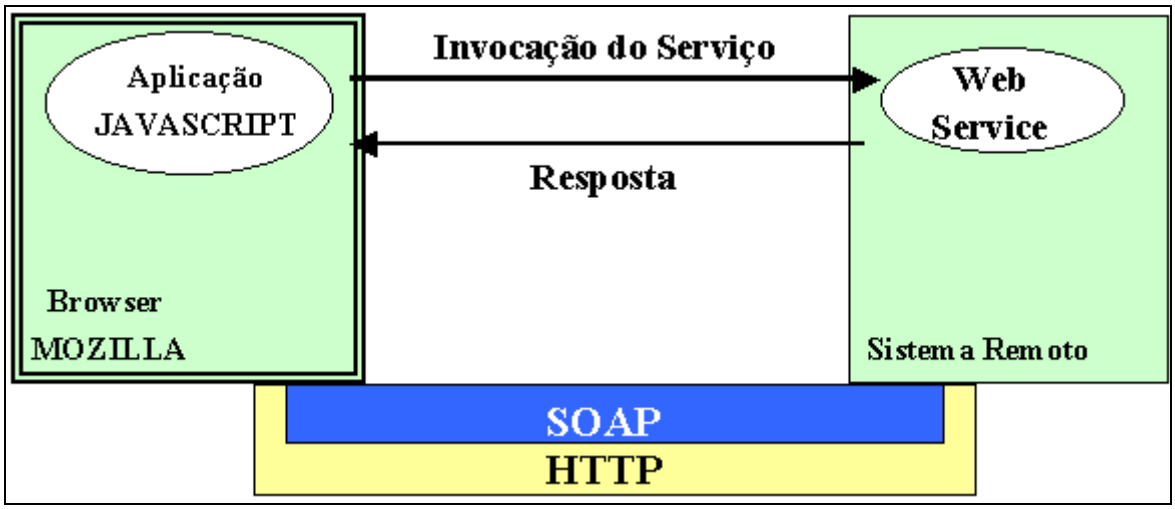
Portanto, o assunto a ser tratado divide-se em utilização de *web services*, SOAP e WSDL.

2.1.1 Utilização de *web services*

Cunha (2002) cita três cenários de utilização de *web services*, dividindo-os em acesso a *web services* através do navegador, acesso a *web services* pelo cliente e acesso a *web services* no lado do servidor.

2.1.1.1 Acesso a w*eb services* através do navegador

O primeiro modelo, conforme mostra a Figura 1, é utilizado para aplicações em que uma página *web* necessita consultar informações em outro servidor.

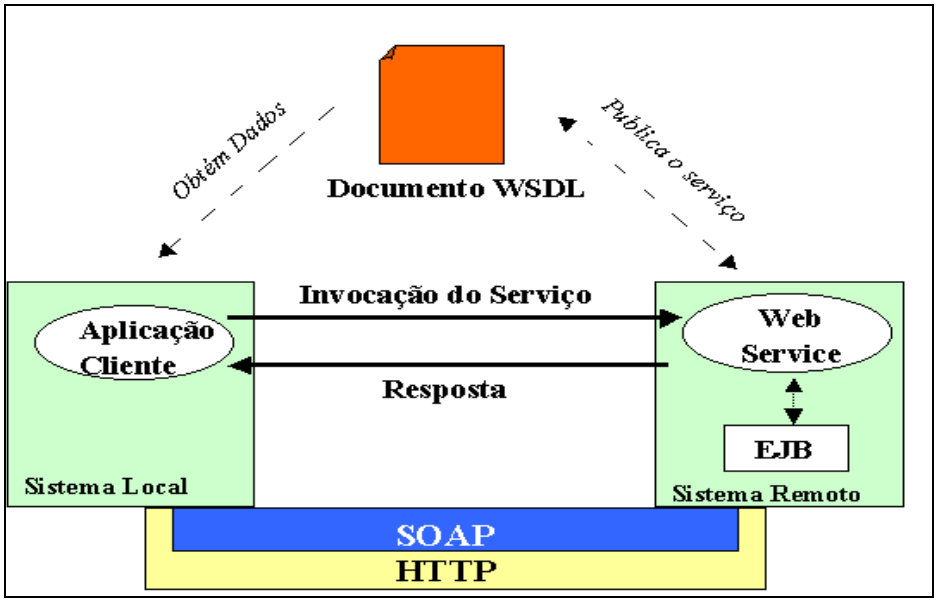


Fonte: Cunha (2002).

Figura 1 – Acesso a *web service* através do navegador

2.1.1.2 Acesso a *web services* pelo cliente

No modelo, conforme mostra a Figura 2, tem-se uma aplicação cliente desenvolvida em qualquer linguagem de programação que envia chamadas via protocolo SOAP para o *web service*. O *web service* recebe a chamada, processa o que foi requisitado e envia a resposta para o cliente. Toda a comunicação é feita pelo protocolo de mensagens SOAP que utiliza o protocolo HTTP para transporte da mensagem.

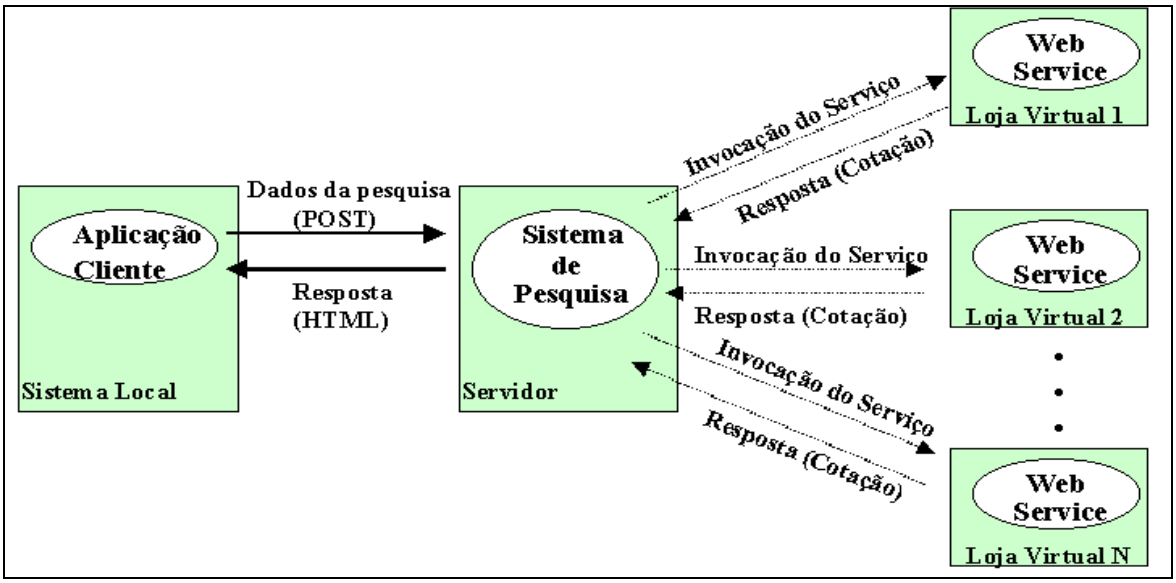


Fonte: Cunha (2002).

Figura 2 – Aplicação cliente acessando diretamente um *web service*

2.1.1.3 Acesso a *web services* no lado do servidor

Por fim, neste último modelo, conforme mostra a Figura 3, tem-se uma aplicação cliente acessando uma aplicação no servidor. Esta, por sua vez, faz chamadas a diversos *web services* em diferentes máquinas, processa as respostas destes serviços e envia a resposta ao cliente.



Fonte: Cunha (2002).

Figura 3 - Aplicação no servidor acessando diversos *web service*s

2.1.2 WSDL

O WSDL é definido como um documento escrito em XML baseado em um esquema que descreve todas as informações relativas à comunicação entre o cliente e o *web service*, especifica como acessá-lo e quais as operações ou métodos estão disponíveis utilizando uma estrutura já definida.

Conforme Sampaio (2006), para criar um WSDL é necessário:

a) definir os tipos de dados que serão utilizados nas mensagens;
b) definir as mensagens compostas pelos tipos que foram definidos;
c) definir os *Port types*, que são compostos pelas operações suportadas por cada porta, juntamente com suas mensagens de entrada e saída;
d) criar o *Binding* que associa a operação com o protocolo de transporte e estilo de

mensagem;

e) associar cada porta com um *Binding* definido e um endereço de rede para acesso ao serviço.

Segundo a *World Wide Web Consortium* (W3C), um arquivo WSDL segue uma sintaxe definida, a qual organiza o documento para que o *web service* possa interpretá-lo.

> Basicamente, quando o cliente deseja enviar uma mensagem para um determinado *web service*, ele obtém a descrição do serviço (através da localização do respectivo documento WSDL), e em seguida constrói a mensagem, passando os tipos de dados corretos (parâmetros, etc.) de acordo com a definição encontrada no documento. Em seguida, a mensagem é enviada para o endereço onde o serviço está localizado, a fim de que possa ser processada. O *web service*, quando recebe esta mensagem valida-a conforme as informações contidas no documento WSDL. A partir de então, o serviço remoto sabe como tratar a mensagem, sabe como processá-la (possivelmente enviando-a para outro programa) e como montar a resposta ao cliente. (CUNHA, 2002).

2.1.3 SOAP

O SOAP foi criado para a utilização de métodos através da *internet*. Surgiu em uma época em que se tinham poucas alternativas de comunicação. O protocolo de mensagens foi criado utilizando parâmetros flexíveis e fazendo o uso da padronização XML (SAMPAIO, 2006).

Ele foi projetado para invocar aplicações remotas através de *Remote Procedure Calls* (RPC) ou trocas de mensagens, em um ambiente independente de plataforma e linguagem de programação. SOAP é, portanto, um padrão para utilizar-se com *web services*. Desta forma, pretende-se garantir a interoperabilidade e intercomunicação entre diferentes sistemas, através da utilização de uma linguagem (XML) e mecanismo de transporte (HTTP) padrões.

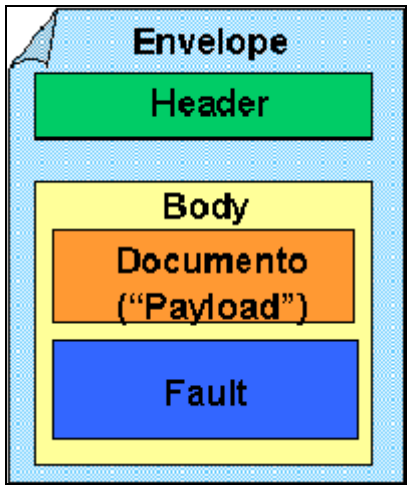
Segundo Cunha (2002), uma mensagem SOAP consiste basicamente nos seguintes elementos:

a) envelope: obrigatório em toda mensagem SOAP. É o elemento raiz do documento XML. O envelope pode conter declarações de *namespaces* e também atributos adicionais como o que define o estilo de codificação (*encoding style*). Um *encoding style* define como os dados são representados no documento XML;

b) *header*: é um cabeçalho opcional. Ele carrega informações adicionais sobre a mensagem. Quando utilizado, o *header* deve ser o primeiro elemento do Envelope;

c) *body*: obrigatório em toda mensagem SOAP. Este elemento contém o *payload* (informação a ser transportada para o seu destino final). O elemento *body* pode conter um elemento opcional *fault*, usado para carregar mensagens de status e erros retornadas pelos "nós" ao processarem a mensagem.

A Figura 4 demonstra a estrutura de um protocolo SOAP.



Fonte: Cunha (2002).

Figura 4 - Estrutura de uma mensagem SOAP

Hoje o protocolo SOAP é um protocolo completo de troca de mensagens, tornando-se um padrão recomendado pela W3C (SAMPAIO, 2006).

2.2 HIBERNATE

Hibernate é um *framework* que visa solucionar problemas gerados com a persistência manual de dados em Java. O objetivo do *framework* é fazer o mapeamento entre classes e tabelas de um banco de dados, deixando o programador livre para desenvolver as regras de negócio (BAUER; KING, 2007, p.4).

Segundo Bauer e King (2007, p.25), a principal característica do Hibernate é o mapeamento objeto/relacional, cujo objetivo é a persistência dos dados de um objeto diretamente em um banco de dados relacional, utilizando metadados. Os metadados são dicionários que descrevem como este mapeamento deve ser realizado.

Bauer e King (2007, p.25) apresentam algumas vantagens ao utilizar o Hibernate em um projeto. São elas:

a) produtividade: o programador concentra-se apenas nas regras de negócio;
b) manutenibilidade: reduz a quantidade de linhas de código fonte no sistema, pois enfatiza a lógica de negócio e não o acesso aos dados, tornando o sistema mais legível;
c) performance: agilidade para otimizar de maneira geral todas as rotinas de acesso a dados, quando comparado ao método manual;
d) independência de fornecedor: o Hibernate abstrai a utilização do dialeto SQL, isto proporciona a aplicação um bom nível de portabilidade para outros bancos de dados.

Segundo Fernandes e Lima (2007, p.19), no início o Hibernate carregava e armazenava objetos de classes persistentes utilizado arquivos XML. Todo o mapeamento era feito neste arquivo, cujo objetivo era efetuar a ligação entre um atributo e um campo do banco de dados e sua respectiva classe/tabela. Com o surgimento das anotações no Java 5.0, tornou-se possível substituir os arquivos XML para o mapeamento objeto relacional.

Fernandes e Lima (2007, p.16) explicam que as anotações são definidas no código fonte da entidade a ser persistida, e começam com um símbolo @ (arroba) seguido do nome da anotação. Essa anotação é ignorada pelo compilador. Com as anotações é possível anotar um atributo ou uma classe, dando-lhe algumas características para o mapeamento, conforme mostra a Figura 5.

```java
package br.com.jeebrasil.hibernate.anotacoes.dominio;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

//Anotação que informa que a classe mapeada é persistente
@Entity
//Informando nome e esquema da tabela mapeada
@Table(name="aluno", schema="anotacoes")
public class Aluno {

	//Definição da chave primária
	@Id
	//Definição do mecanismo de definição da chave primária
	@GeneratedValue(strategy = GenerationType.SEQUENCE)
	//Informa o nome da coluna mapeada para o atributo
	@Column(name="id_aluno")
	private int id;
	private int matricula;
	private String nome;
	private long cpf;

	public void Aluno(){}

	//Métodos getters e setters
	//...
}
```

Fonte: Lima, Fernandes (2007, p.16).

Figura 5 - Exemplo de código fonte usando Hibernate com anotação

Segundo Lemos (2011), o Hibernate pretende retirar do desenvolvedor 95% das tarefas mais comuns da persistência dos dados, sendo mais indicado para aplicação orientada a objetos, cuja regra de negócio é implementada na camada de aplicação, sendo ineficaz quando as regras de negócio são declaradas em funções dentro de um banco de dados utilizando

Stored Procedures.

Hibernate é mantido pela Jboss sob licença da *Lesser General Public License* (LGPL), sendo um dos *frameworks* de persistência de dados mais utilizado e documentado atualmente (LEMOS, 2011).

2.3 QUARTZ

Conforme Lemke (2011), o Quartz é um *framework* de agendamento de tarefas empresarial de código aberto baseado em Java. Este *framework* pode ser integrado com qualquer aplicação em Java EE ou Java SE, construída para trabalhar com aplicações desde pequenas até de grande porte.

Lemke (2011) comenta que o Quartz oferece rapidez, flexibilidade e um alto grau de confiabilidade a seus utilizadores, podendo-se trabalhar com múltiplas tarefas apenas registrando um agendamento, utilizando uma classe Java, e instalando a aplicação em um servidor Apache Tomcat.

Conforme Coelho (2008, p.8), o Quartz possui dois elementos que são responsáveis pelo seu funcionamento:

a) *jobs*: é uma classe Java que contém instruções que serão executadas;
b) *triggers*: são gatilhos que disparam a execução das *Jobs*.

Coelho (2008, p.9) explica que existem dois tipos de *triggers*, a *SimpleTrigger* e a *CronTrigger*. A *SimpleTrigger* é utilizada quando o agendamento não possui complexidade, pois ela utiliza uma expressão de agendamento bem simples. Já a *CronTrigger* é utilizada para agendamentos que necessitam de complexidade, como por exemplo uma tarefa que precisa de muitas repetições em um dia, semana, mês ou ano.

O Quartz é indicado para quem precisa de um agendamento simples e rápido sem necessitar de muitos recursos do sistema operacional e de hardware, utilizando uma forma flexível de agendamento sem sacrificar a simplicidade (COELHO, 2008, p.10).

Quartz Scheduler está licenciado sob a Apache 2.0 como código aberto, e está disponível para *download* a partir do site do projeto (LEMKE, 2011).

2.4 AXIS

Segundo Destro (2006), a principal função da ferramenta Axis é a criação e a publicação de um *web service* de maneira simplificada, gerando o *web service* de forma automática a partir de uma classe Java. O processo de geração resulta em um arquivo WSDL, contendo todas as informações de comunicação do cliente com o servidor. Para que seja possível a geração, existem ferramentas e *plugins* disponíveis para utilização em ambientes de programação.

Segundo Apache Software Foundation (2009), o Axis2 é uma evolução do Axis, com um novo conceito de implementação, a qual foi totalmente reestruturada. Essa renovação permitiu aos desenvolvedores corrigir todos os erros que foram cometidos no Axis e adicionar várias outras funcionalidades para a nova versão. As seguintes tarefas estão liberadas no Axis2:

a) enviar mensagens SOAP;
b) receber e processar mensagens SOAP;
c) criar um *web service* a partir de uma classe Java simples;
d) criar classes de implementação para o servidor e cliente a partir de um WSDL;
e) facilidade para recuperar o WSDL de um serviço;
f) enviar e receber mensagens SOAP com anexos;
g) criar ou utilizar um *web service* baseados em REST.

O Apache Axis e Axis2 são projetos de código aberto, mantidos pela Apache Software Foundation.

2.5 QR CODE

Segundo Wilbert (2010), o QR Code é um código de barras bi-dimensional, criado em 1994 pela empresa japonesa Denso-Wave, cujo objetivo principal era a capacidade de ser interpretado rapidamente. Por este motivo foi nomeado *Quick Response Code* (QR Code).

Prass (2011) comenta que o QR Code (exemplo mostrado na Figura 6) possui grande capacidade de armazenamento de dados, podendo conter diversos conteúdos, conforme os tipos abaixo:

a) numéricos: 7.089 caracteres;
b) alfa-numérico: 4.296 caracteres;
c) binário (8 bits): 2.953 caracteres;
d) kanji/kana (alfabeto japonês): 1.817 caracteres.

Segundo Prass (2011)**,** com a flexibilidade do conteúdo e um grande poder de armazenamento, o QR Code está sendo amplamente utilizado em empresas de publicidade para divulgação de marcas e produtos, utilizando a câmera do celular e um software que interpreta a imagem para capturar o código e rapidamente abrir a página da internet, como mostra a Figura 6.

Fonte: Prass (2011).

Figura 6 - Exemplo de código de barras QR Code

O Banco do Brasil, com o objetivo de facilitar a digitação do código de barras para pagamento de boletos, desenvolveu um leitor de QR Code gratuito para qualquer usuário através da internet, onde as informações são capturadas automaticamente pela câmera de um *smartphone* e são inseridas no aplicativo, bastando o cliente confirmar o pagamento (PRIMOS, 2011).

2.6 SISTEMA ATUAL

Para manter um bom nível de satisfação dos clientes com os produtos oferecidos, a empresa Pública Informática realiza visitas de consultores periodicamente, entrevistando alguns funcionários que utilizam as ferramentas da empresa. O objetivo da entrevista é garantir a satisfação do cliente com relação aos softwares fornecidos por ela. Em algumas dessas visitas, observou-se que os funcionários que efetuam o controle dos pagamentos de fornecedores enfrentam dificuldades para os pagamentos de fornecimento de combustível. O cliente indica que os abastecimentos no local ocorrem sem nenhum controle, na maioria das vezes, escrita a caneta no momento do abastecimento.

Além disso, a empresa Pública Informática fez a pesquisa na base de dados do sistema de gerenciamento de frotas de vários clientes e concluiu que os abastecimentos não estão sendo registrados no sistema.

Todos os sistemas desenvolvidos pela empresa utilizam uma única base de dados, a do *Firebird*. Desta forma, os dados são integrados em tempo real. A Pública Informática possui quatro sistemas fundamentais para si, os quais são:

a) o sistema de compras públicas, chamado PCP, que controla os processos que estão definidos na Lei N.º 8.666 e N.º 10.520, no qual são realizadas as requisições dos materiais, confecção da minuta do edital, as cotações dos fornecedores e a homologação do vencedor;
b) o sistema de contabilidade, intitulado de COP, é baseado na Lei N.º 4.320 e é responsável pelo registro, controle, demonstração e execução dos orçamentos;
c) o sistema de gerenciamento de frotas, denominado SAV, mantém o cadastro de veículos, bem como todos os registros de manutenção e abastecimento;
d) o sistema NET-Fornecedores, é utilizado para comunicação entre o fornecedor e o órgão.

A integração entre o sistema PCP e o SAV fica evidente através da autorização de abastecimento, na qual o usuário indica a licitação de abastecimento. Já a integração entre o PCP e o COP fica evidente através da emissão de pagamento de empenho, pelo qual a contabilidade paga o que foi licitado. E o sistema NET-Fornecedores é integrado com o PCP, e é utilizado pelo fornecedor para consulta de cotações realizadas.

Para realização dos abastecimentos, o órgão faz o lançamento de uma licitação. Quando a empresa vencedora da licitação é definida, o usuário emite a requisição de

abastecimento no SAV, quando o motorista vai até o fornecedor e abastece o veículo. Em uma data determinada, o órgão recolhe os formulários de requisições e os revisa para efetuar o pagamento no COP. O fluxo atual é demonstrado pelo diagrama de atividades conforme a Figura 7.

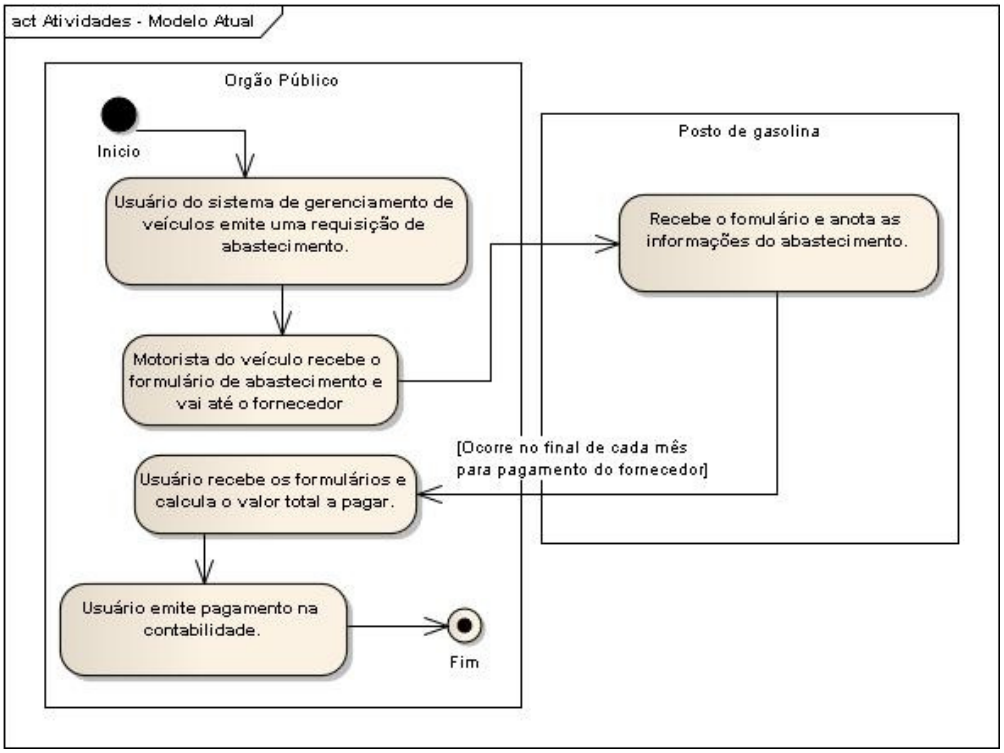


Figura 7 - Fluxo atual do processo

2.7 TRABALHOS CORRELATOS

São analisados quatro trabalhos correlatos, a saber, o sistema de gerenciamento de veículos do estado do Ceará, o sistema de controle de abastecimento desenvolvido pelo banco Bradesco, um sistema de transferência de arquivos para dispositivos móveis baseados em *web service* e um sistema de *web service* para inventário de estações em rede.

2.7.1 Sistema de abastecimento de veículos do Estado do Ceará

Santos (2011) desenvolveu um trabalho para implantação de um sistema de abastecimento de veículos no Estado do Ceará. O autor demonstra a experiência na

implantação de melhorias no processo de abastecimento de combustível dos veículos e máquinas que ocorreu no Estado. O projeto foi desenvolvido pela Secretaria do Planejamento e Gestão (SEPLAG), que diagnosticou alguns problemas antes da implantação do projeto.

> Os abastecimentos eram efetivados por meio de postos internos e postos externos; utilizavam diferentes mecanismos de coleta de dados dos abastecimentos; as informações eram disponibilizadas em dois sistemas gerenciais distintos que não possuíam comunicação para integração dos dados. (SANTOS, 2011).

Os modelos foram criados por uma equipe de consultores contratada pelo governo. Após a criação, os modelos foram submetidos à avaliação dos secretários da Casa Civil e da SEPLAG, que decidiram por aquele em que os abastecimentos seriam realizados em postos externos tendo os dados coletados por meio de cartão magnético, o qual disponibiliza as informações para um sistema de gerenciamento.

Definido o modelo, o próximo passo foi a elaboração de licitação, contratação do fornecedor, definição de categorias de veículos e confecção dos cartões magnéticos.

A operação de abastecimento por meio do modelo proposto teve seu início no dia 1 de agosto de 2009. No início houve resistência, pois em determinados abastecimentos houveram bloqueios quando realizados fora dos parâmetros que foram definidos.

Um objetivo alcançado foi a realização dos abastecimentos em um único sistema, possibilitando a criação de relatórios gerenciais. A mudança do comportamento do servidor público também foi necessária para um maior comprometimento com o processo. Hoje o esforço está concentrado na segurança do processo e na elaboração de relatórios de gestão. A Figura 8 demonstra o cartão magnético desenvolvido.

Fonte: Governo do Estado do Ceará (2011).

Figura 8 - Cartão magnético utilizado para os abastecimentos

2.7.2 Sistema de controle de abastecimento desenvolvido pelo banco Bradesco

Outro trabalho foi desenvolvido pelo Bradesco em parceria com a CTF Technologies do Brasil Ltda e as duas maiores distribuidoras de combustíveis nacionais, BR-Petrobras e Ipiranga. O projeto faz uma integração entre os postos CTF em todo o território nacional e o Bradesco, que faz a intermediação financeira das operações.

No momento do abastecimento nos postos credenciados CTF, antenas instaladas na boca do tanque de combustível do veículo e no bico da bomba de abastecimento se conectam e transferem os dados armazenados na Unidade de Veículo (UVE), instalada no veículo, para a unidade computadorizada do posto, o que permite obter eletronicamente, sem intervenção humana, informações como a identificação do veículo, registro de quilometragem do odômetro, quantidade, tipo e valor do combustível colocado, local do abastecimento, data, hora e média de consumo (BRADESCO, 2011). A Figura 9 demonstra a página inicial do sistema de abastecimento.

Fonte: CTF (2011).

Figura 9 - Portal de abastecimento de veículos

2.7.3 Sistema de transferência de arquivos para dispositivos móveis baseados em *web service*

O objetivo do projeto foi o desenvolvimento de um sistema que se utiliza, como parte da estrutura de comunicação, de um *web service*, que fará o controle de envio e recebimento de arquivos, tornando-se um repositório que disponibiliza o conteúdo para todo o ambiente corporativo. O software de *web service* pode também estender-se para um ambiente residencial, permitindo um acesso remoto através do *web service* (MICHELS, 2010).

Para que seja possível o acesso ao *web service* foi desenvolvido um cliente para dispositivos móveis, que deve possuir acesso à internet. Desta forma, a organização pode disponibilizar todos os arquivos necessários para seus representantes ou para funcionários externos que podem visualizar e enviar arquivos. A segurança da comunicação é garantida pela validação de acesso dos usuários no *web service* (MICHELS, 2010). A Figura 10 demonstra as telas de recebimento de arquivos.

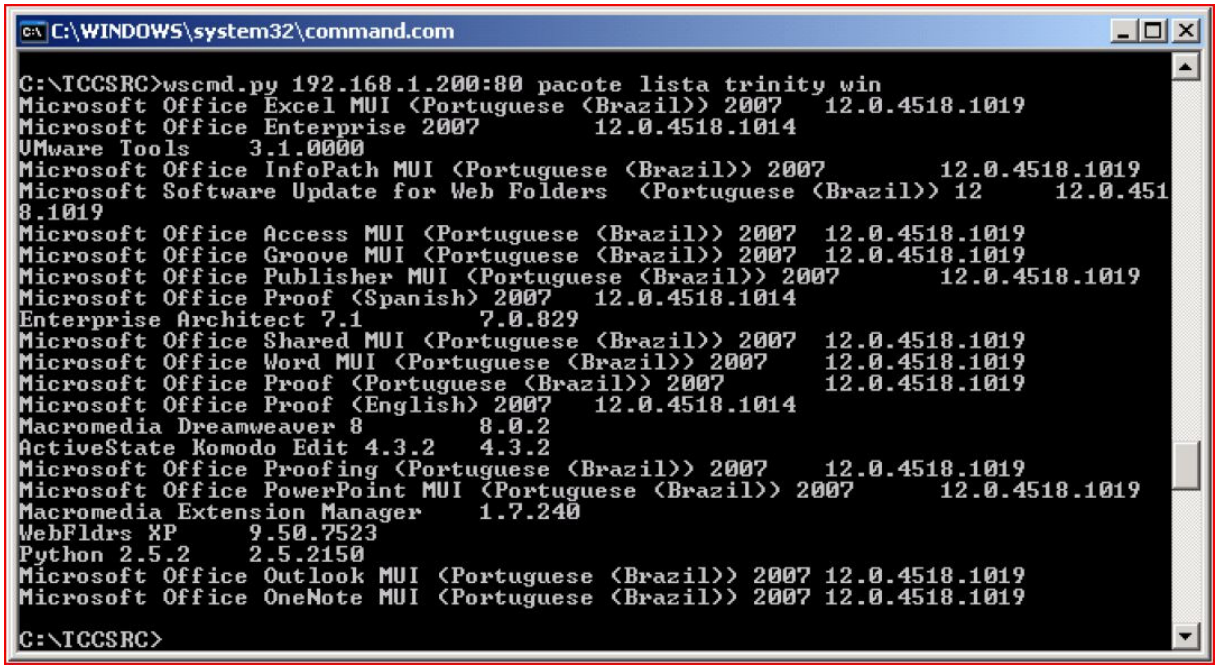
Fonte: Michels (2010).

Figura 10 - Tela de recebimento de arquivos

2.7.4 Sistema de *web service* para inventário de estações em rede

O projeto realizado desenvolveu um software baseado em *web service* para o armazenamento de informações relacionadas a equipamentos de informática, que são essenciais para a administração de uma rede de computadores. O *web service* é disponibilizado em um servidor, cuja responsabilidade é a persistência dos dados, apuração de relatórios gerenciais e a verificação de condições de alertas que podem ser definidas pelo administrador da rede (BAMBINETI, 2008).

O cliente deste *web service* é responsável pelo envio das informações que serão utilizadas para a gerência da máquina. O aplicativo é executado logo na inicialização do sistema operacional, analisa o hardware e os softwares instalados na máquina e envia estas informações ao *web service* (BAMBINETI, 2008). A Figura 11 demonstra a interface de consulta de informações da estação trinity.



Fonte: Bambineti (2008).

Figura 11 - Interface de consulta de informações da estação trinity

3 DESENVOLVIMENTO

Neste capítulo serão abordados os tópicos sobre o levantamento das informações, as especificações, a implementação e a operacionalidade da aplicação. Apresenta também os diagramas elaborados utilizando a *Unified Modeling Language* (UML), tais como o diagrama de atividades, o diagrama de caso de uso e o diagrama de casses.

3.1 SOLUÇÃO PROPOSTA

Este trabalho utiliza técnicas de comunicação de dados através da *internet*, aproveitando a infra-estrutura já existente no fornecedor e no cliente, evitando gastos com aquisição de qualquer outra ferramenta e equipamentos.

O processo de emissão de requisições de abastecimento foi totalmente remodelado. Algumas estruturas foram criadas para tornar o processo de abastecimento confiável e flexibilizar as consultas de valores para o órgão público e o fornecedor. Este processo foi dividido em duas etapas, sendo a primeira o cadastro do abastecimento e a segunda parte a transmissão e recebimento do abastecimento.

A seguir a Figura 12 demonstra o fluxo do processo para o cadastro do abastecimento.

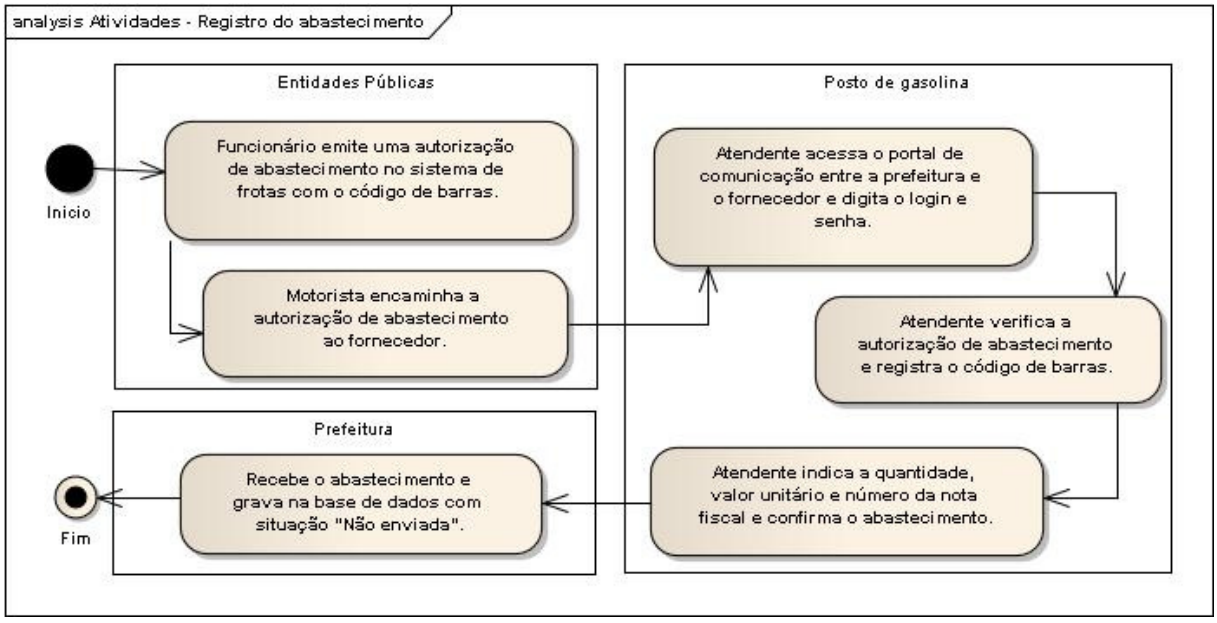


Figura 12 – Fluxo do cadastro de abastecimento proposto

A Figura 13 demonstra o fluxo do processo de transmissão e recebimento do abastecimento.

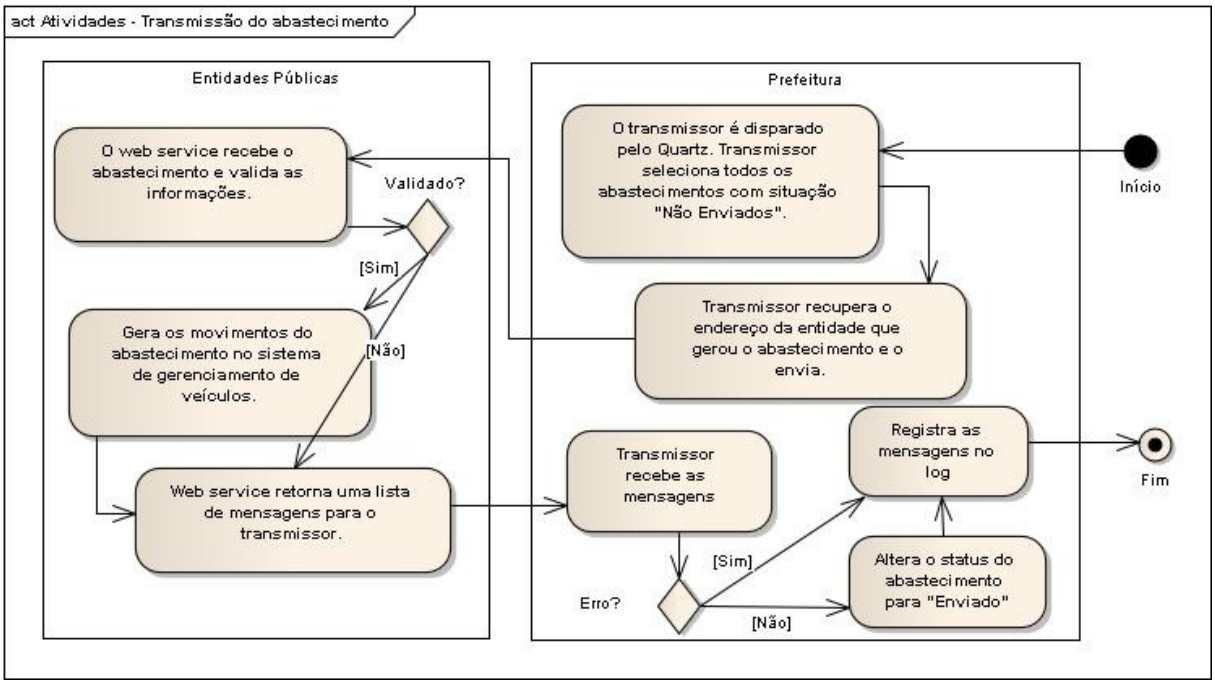


Figura 13 - Fluxo da transmissão e recebimento proposto

O processo inicia no cadastro de uma autorização de abastecimento que é realizado por um funcionário do órgão público no sistema Pública-SAV. Esta autorização é um documento que indica a liberação do condutor para abastecer o veículo em um determinado fornecedor com o material e a quantidade máxima permitida.

Para utilizar estas autorizações, o condutor imprime a autorização de abastecimento e encaminha até o fornecedor descrito no documento. O atendente do posto de gasolina só realizará o abastecimento se o condutor estiver com a autorização impressa contendo o código de barras, a numeração do código e as informações do veículo.

Os abastecimentos são cadastrados através de um portal de relacionamento entre cliente e fornecedor, chamado NET-Fornecedores, que a prefeitura deverá possuir. O fornecedor de combustível deve estar devidamente cadastrado na prefeitura para acessar este portal. A Figura 14 demonstra uma visão geral da estrutura de comunicação que foi desenvolvida entre os envolvidos.

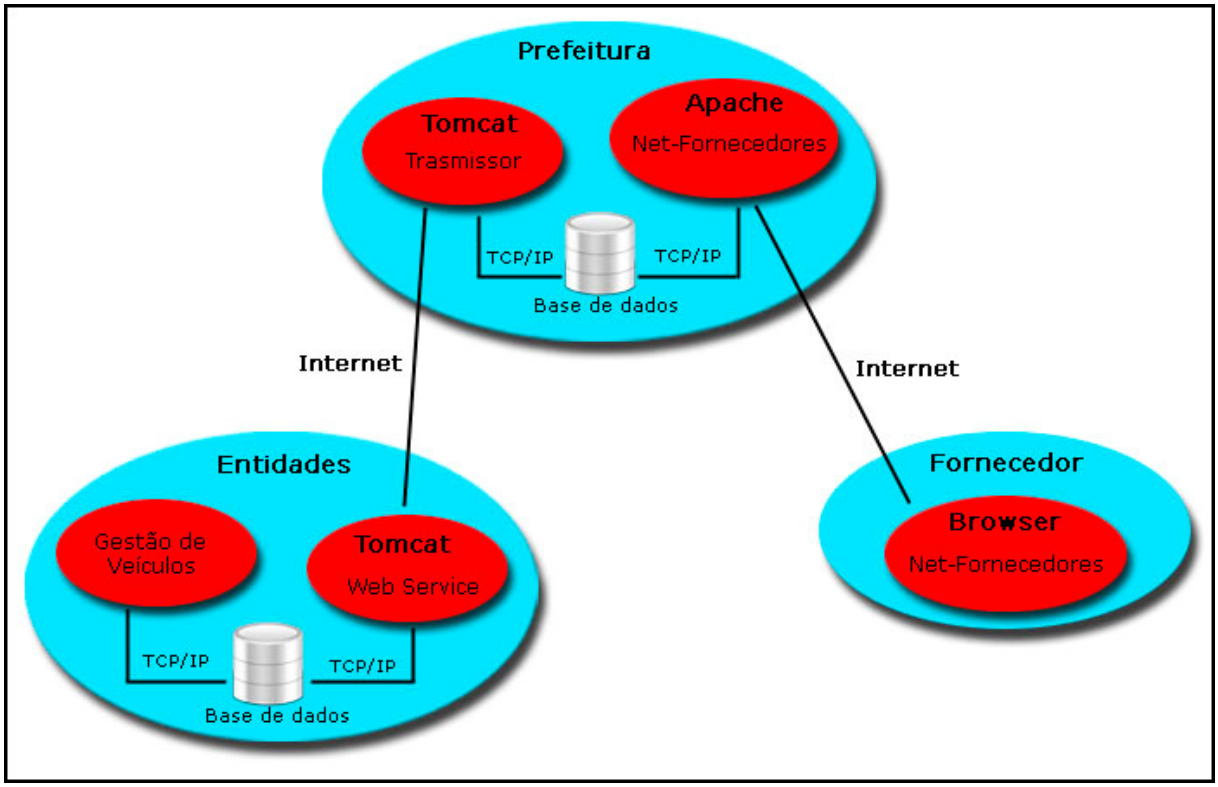


Figura 14 – Estrutura de comunicação entre os envolvidos

Um dos principais objetivos deste processo é a simplicidade de utilização do portal. O funcionário do posto necessita apenas de um endereço para efetuar o registro de abastecimento de todos os fundos, da câmara e da prefeitura da cidade, facilitando o acesso ao sistema de abastecimento. Para registrar o abastecimento com mais rapidez, o funcionário do posto pode utilizar o leitor de código de barras que seja compatível com o QR Code ou digitar a sequência que está impressa no relatório de autorização de abastecimento. Este código de barras transporta todas as informações necessárias para o atendente, inclusive de qual entidade originou o abastecimento, tornando possível a utilização de qualquer entidade pública da cidade desde que esteja devidamente cadastrada na prefeitura.

A prefeitura torna-se uma central de abastecimento que armazenará todos os abastecimentos realizados pelos órgãos públicos da cidade. Estes abastecimentos cadastrados na prefeitura serão transmitidos aos órgãos de origem para que possa ser cadastrado as utilizações do abastecimento, bem como a nota fiscal. Esta transmissão é realizada pelo módulo chamado transmissor.

O módulo transmissor utiliza-se de um *framework* de agendamento de tarefas chamado Quartz. O módulo está instalado em um servidor Tomcat juntamente com o Quartz

que está configurado para ser executado em um determinado período. Quando a condição do agendamento é satisfeita, o Quartz instancia o módulo transmissor e inicia a transferência.

Na base de dados da prefeitura existe um cadastro com todas as entidades públicas da cidade, juntamente com o endereço do *web service*. Este endereço é utilizado para enviar o abastecimento da prefeitura até a entidade que o originou, através do código do cliente que está impresso na autorização. Cada abastecimento possui o código da entidade de origem, desta maneira o transmissor envia um abastecimento por vez, aguardando a resposta do *web service* com as mensagens.

As mensagens que retornam do *web service* são cadastradas na base de dados da prefeitura. Estas mensagens podem ser erros que ocorreram durante a gravação ou o sucesso da transmissão. Se a transmissão ocorrer sem nenhum erro, o transmissor atualiza o abastecimento marcando-o como "Enviado com sucesso", caso contrário, apenas grava as mensagens. Posteriormente um funcionário da prefeitura pode acompanhar estes registros de *log* para avaliar os problemas que podem estar ocorrendo nas transmissões.

O recebimento do abastecimento é realizado por um módulo chamado *web service*, que está instalado em um servidor Tomcat no órgão que originou a autorização de abastecimento. O *web service* conecta-se diretamente na base de dados da entidade que possui o sistema de gerenciamento de frotas e gera toda a movimentação de utilização do abastecimento.

O *web service* é responsável por receber e fazer as verificações de integridade das informações dos abastecimentos, se o abastecimento estiver apto para ser gravado, o *web service* gera os movimentos de abastecimento juntamente com a nota fiscal no sistema de gerenciamento de frotas. No processo de verificação e gravação, o *web service* preenche uma lista de mensagens com todos os erros que ocorreram durante o processo. Caso esta lista esteja preenchida, o *web service* não grava a movimentação de utilização do abastecimento e retorna ao transmissor todos os erros que ocorreram durante o processo. Se não gerou erro, o *web service* grava os movimentos de utilização e envia ao transmissor uma mensagem indicando sucesso na transmissão. Este resultado é repassado ao transmissor na forma de uma lista de mensagens em XML.

O funcionário do posto de gasolina pode consultar todos os abastecimentos realizados por ele utilizando a opção de listagem de abastecimento no *software* NET-Fornecedores. Nesta listagem ele poderá verificar se o abastecimento foi realizado com sucesso e se já foi transmitido para a entidade de origem, indicando que o abastecimento está comprovado.

A entidade que originou a autorização de abastecimento pode visualizar todas as utilizações de abastecimento e as notas fiscais utilizando as consultas já existentes no sistema Pública-SAV.

As mensagens que ocorreram durante a transmissão dos abastecimentos podem ser visualizadas em um relatório que foi desenvolvido no Pública-SAV. Este relatório poderá ser visualizado somente na prefeitura, a qual possui todas as mensagens das transmissões realizadas.

Com esta estrutura foi possível automatizar o processo de abastecimento de combustível em toda a cidade com o mínimo de impacto possível aos clientes da empresa Pública Informática.

3.2 ESPECIFICAÇÃO

Nesta seção são apresentados os requisitos funcionais e não funcionais, os casos de uso bem como o diagrama de classe. A especificação foi elaborada utilizando-se a UML. Para a construção dos diagramas foi utilizada a ferramenta Enterprise Architect (EA).

3.2.1 Requisitos funcionais

O Quadro 1 apresenta os requisitos funcionais previstos para o sistema e sua rastreabilidade, ou seja, vinculação com o(s) caso(s) de uso associado(s).

Requisitos Funcionais	**Caso de Uso**
RF01: O sistema de gerenciamento de veículos deverá permitir a emissão da autorização de abastecimento com código de barras.	UC01
RF02: O sistema de gerenciamento de veículos deverá permitir a emissão de relatório de erros que ocorreram no envio.	UC02
RF03: O módulo de abastecimento deverá permitir ao atendente do posto de gasolina cadastrar o abastecimento utilizando código de barras ou através da sequência de números.	UC03
RF04: O módulo de abastecimento deverá permitir ao atendente do posto de	UC04

gasolina consultar todos os abastecimentos realizados por ele.	
RF05: O módulo de *web service* deverá receber os abastecimento.	UC05
RF06: O módulo de *web service* deverá validar os abastecimentos.	UC06
RF07: O módulo de *web service* deverá gravar um movimento de abastecimento e a nota fiscal no sistema de gerenciamento de veículos.	UC07
RF08: O módulo de *web service* deverá retornar uma lista de mensagens com erros ou o sucesso da transmissão para o transmissor.	UC08
RF09: O módulo transmissor deverá permitir a transmissão de todos os abastecimentos para os *web services* cadastrados.	UC09
RF10: O módulo transmissor deverá atualizar o status dos abastecimentos cadastrados.	UC10
RF11: O módulo transmissor deverá cadastrar as mensagens que o *web service* retornará.	UC11
RF12: O módulo transmissor deverá agendar as transmissões dos abastecimentos.	UC12

Quadro 1 - Requisitos funcionais

3.2.2 Requisitos não funcionais

O Quadro 2 lista os requisitos não funcionais previstos para o sistema.

Requisitos Não Funcionais
RNF01: Somente o fornecedor cadastrado na prefeitura terá acesso ao sistema (segurança).
RNF02: O módulo de cadastro de abastecimento será desenvolvida em PHP versão 5 (implementação).
RNF03: O *web service* deverá ser desenvolvido em Java (implementação).
RNF04: O transmissor deverá ser desenvolvido em Java (implementação).
RNF05: A interface deverá utilizar banco de dados Firebird (implementação).
RNF06: O *web service* deverá utilizar banco de dados Firebird (implementação).
RNF07: O transmissor deverá utilizar banco de dados Firebird (implementação).
RNF08: O módulo de cadastro de abastecimento deve ser acessível via *browser* Firefox versão 6 ou superior (portabilidade).

Quadro 2 - Requisitos não funcionais

3.2.3 Diagrama de caso de uso

Esta subseção apresenta na Figura 15 o diagrama de casos de uso do sistema. Para o melhor entendimento do projeto, o detalhamento dos principais casos de uso encontra-se no Apêndice A.

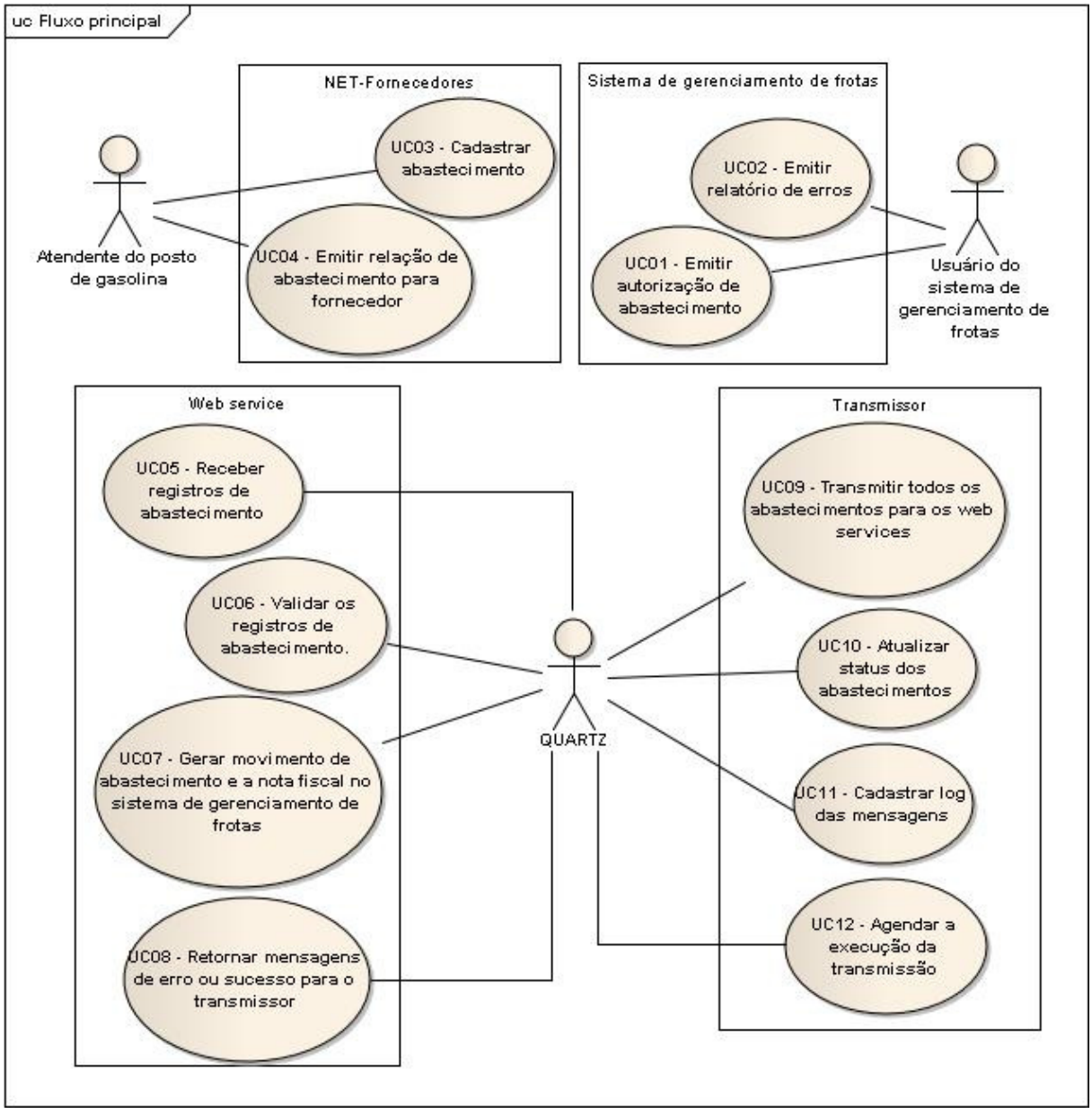


Figura 15 - Diagrama de caso de uso do sistema

O ator Quartz é a framework de agendamento que executa as rotinas do transmissor e as rotinas do *web service*. O ator usuário do sistema de gerenciamento de frotas é um

funcionário da entidade pública responsável por manter as frotas. O ator atendente do posto de gasolina é o funcionário responsável por registrar os abastecimentos no posto de gasolina.

O trabalho é dividido em três módulos, cada módulo possui objetivos específicos com regras e soluções distintas, não são instalados em um mesmo local e utilizam linguagem de programação e *frameworks* diferentes.

A implementação realizada no sistema Pública-SAV não é considerada um módulo, pois foi desenvolvido apenas a impressão do código de barras no relatório de autorização e um relatório para visualização dos erros das transmissões.

3.2.4 Diagrama de classe

O diagrama de classe é divido em dois módulos, o primeiro diagrama se refere ao módulo transmissor e o segundo se refere ao módulo *web service*. Na Figura 16 é apresentado o diagrama de classe do módulo transmissor.

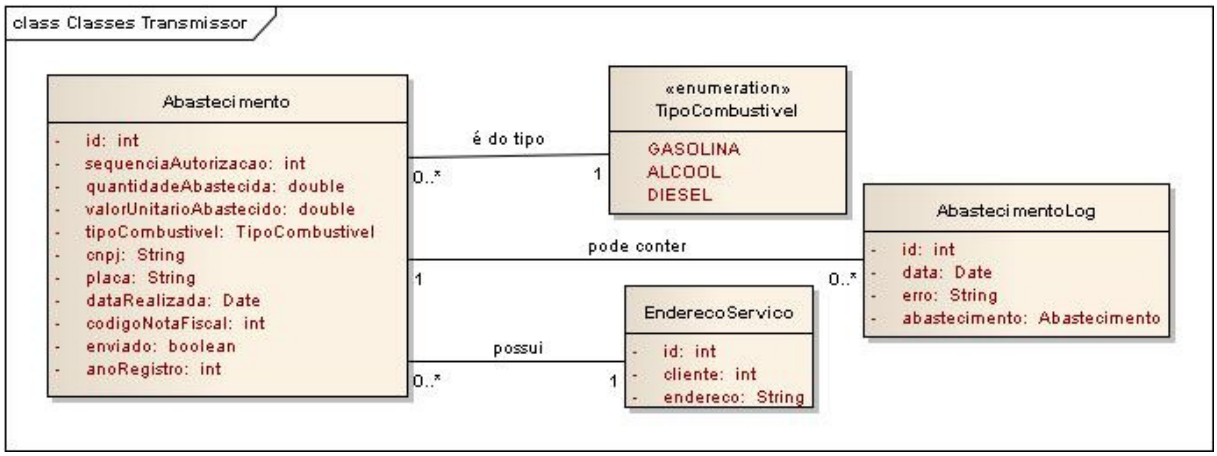


Figura 16 - Diagrama de classe do módulo transmissor

Este diagrama de classe é utilizado pelo transmissor para efetuar a leitura dos abastecimentos e registrar as mensagens que ocorreram durante a transmissão. Para pesquisar os endereços do *web service*, o transmissor utiliza a classe EnderecoServico, que possui o cadastro de todos os clientes que utilizam este processo.

O módulo de cadastro de abastecimento que foi desenvolvido no Net-Fornecedores não utiliza estas classes, pois o projeto não é orientado a objetos.

A Figura 17 apresenta o diagrama de classe do módulo *web service*.

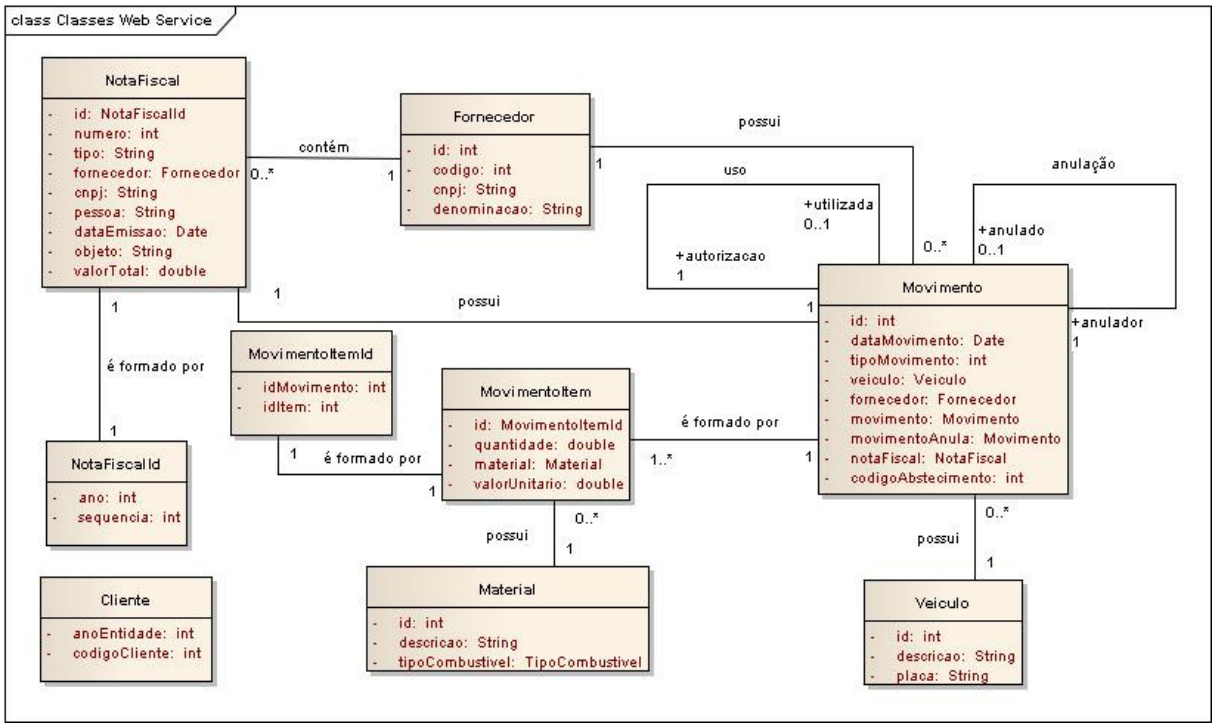


Figura 17 – Diagrama de classe do módulo *web service*

Este diagrama de classe foi mapeado através da estrutura de movimentação existente no projeto Pública-SAV. Esta estrutura permite ao *web service* gerar todas as informações necessárias para utilizar o abastecimento. A entidade Cliente é utilizada para validar informações do abastecimento, por este motivo ela não está relacionada a nenhuma entidade.

O dicionário de dados das classes está descrito a partir do Apêndice B.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

Para o desenvolvimento do *web service*, foi utilizado a linguagem Java e a *framework* Axis2 para auxiliar na publicação do *web service*. Esse *framework* faz o mapeamento da

classe Java que será publicada e gera o arquivo WSDL. A Figura 18 mostra um arquivo WSDL gerado pelo Axis2.

Figura 18 - Arquivo WSDL gerado pelo Axis2

No *web service* apenas o método CadastraAbastecimento foi publicado. Esse método é responsável por invocar a validação e geração dos movimentos de abastecimento. O método retorna uma lista que é preenchida no andamento do processo. Esta lista contém apenas os erros que ocorreram nas validações do abastecimento. A Figura 19 demonstra a implementação realizada neste método.

```
/**
 * Instancia a classe {@link AbastecimentoEntity} e carrega todo o conteúdo para efetuar as validações.
 * @return XML com uma lista de erros
 */
public String CadastraAbastecimento(String xml) {
    Mensagens.getInstance().listaErros.clear();
    AbastecimentoEntity abastecimento = null;
    System.out.println("XML recebido: "+ xml);
    XStream xStream = new XStream();
    try{
        abastecimento = (AbastecimentoEntity) xStream.fromXML(xml);
    } catch(Throwable ex){
        System.out.println("Erro ao montar o abastecimento...");
        Mensagens.getInstance().listaErros.add(AbastecimentoResources.getMensagem("abastecimento.erro.xml.i
        return xStream.toXML(Mensagens.getInstance()) ;
    }
    //Gera as validações e grava os movimentos caso sucesso ou retorna os erros.
    AbastecimentoStorageImpl abastecimentoStorage = new AbastecimentoStorageImpl(abastecimento);
    try {
        abastecimentoStorage.Execute();
    } catch (Exception ex) {
        System.out.println("Erro ao cadastrar abastecimento: " + ex.getMessage());
    }
    return xStream.toXML(Mensagens.getInstance());
}
```

Figura 19 - Implementação do método publicado no *web service* responsável por cadastrar o abastecimento

A execução da validação e a geração dos movimentos é garantida pelo método Execute() da classe AbastecimentoStorageImpl, cuja função é transformar os atributos da classe Abastecimento em entidades, fazer a validação das informações e gerar os movimentos de abastecimento caso tudo estiver correto. A Figura 20 demonstra a implementação deste método.

```
/**
 * Instancia as classes que serão utilizadas pelo web service e grava os movimentos.
 */
public void Execute(){
    //Recupera as entidades. Será feita todas as validações no validate.
    cliente = getClienteObj();
    material = getMaterialObj();
    movimentoAutorizacao = getMovimentoAutorizacaoObj();
    movimentoItemAutorizacao = getMovimentoItemAutorizacaoObj();
    veiculo = getVeiculoObj();
    fornecedor = getFornecedorObj();
    //chama todas as validações necessárias para gravar o abastecimento
    AbastecimentoKnowledge validate = new AbastecimentoKnowledge(sessao);
    validate.execute(this);
    //Somente gera o movimento se não ocorreu nenhum tipo de erro.
    if (Mensagens.getInstance().listaErros.isEmpty()){
        GeraNotaFiscal();
        GeraMovimento();
        GeraMovimentoItem();
    }
    // Nos métodos que geram documentos podem ocorrer erros, por isso deve validar a lista aqui também
    if (Mensagens.getInstance().listaErros.isEmpty()){
        Mensagens.getInstance().listaErros.add(AbastecimentoResources.getMensagem("sucesso"));
        transacao.commit();
    } else {
        transacao.rollback();
    }
    sessao.close();
}
```

Figura 20 - Implementação da rotina principal do *web service*

A implementação do transmissor foi realizada através da linguagem de programação Java com o auxilio do *framework* Quartz para o agendamento das transmissões. O Quartz é iniciado junto com o servidor Tomcat, sendo assim o agendamento é realizado somente se o Tomcat estiver funcionando. Para realizar o agendamento foi utilizada a *trigger CronTrigger*, inicialmente configurada para ser executada a cada um minuto. A função do agendamento é engatilhar uma tarefa para que ela seja executada quando se cumprir o tempo programado. A Figura 21 demonstra como é criado o agendamento de tarefas.

```
protected void init() throws Exception {
    try {
        // Fabrica para criar instância do agendador
        StdSchedulerFactory schedFactory = new StdSchedulerFactory();
        getLog().info("1 -> Instancia um agendador");
        Agendador.agenda = schedFactory.getScheduler();

        getLog().info("2 -> Instancia a tarefa de transmissão dos abastecimento (job)");
        JobDetail job = new JobDetail("PublicaTransmissor", Scheduler.DEFAULT_GROUP, Tarefa.class);

        //http://www.patternizando.com.br/2011/08/tutorial-crontrigger/
        getLog().info("3 -> Instancia uma trigger com as configurações da agenda ");
        CronTrigger triggerAgenda = new CronTrigger("PublicaAgendadorTrigger", Scheduler.DEFAULT_GROUP,
                "PublicaTransmissor", Scheduler.DEFAULT_GROUP, "0/30 * * * * ?");

        java.util.Date ft = agenda.scheduleJob(job, triggerAgenda);
        getLog().info(job.getFullName() + " Foi programado para funcionar em: " + ft +
                " e repete baseado na expressão: " + triggerAgenda.getCronExpression());

        getLog().info("4 -> Agendador iniciado...");
        Agendador.agenda.start();

    } catch (Exception e) {
        System.out.println("\n\nErro ao iniciar o Pública-Transmissor.\n\n" + e.getMessage());
    }
}
```

Figura 21 - Implementação do agendamento de tarefas do Quartz

O transmissor conecta-se no *web service* utilizando o endereço da entidade que originou o abastecimento. Este endereço é cadastrado na base de dados da prefeitura e é consultado a cada abastecimento.

A programação da conexão com o *web service* pode ser realizada de duas maneiras. A primeira maneira é gerar um cliente do *web service* a partir do endereço do WSDL utilizando um *plugin* do NetBeans. Desta maneira o módulo transmissor ficou limitado, pois foi possível consumir apenas um *web service*. A segunda maneira é implementar uma classe que faz a conexão com o *web service* a partir de um endereço dinâmico. A grande vantagem desta segunda maneira é a utilização de vários *web services*, vantagem esta que é essencial para o funcionamento adequado do módulo transmissor. A implementação da rotina está disponível no site da Apache Axis2. Para consumir o *web service* é necessário configurar o *namespace*, o nome do método que será acessado e o endereço de conexão do *web service*.

Quando a tarefa é executada, o transmissor seleciona todos os registros de abastecimentos que ainda não foram enviados. O registro de abastecimento é serializado para XML contendo todas as informações necessárias. A cada registro de abastecimento uma nova conexão com um *web service* é estabelecida e essa conexão deve aguardar a resposta do *web service* para enviar o próximo abastecimento. A resposta do *web service* é um objeto serializado em XML contendo uma lista de mensagens que são registradas em uma tabela de *log*. A Figura 22 mostra a implementação do envio dos abastecimentos.

```
public void execute(){
    //Carrega a lista de abastecimentos não enviados
    setLista();
    //Percorre a lista e envia cada abastecimento
    for (AbastecimentoEntity abastecimento : list){
        transacao.begin();
        List<String> listaErros = null;
        //Cria a classe que envia o abastecimento conforme o código do cliente
        UseWS useWS = new UseWS(getUrlWebService(abastecimento.getCodigoCliente()));
        if (useWS != null){
            //Envia o abastecimento para o web service e aguarda o retorno
            String xml = useWS.cadastraAbastecimento(abastecimento);
            System.out.println("Retorno do web service: " + xml);
            if (xml != null){
                listaErros = getListaErros(xml).listaErros;
                if (listaErros == null){
                    listaErros.add(AbastecimentoResources.getMensagem("servico.retorno.xml.vazio"));
                }
            } else {
                listaErros.add(AbastecimentoResources.getMensagem("servico.retorno.vazio"));
            }
        } else {
            listaErros.add(AbastecimentoResources.getMensagem("servico.conexao.falha"));
        }

        //Salva a lista de erro em uma tabela de log
        if (listaErros != null){
            salvaLog(listaErros, abastecimento);
        }
```

Figura 22 - Implementação do envio do abastecimento para o *web service*

A passagem de informação entre o *web service* e o transmissor é realizada através de uma biblioteca chamada XStream. Esta biblioteca é responsável por serializar e desserializar os objetos, transformando-os em uma estrutura XML. Esta estrutura é transportada utilizando caracteres em texto plano, agilizando a implementação dos parâmetros de comunicação entre o *web service* e o transmissor.

A leitura e persistência dos dados foi totalmente realizada pelo *framework* Hibernate. Todas as informações foram mapeadas para a estrutura objeto-relacional inclusive as tabelas de movimento que já existiam no sistema Pública-SAV. Os relacionamentos das tabelas de movimento deste sistema possuem chaves compostas, sendo necessária a implementação de uma classe para unir todos os atributos que formam a chave utilizando a anotação @Embeddable. Esta classe é referenciada na entidade que possui a chave composta utilizando a anotação @EmbeddedId. Essa implementação é necessária porque o Hibernate aceita apenas um atributo com a anotação @Id em cada entidade e também, por outro lado, por ser totalmente inviável a alteração da estrutura da base de dados. A Figura 23 e 24 demonstra este tipo de relacionamento que acontece na entidade NotaFiscal.

```java
/**
 * Representa a chave composta de uma {@link  NotaFiscalEntity}
 * @author Gabriel Vieira
 */
@Embeddable
public class NotaFiscalId extends BasicEntity {

    @Column(name="dreano")
    private int ano;
    @Column(name="dresq")
    private int sequencia;
```

Figura 23 - Entidade que agrupa todos os atributos que formam a chave da nota fiscal

```java
/**
 * Representa uma nota fiscal
 * @author Gabriel Vieira
 */
@MappedSuperclass
public abstract class NotaFiscal extends BasicEntity {

    @EmbeddedId
    private NotaFiscalId id;

    @Column(name = "drenumero", nullable = false)
    private int numero;

    @Column(name = "dretipo", nullable = false)
    private String tipo = "No";
```

Figura 24 - Entidade da nota fiscal com a referência do id

As consultas de dados do sistema foram realizadas através da API Criteria do Hibernate, tornando-se o padrão para pesquisa de dados na base. Nenhuma consulta foi elaborada utilizando SQL ou o HQL. A Figura 25 demonstra a implementação da consulta do fornecedor pelo Cadastro Nacional de Pessoa Jurídica (CNPJ).

```
/**
 * Retorna o fornecedor pelo cnpj.
 * @return fornecedor ou nulo
 */
private FornecedorEntity getFornecedorObj(){
    Criteria busca = sessao.createCriteria(FornecedorEntity.class);
    busca.add(Restrictions.eq("cnpj", abastecimento.getCnpj()));
    List<FornecedorEntity> fornecedores = busca.list();
    if (fornecedores.size() == 1){
        return fornecedores.get(0);
    } else {
        return null;
    }
}
```

Figura 25 - Implementação de consulta usando a API Criteria

A página de cadastro do abastecimento foi desenvolvida utilizando a linguagem de programação PHP instalada em um servidor Apache. A estrutura do sistema Net-Fornecedores já existia. Assim, a implementação realizada foi apenas da página de cadastro do abastecimento e da consulta dos abastecimentos realizados. A Figura 26 mostra como é gravado o abastecimento na plataforma PHP.

```
$SQL = "SELECT gen_id(SQ_ABASTECIMENTO,1) FROM rdb" . "$" . "database";
//Executa o SQL para recuperar o próximo número
$rsCodigo = $conexao->SelectLimit($SQL);
//Recupera o próximo número.
$ID_ABASTECIMENTO = $rsCodigo->Fields("GEN_ID");
$SQL = "INSERT INTO ABASTECIMENTO (ID_ABASTECIMENTO, SQ_AUTORIZACAO, CD_CLIENTE,
                                   QT_ABASTECIDA, VL_UNITARIO, TP_COMBUSTIVEL,
                                   NR_CNPJ, NR_PLACA, DT_REALIZADA, CD_NOTAFISCAL,
                                   FL_ENVIADO, NR_ANOREGISTRO)
VALUES (" . $ID_ABASTECIMENTO . "
        , " . $autorizacao . "
        , " . $codigoCliente . "
        , " . $quantidade . "
        , " . numToDB($valorUnitario) . "
        , " . $combustivel . "
        , " . $cnpj . "
        , " . $placa . "
        , '" . converte_data_us($dataAtual)."'" . "
        , " . $notafiscal . "
        , " . "'" . "N" . "'" . "
        , " . "2010);";
echo $SQL;
$conexao->Execute($SQL);

zeraVariaveis();
```

Figura 26 - Implementação da inserção do abastecimento

A tela de cadastro de abastecimento conta com uma formatação dos campos de valor, quantidade e CNPJ utilizado a linguagem JavaScript, conforme mostra a Figura 27.

```
$("#valorUnitario")
.priceFormat({
    prefix: 'R$ '
})
.click(function(){ $(this).select()});

$("#quantidade")
.priceFormat({
    decimalPlaces: 3,
    centsSeparator: '.'
})
.click(function(){ $(this).select()});

$("#quantidadeMaxima")
.priceFormat({
    decimalPlaces: 3,
    centsSeparator: '.'
})
.click(function(){ $(this).select()});

$("#cnpj").unmask();
// Limpando o campo
$("#cnpj").mask("99.999.999/9999-99",{ completed : function() {
        $("#cnpj").focus();
    }});
```

Figura 27 - Formatação do campo valor, quantidade e CNPJ em JavaScript

A geração do código de barras e impressão da autorização do abastecimento foi implementada no sistema de gestão de veículos chamado Pública-SAV. Para auxiliar na implementação do código de barras foi utilizado um componente chamado TAsBarcode que é gratuito para testar, funcionando apenas na máquina que possui o Delphi instalado. O componente gera a imagem automaticamente, necessitando apenas das informações da autorização de abastecimento juntamente com os dígitos verificadores.

Para gerar os dígitos verificadores foi necessário construir um algoritmo baseado na quebra das informações da autorização em cadeias de números em três partes. Para gerar o primeiro dígito é aplicada uma divisão dos números da primeira parte por sete, sendo o resto da divisão o dígito verificador. A primeira parte dos dígitos é somada com a segunda parte somando também o primeiro dígito verificador, sendo que o resultado desta soma é novamente dividido por sete e o resto da divisão é o segundo dígito verificador. Para gerar o

terceiro dígito é necessário somar a primeira parte com a segunda parte e com a terceira parte, juntamente com os dois dígitos que foram gerados. A soma é dividida por sete e o resto da divisão é o terceiro e último dígito verificador. A Figura 28 mostra a implementação da geração dos dígitos verificadores e da geração do código de barras QR Code.

```
begin
  inherited;
  codigoInt:= Qr_Movtovei.FieldByName('MVESQ').AsFloat +
          StrToFloat(TiraFormatacaoCPFCGC(Qr_Movtovei.FieldByName('FORCPFCGCFORNECEDOR').AsString));
  temp:= Round(codigoInt) mod 7;
  codigo:= TiraFormatacaoCPFCGC(Qr_Movtovei.FieldByName('FORCPFCGCFORNECEDOR').AsString) +
           FormatFloat('00000000', Qr_Movtovei.FieldByName('mvesq').AsInteger) + IntToStr(temp);
  codigoInt:= codigoInt + temp +
              FormComum.CodigoCliente +
              StrToInt(ReplaceStr(FormatFloat('000.000', Qr_ItensVeiIveQt.AsFloat), ',',')) +
              Qr_ItensVei.FieldByName('tp_combustivel').AsInteger;
  temp:= Round(codigoInt) mod 7;
  codigo:= codigo + FormatFloat('0000',FormComum.CodigoCliente) +
              ReplaceStr(FormatFloat('000.000', Qr_ItensVeiIveQt.AsFloat), ',','') +
              Qr_ItensVei.FieldByName('tp_combustivel').AsString + IntToStr(temp);
  codigoInt:= codigoInt + temp +
               StrToInt(getPlaca) + StrToInt(TiraFormatacaoCPFCGC(Qr_Movtovei.fieldbyname('mvedt').AsString));
  temp:= Round(codigoInt) mod 7;
  codigo:= codigo + getPlaca + TiraFormatacaoCPFCGC(Qr_Movtovei.fieldbyname('mvedt').AsString) + IntToStr(temp);
  Barcode2D_QRCode1.Barcode := codigo;
  Barcode2D_QRCode1.Draw;
end;
```

Figura 28 - Implementação dos dígitos verificadores e geração do QR Code

3.3.2 Operacionalidade da implementação

O processo inicia quando o usuário do sistema de gestão de veículos Pública-SAV cadastra uma autorização de abastecimento para um veículo indicando a data, o fornecedor, o material utilizado e o condutor que será o responsável pelo abastecimento. A Figura 29 mostra a tela de cadastro da autorização.

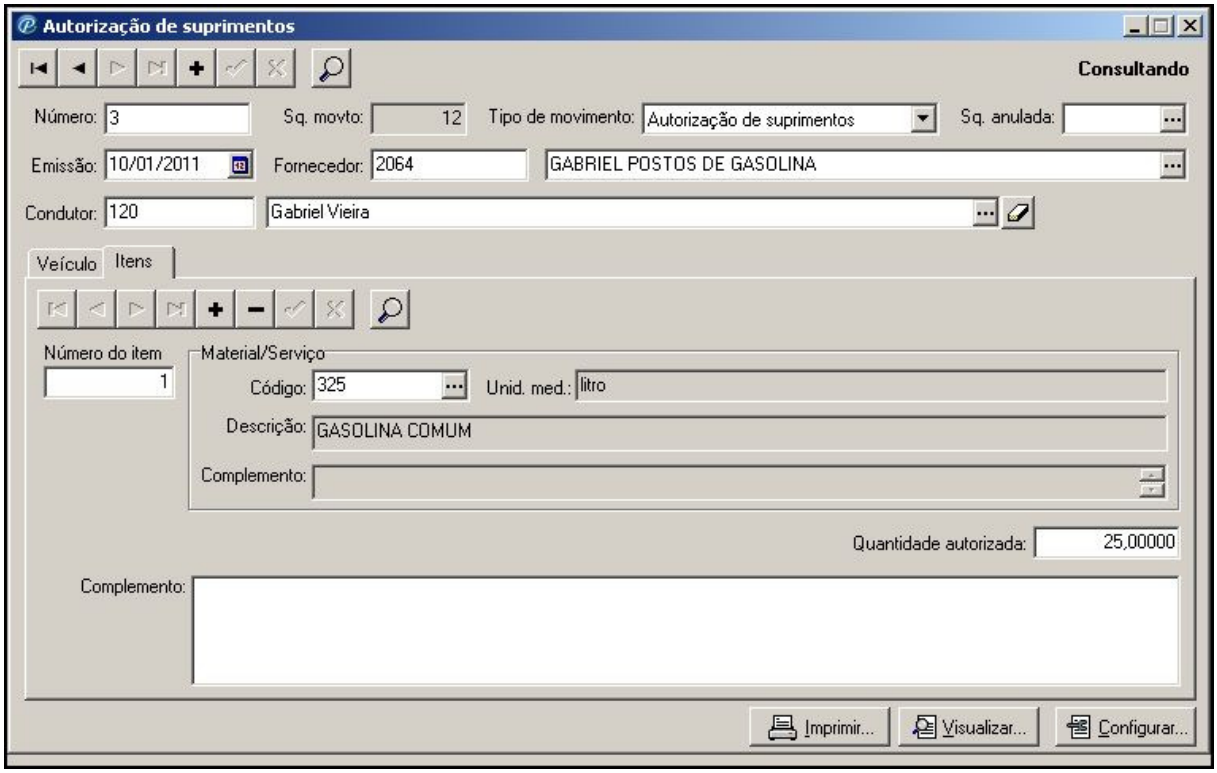


Figura 29 - Tela de cadastro da autorização de suprimento do sistema Pública-SAV

Para completar o cadastro é necessário imprimir a autorização juntamente com o código de barras. O relatório é impresso com a imagem do código de barras e também com a sequência numérica, que pode ser digitada na tela de cadastro do abastecimento. A Figura 30 mostra o relatório impresso pronto para ser entregue ao atendente do posto de gasolina.

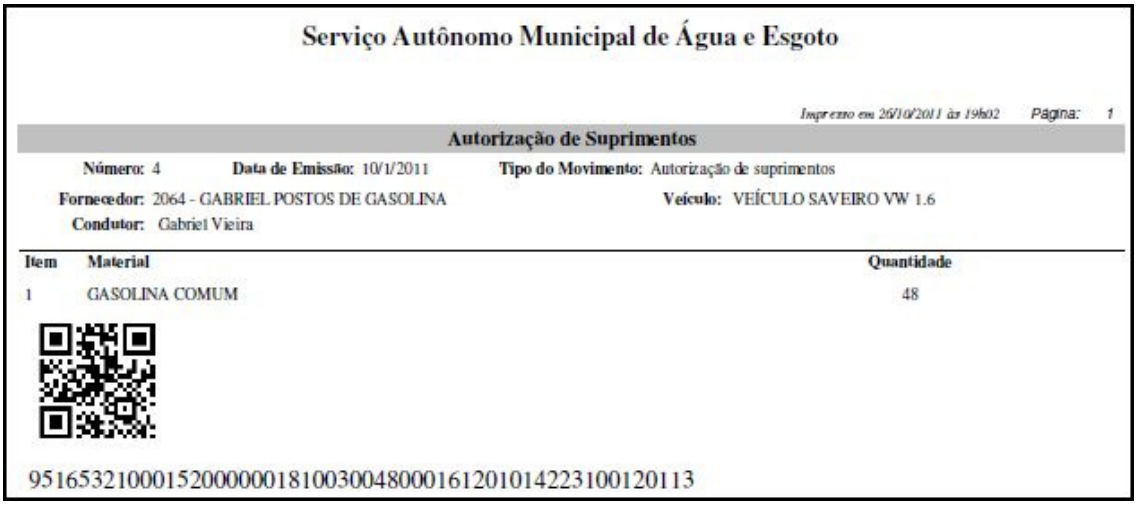


Figura 30 - Relatório de autorização de suprimento

O usuário do sistema de gestão de veículos entrega a autorização impressa para o

condutor do veículo efetuar o abastecimento. O condutor leva a autorização até o fornecedor e abastece o veículo com a quantidade máxima impressa no relatório. O atendente poderá acessar o portal da prefeitura no momento do abastecimento ou quando ele possuir tempo livre utilizando as credenciais de acesso devidamente cadastradas na prefeitura. A Figura 31 mostra a tela de cadastro do abastecimento.

Figura 31 - Cadastro de abastecimento

Ao passar a leitora do código de barras na autorização de abastecimento, os campos do código de barras são preenchidos. Caso a leitora não consiga ler o código de barras o atendente deverá digitar a sequência de caracteres. O botão confirmar carrega as informações do abastecimento nos campos superiores, permitindo o atendente visualizar as informações do veículo e digitar o valor unitário, a quantidade abastecida e o número da nota fiscal. A Figura 32 demonstra a tela de abastecimento pronta para ser gravada.

Figura 32 - Cadastro de abastecimento pronto para ser gravado

O cadastro do abastecimento é concluído quando o atendente do posto de gasolina confirma a gravação e o sistema executa com sucesso todas as validações dos dados que foram digitados. Agora o abastecimento está sob responsabilidade da prefeitura, que o enviará para o órgão de origem.

Paralelo ao cadastro de abastecimento funciona o transmissor instalado em um servidor Tomcat na prefeitura preparado para transmitir todos os registros de abastecimentos para as entidades que o originaram conforme o tempo agendado. A Figura 33 mostra uma parte do *log* de envio do abastecimento do servidor Tomcat.

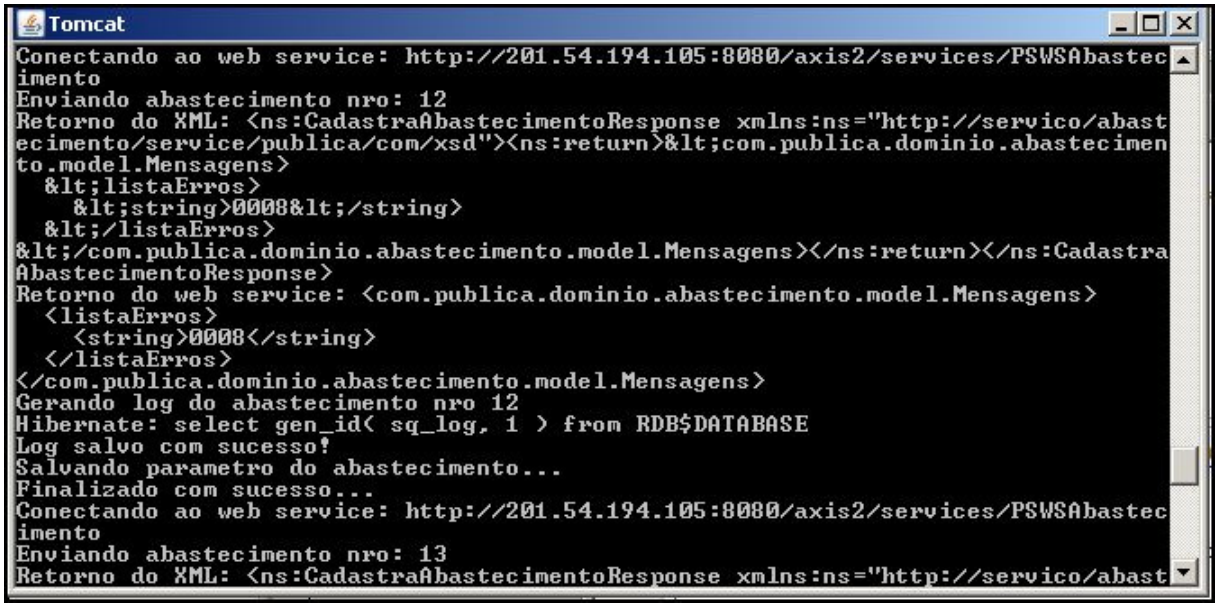


Figura 33 - *Log* do transmissor enviando um abastecimento

O recebimento destes abastecimentos é feito pelo *web service* que funciona paralelo ao transmissor instalado em um servidor Tomcat em uma entidade pública. A função do *web service* é registrar todos os movimentos necessários para concluir a utilização da autorização de abastecimento no sistema Pública-SAV. A Figura 34 mostra o *log* do servidor Tomcat ao efetuar um registro de abastecimento.

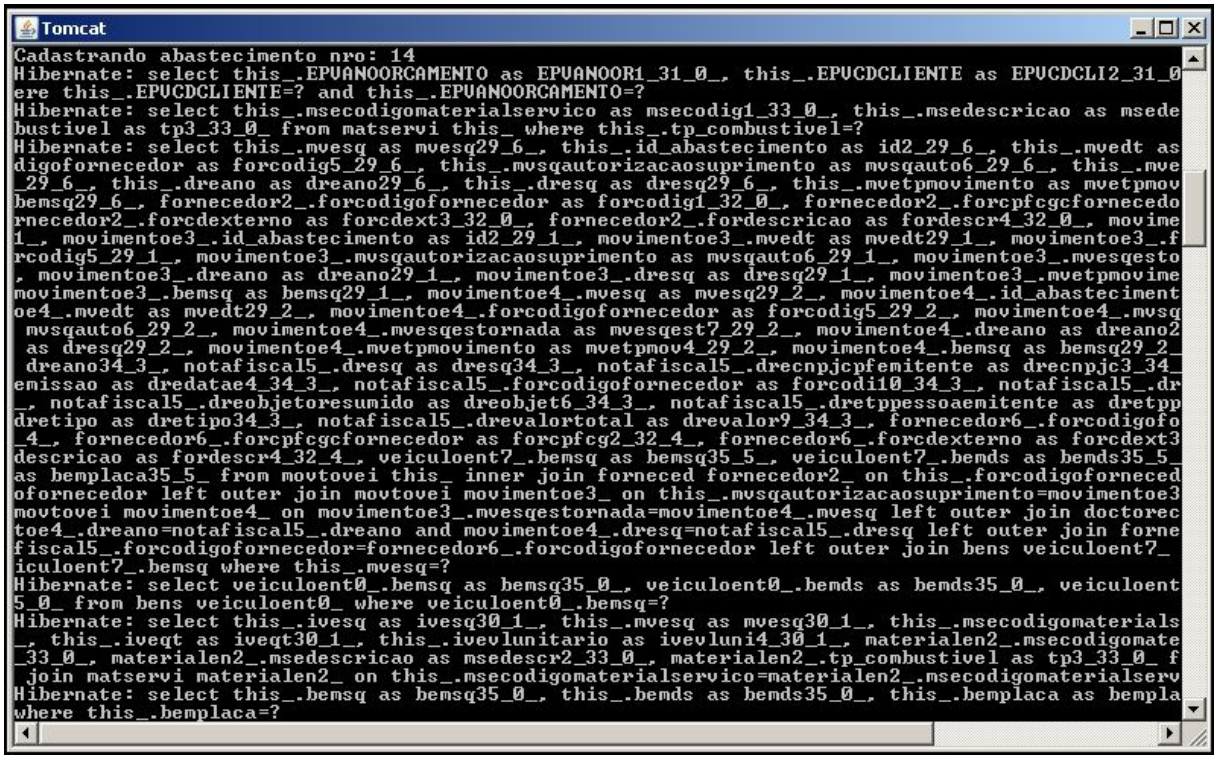


Figura 34 - *Log* do *web service* recebendo um pedido de gravação de abastecimento

Os movimentos de utilização do abastecimento podem ser confirmados consultando na tela de lançamento manual de utilização de abastecimento na entidade que originou a autorização. A Figura 35 demonstra os dados da tela de utilização que foi gerada pelo *web service*.

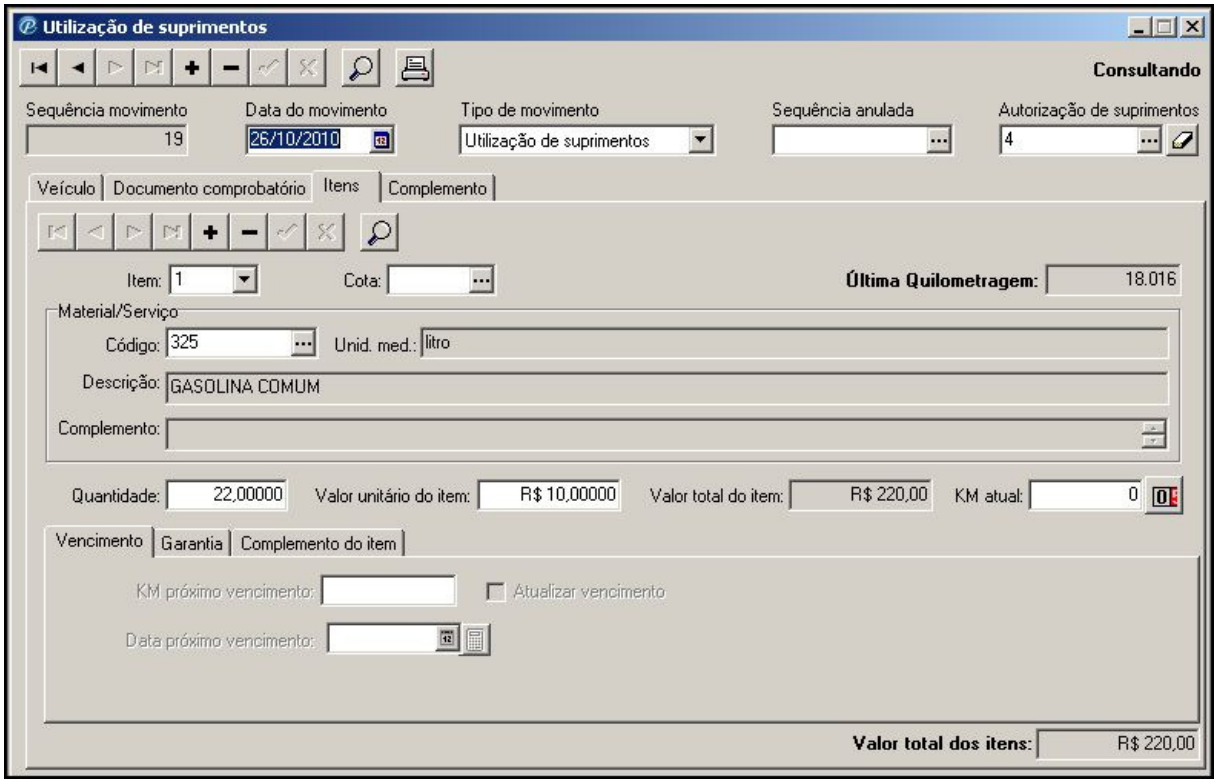


Figura 35 - Utilização do abastecimento gerada pelo *web service*

O usuário da prefeitura responsável por gerenciar os abastecimentos poderá acompanhar o *log* de erros utilizando um relatório conforme demonstra a Figura 36.

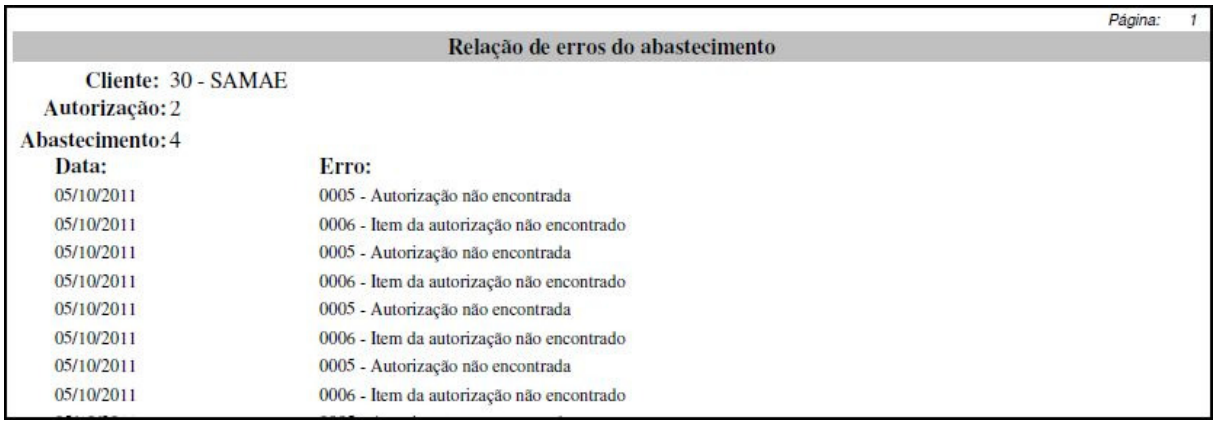


Figura 36 - Consulta de erros na base de dados da prefeitura

O funcionário do posto de gasolina poderá consultar todos os abastecimentos que foram gravados na prefeitura cujo fornecimento foi realizado por ele, podendo visualizar se o abastecimento foi registrado na entidade que originou a autorização. A Figura 37 mostra a consulta de abastecimentos realizados pelo fornecedor.

Figura 37 - Consulta de abastecimentos realizados pelo fornecedor

3.4 RESULTADOS E DISCUSSÃO

A construção da arquitetura de comunicação entre o fornecedor e as entidades públicas foi concluída alcançando todos os objetivos, utilizando apenas um endereço de acesso ao sistema de abastecimento e a comunicação automática com as entidades envolvidas.

A meta do projeto é gerar o menor impacto possível, utilizando-se da arquitetura de comunicação atual, cuja base de dados não é unificada para todo o município, ou seja, cada entidade mantém seus dados conforme necessita sem adotar um padrão nas informações, exigindo um grande trabalho de análise nas integrações entre os envolvidos.

O TCC de Bambineti (2008) e Michels (2010) são de segmentos diferentes do projeto desenvolvido, mas utilizam a mesma arquitetura de comunicação, sendo que o maior diferencial é a geração do *log* de erros na comunicação entre o *web service* e o transmissor.

Quanto aos trabalhos correlatos, a proposta elaborada pelo Bradesco (2011) possui um público alvo de grande escala, podendo estender-se à área privada por conter grande infraestrutura tecnológica. O mesmo ocorre com a implantação de cartões magnéticos, citado pelo trabalho correlato de Santos (2011), que trata de abastecimentos de combustível para veículos de um estado inteiro, extensível às demais regiões do país. Já o presente trabalho é direcionado para pequenos municípios, que possuem pouca infraestrutura de comunicação e que necessitam automatizar os abastecimentos utilizando um baixo orçamento para a realização, utilizando assim tecnologia de comunicação simples, tornando o projeto mais barato, quando comparado com os trabalhos correlatos de Santos (2011) e Bradesco (2011).

Como maiores dificuldades, destacam-se a configuração do *framework* Axis2 para a

criação do *web servic*e e a configuração do mapeamento das classes do *framework* Hibernate quando foi necessário mapear entidades que utilizam chave composta. A dificuldade foi entender o funcionamento de um aplicativo web e como são armazenadas as configurações dos serviços. Já no Hibernate, a dificuldade foi encontrar uma solução para entidades de chave composta, pois o Hibernate trata estes casos com algumas particularidades.

Um questionamento informal, na forma de entrevista, foi realizado com o gestor de negócios da empresa para avaliar o sistema e a implantação em um cliente piloto. O gestor comentou que o software está bastante completo, atendendo todas as expectativas da empresa por não exigir mudanças radicais, como por exemplo, a compra de equipamentos.

Com relação à possível implantação em um cliente, o gestor comentou que o software deve ser mais simples, pois a infra-estrutura dos clientes que foram planejados para receber o projeto piloto é muito inferior ao que se tem hoje em outros locais, tornando-se um sistema para ser implantado em uma grande cidade, fugindo do foco da empresa neste momento.

Por fim, o gestor decidiu alterar o projeto, removendo a estrutura de comunicação que utiliza *web service* e transmissor, utilizando somente a tela de cadastro de abastecimento. O gestor não descartou totalmente a ideia de implantar a estrutura de transmissão e recebimento conforme foi apresentado neste trabalho. Contudo, segundo ele, a estrutura construída para transmitir e receber deve passar por outras análises a fim de garantir a segurança das informações, inclusive a de aceitação do cliente por reunir informações de todas as entidades na base de dados da prefeitura. No momento, a instalação do Net-Fornecedores já é de conhecimento dos consultores, o que facilita a implantação. Desta maneira, cada cliente deverá instalar o sistema Net-Fornecedores para ter esta funcionalidade.

4 CONCLUSÕES

Neste trabalho foi proposto o desenvolvimento de um processo de abastecimento de veículos utilizando a arquitetura de comunicação entre as entidades envolvidas sem maiores alterações da infra-estrutura atual. O objetivo final do processo é a garantia de maior segurança e agilidade nos abastecimentos e pagamentos de fornecedores nas entidades públicas.

Para que fosse possível a integração entre os clientes da empresa em um município, foi necessária a criação de três módulos. Cada módulo possui uma função diferente dentro do processo e todos são essenciais para a conclusão do abastecimento, sendo imprescindível a utilização da internet em todos os envolvidos. Toda essa estrutura foi montada para garantir o menor impacto possível no processo anterior, passando parcialmente todo o processo para a forma eletrônica. O único documento que não foi possível converter na forma eletrônica foi o relatório de abastecimento, pois não foi encontrada uma solução para a segurança da informação ao transmitir os dados do abastecimento para o fornecedor. Desta maneira, pode-se concluir que todos os objetivos específicos do trabalho foram alcançados.

Foram utilizados diversos *frameworks* e APIs para o desenvolvimento dos módulos, sendo que todos tiveram uma grande contribuição para a agilidade no desenvolvimento. O único módulo que não foi utilizado nenhum tipo de *framework* e API foi a página de cadastro em PHP, pois ela deve seguir o padrão de desenvolvimento da empresa.

Este trabalho demonstrou como é possível trazer uma realidade do cliente para um conjunto de módulos sem grandes alterações na infra-estrutura, fazendo uma integração aos softwares da empresa. Também demonstrou que o abastecimento pode ter um controle maior para o gestor público e para o fornecedor com consultas e relatórios dos dados em tempo real.

Com o desenvolvimento do trabalho foi possível concluir algumas vantagens:

a) baixo custo na implantação do processo;
b) não é necessária a aquisição de nenhum equipamento adicional;
c) maior controle nos abastecimentos realizados;
d) *log* de erros na comunicação entre os envolvidos.

Conclui-se com a realização deste trabalho o aumento dos conhecimentos sobre as aplicações desenvolvidas para o ambiente *web*, o que compõem o seu desenvolvimento e as integrações que são possíveis de serem realizadas. A integração do *framework* Hibernate em um projeto Java utilizando anotações nos atributos das classes, mapeando também as chaves

compostas das tabelas de movimento também foram responsável por grande aumento de conhecimento nesta área. A realização de agendamento utilizando o *framework* Quartz contribuiu para o aprendizado de um assunto não conhecido até o desenvolvimento do trabalho.

4.1 EXTENSÕES

Para dar continuidade ao sistema, pode-se desenvolver a comunicação entre o fornecedor e a prefeitura utilizando certificado digital, retirando do sistema a necessidade de utilizar *login* e senha para abastecer um veículo. O objetivo principal desta implementação é a prefeitura criptografar o abastecimento, desta forma apenas o fornecedor que poderá abastecer conseguirá visualizar os dados e efetuar o abastecimento.

Outra extensão que poderia ser desenvolvida é a autorização e utilização de peças e outras manutenções que possam ocorrer no veículo, utilizando a mesma forma de comunicação, com alterações no item da autorização que poderá ser um material ou um serviço e o suporte a vários materiais ou serviços em uma única autorização.

Um item que não foi desenvolvido, por não fazer parte do escopo do trabalho, e que poderá fazer parte de uma extensão é a implementação de segurança no envio e recebimento de abastecimentos entre o *web service* e o transmissor, utilizando criptografia para gerar um canal de comunicação seguro entre os envolvidos.

REFERÊNCIAS BIBLIOGRÁFICAS

APACHE SOFTWARE FOUNDATION. **Apache Axis2 user's guide**. Los Angeles, 2009. Disponível em: <http://axis.apache.org/axis2/java/core/docs/userguide.html#intro> Acesso em: 07 out. 2011.

BAMBINETI, Charles. **Sistema de Web Service para inventário de estações em rede.** 2008. 50 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

BAUER, Christian; KING, Gavian. **Java Persistence com Hibernate**. Rio de Janeiro: Editora Ciência Moderna Ltda., 2007.

BRADESCO. **CTF - Controle Telefrotas.** [S.l.], [2011]. Disponível em <http://www.bradescopoderpublico.com.br/Conteudo/Municipal/Legislativo/ControleTelefrotas.aspx?menuid=1180>. Acesso em: 25 mar. 2011.

COELHO, Igo. Conhecendo o Quartz. **Java Magazine**, Grajaú, n.60, p. 8-10, 2008.

CTF. **CTF – Abastecimento Inteligente**. São Paulo, [2011]. Disponível em: <http://www.portalctf.com.br/institucional/>. Acesso em: 12 out. 2011.

CUNHA, Davi. **Web Services, SOAP e Aplicações Web.** [S.l.], 2002. Disponível em <http://devedge-temp.mozilla.org/viewsource/2002/soap-overview/index_pt_br.html>. Acesso em: 26 mar. 2011.

DESTRO, Daniel. **Tutorial do Apache Axis 1.3**. [S.l.], 2006. Disponível em: <http://www.guj.com.br/articles/180>. Acesso em: 07 out. 2011.

FERNANDES, Raphaela G.; LIMA, Gleydson de A. F. **Hibernate com Anotações**. Natal, 2007. Disponível em: <http://wiki.futurepages.org/lib/exe/fetch.php?media=quickstart:hibernate_anotacoes.pdf>. Acesso em: 11 out. 2011.

GOVERNO DO ESTADO DO CEARÁ. **Manual de Abastecimento de Frota**. [S.l.], [2011]. Disponível em: <http://www.gestaodoservidor.ce.gov.br/site/images/stories/manuais/bt20.pdf>. Acesso em: 14 out. 2011.

LEMKE, Camilla. **Quartz Scheduler Atualizado Para Versão 2.0**. [S.l.], 2011. Disponível em: <http://under-linux.org/quartz-scheduler-atualizado-para-versao-2-0-2577/>. Acesso em: 07 out. 2011.

LEMOS, Marcus V. de S. **Introdução a Persistência de Dados com Hibernate e Annotation**. Teresina, [2011?]. Disponível em: <http://www.youblisher.com/files/publications/25/147600/pdf.pdf>. Acesso em: 11 out. 2011.

MICHELS, F. **Sistema de transferência de arquivos para dispositivos móveis baseados em web services.** 2010. 54 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

PRASS, Ronaldo. **Entenda o que são os 'QR Codes', Códigos Lidos Pelos Celulares**. [S.l.], 2011. Disponível em: <http://g1.globo.com/tecnologia/noticia/2011/05/entenda-o-que-sao-os-qr-codes-codigos-lidos-pelos-celulares.html>. Acesso em: 11 out. 2011.

PRIMOS, Raphael. **Cliente do BB pode pagar boletos por leitura de QR Code**. Rio de Janeiro, 2011. Disponível em: <http://www.conexaomercado.com.br/wp/index.php/2011/09/cliente-do-bb-pode-pagar-boletos-via-celular-por-leitura-de-qr-code/>. Acesso em: 12 out. 2011.

SAMPAIO, Cleuton. **SOA e Web services em Java.** Rio de Janeiro: Brasport, 2006.

SANTOS, Ricardo Ribeiro. **Projeto melhoria no abastecimento de combustível da frota do estado do Ceará.** Curitiba, [2011]. Disponível em <http://www.escoladegoverno.pr.gov.br/arquivos/File/Material_%20CONSAD/paineis_III_congresso_consad/painel_8/projeto_melhoria_no_abastecimento_de_combustivel_da_frota_do_estado_do_ceara.pdf >. Acessado em: 06 dez. 2011.

TRIBUNAL DE CONTAS DE SANTA CATARINA. **O papel do TCE.** Florianópolis, [2011]. Disponível em: <http://www.tce.sc.gov.br/web/instituicao/papel>. Acesso em: 27 mar. 2011.

WILBERT, Christian. **O Que é QR Code?**. Caxias do Sul, 2010. Disponível em: < http://www.oficinadanet.com.br/artigo/celulares_e_telefonia/o_que_e_qr_code>. Acesso em: 11 out. 2011.

APÊNDICE A – Detalhamento dos casos de uso

O Quadro 3 apresenta o detalhamento do caso de uso "Emitir autorização de abastecimento".

Caso de uso – Emitir autorização de abastecimento
Ator: Usuário do sistema de gerenciamento de frotas
Objetivo: Emitir autorização para abastecimento de veículo
Pré-condições: Usuário deve fazer login(autenticação) no sistema.
Pós-condições: Usuário imprime a requisição de abastecimento com código de barras.

Cenário Principal:
1. Usuário acessa a tela de emissão de autorização de abastecimento
2. Sistema verifica se o usuário tem permissão
3. Usuário escolhe o fornecedor
4. Usuário escolhe o veículo
5. Usuário escolhe o material
6. Usuário escolhe a data de emissão
7. Usuário grava o cadastro da autorização de abastecimento
8. Sistema grava o cadastro e permite ao usuário imprimir a autorização
9. Usuário imprime a autorização
10. Sistema gera as informações do código de barras e imprime.

Cenário Alternativo:
No passo 2, caso o usuário não tenha permissão:
2.1 Sistema não habilita o menu para emissão de autorização
2.2 Caso de uso é encerrado

Cenário Alternativo:
No passo 8, caso ocorra algum erro na gravação:
8.1 Sistema apresenta mensagem ao usuário
8.2 Volta para o caso de uso

Quadro 3 - Caso de uso "Emitir autorização de abastecimento"

O Quadro 4 apresenta o detalhamento do caso de uso "Emitir relatório de erros".

Caso de uso – Emitir relatório de erros
Ator: Usuário do sistema de gerenciamento de frotas
Objetivo: Emitir relatório dos erros que ocorreram na transmissão dos abastecimentos
Pré-condições: Usuário deve fazer login(autenticação) no sistema.
Pós-condições: Usuário imprime o relatório com as informações dos erros.

Cenário Principal:
1. Usuário acessa a tela de emissão de relatórios
2. Usuário escolhe o relatório dos erros do abastecimento
3. Sistema carrega o relatório com todos os erros que ocorreram.

Quadro 4 - Caso de uso "Emitir relatório de erros"

O Quadro 5 apresenta o detalhamento do caso de uso “Cadastrar abastecimento”.

Caso de uso – Cadastrar abastecimento
Ator: Atendente do posto de gasolina
Objetivo: Cadastrar o abastecimento por meio da autorização impressa
Pré-condições: Atendente logar no sistema Net-Fornecedores.
Pós-condições: Abastecimento cadastrado na prefeitura.

Cenário Principal:
1. Atendente passa o leitor de código de barras na autorização de abastecimento
2. Atendente confirma a sequência do código de barras
3. Sistema valida a da ta de validade da autorização
4. Sistema valida o fornecedor da autorização com o fornecedor logado
5. Atendente informa o valor unitário
6. Atendente informa a quantidade abastecida
7. Atendente informa o número da nota fiscal
8. Atendente grava o abastecimento
9. Sistema grava o abastecimento

Cenário Alternativo:
No passo 1, caso a leitora não leia o código:
1.1 Atendente deve digitar o código impresso na autorização
1.2 Volta ao cenário principal

Cenário Alternativo:
No passo 8, caso a quantidade é maior do que a máxima permitida:
8.1 Sistema apresenta mensagem ao usuário indicando quantidade inválida
8.2 Atendente informa a quantidade menor do que a máxima permitida
8.3 Volta ao cenário principal

Quadro 5 - Caso de uso “Cadastrar abastecimento”

O Quadro 6 apresenta o detalhamento do caso de uso “Transmitir todos os abastecimentos para os *web services*”.

Caso de uso – Transmitir todos os abastecimentos para os *web services*
Ator: Transmissor
Objetivo: Enviar os abastecimentos cadastrados na prefeitura para os órgãos responsáveis pelo abastecimento
Pré-condições: O agendamento ter sido efetuado em um servidor Tomcat.
Pós-condições: Transmitir todos os abastecimentos cadastrados.

Cenário Principal:
1. O agendamento é disparado pelo Quartz em um período determinado
2. Transmissor seleciona todos os abastecimentos cadastrados que não foram enviados
3. Transmissor recupera o endereço dos *web services* cadastrados
4. Transmissor envia o abastecimento para o *web service* que o originou
5. Transmissor aguarda a resposta do *web service*
6. Transmissor grava o *log* das mensagens que o *web service* retornou
7. Transmissor atualiza o status dos abastecimentos para “Enviado”
8. Transmissor finaliza as conexões e grava as alterações na base de dados

Cenário Alternativo:
No passo 1, caso o agendamento não foi realizado:
1.1 Transmissor não é executado, nenhum abastecimento é enviado
1.2 O caso de uso é encerrado

Cenário Alternativo:
No passo 3, caso o abastecimento seja de um órgão que não tem endereço de *web service*:
3.1 Transmissor ignora o abastecimento, deixando-o com status "Não Enviado"
3.2 Volta ao cenário principal

Quadro 6 - Caso de uso "Transmitir todos os abastecimentos para os *web services*"

O Quadro 7 apresenta o detalhamento do caso de uso "Receber autorização de abastecimento".

Caso de uso – Receber autorização de abastecimento
Ator: *Web service*
Objetivo: Receber os abastecimentos que foram enviados pelo transmissor
Pré-condições: *Web service* deve estar instalado em um servidor Tomcat.
Pós-condições: Recebe a autorização de abastecimento.

Cenário Principal:
1. *Web service* recebe um abastecimento no formato XML
2. *Web service* executa as regras e gera os movimentos
3. *Web service* retorna as mensagens que foram geradas no formato XML

Cenário Alternativo:
No passo 1, caso o XML seja inválido:
1.1 *Web service* retorna o erro indicando falha no arquivo XML
1.2 O caso de uso é encerrado

Quadro 7 - Caso de uso "Receber autorização de abastecimento"

O Quadro 8 apresenta o detalhamento do caso de uso "Validar a autorização de abastecimento".

Caso de uso – Validar a autorização de abastecimento
Ator: *Web service*
Objetivo: Analisar o abastecimento, verificando todas as informações
Pré-condições: *Web service* deve receber um abastecimento do transmissor.
Pós-condições: Permissão para gravar os movimentos.

Cenário Principal:
1. *Web service* verifica se todas as informações do abastecimento foram preenchidas
2. *Web service* verifica se a autorização existe
3. *Web service* verifica se a autorização não está anulada
4. *Web service* verifica se o fornecedor da autorização é o mesmo do abastecimento
5. *Web service* verifica se o veículo da autorização é o mesmo do abastecimento
6. *Web service* conclui as validações.

Cenário Alternativo:
Nos passos 6, caso ocorra alguma inconsistência nas validações:
6.1 *Web service* insere em uma lista de mensagem o problema ocorrido
6.2 Volta para o cenário principal

Quadro 8 - Caso de uso "Validar a autorização de abastecimento"

O Quadro 9 apresenta o detalhamento do caso de uso "Gerar movimento de abastecimento e nota fiscal no sistema de gerenciamento de frotas".

Caso de uso – Gerar movimento de abastecimento e nota fiscal no sistema de gerenciamento de frotas
Ator: *Web service*
Objetivo: Gerar todos os movimentos necessários para concluir o registro de um abastecimento no sistema de frotas, juntamente com a nota fiscal
Pré-condições: Abastecimento deve estar validado
Pós-condições: Movimentos de abastecimento e nota fiscal cadastrados.

Cenário Principal:
1. *Web service* gera o movimento de utilização de suprimento
2. *Web service* gera o movimento do item da utilização de suprimento
3. *Web service* gera a nota fiscal
4. *Web service* grava todas as alterações na base de dados

Cenário Alternativo:
No passo 4 caso ocorra algum erro na gravação:
4.1 *Web service* retorna o erro indicando falha ao gravar o documento
4.2 O caso de uso é encerrado

Quadro 9 - Caso de uso "Gerar movimento de abastecimento e nota fiscal no sistema de gerenciamento de frotas"

APÊNDICE B – Dicionário de dados

Este apêndice apresenta a descrição detalhada das entidades do diagrama de classes previstas na seção 3.2.4. O campo chave é nomeado como “id”, sempre existirá em toda a entidade. Os tipos de dados de cada campo são descritos a seguir:

a) *String*: armazena caracteres;
b) *int*: armazena números inteiros;
c) *Date*: armazena datas;
d) *double*: armazena valores com decimais;
e) TipoCombustivel: enumerador que contêm os tipos de combustível;
f) *boolean*: armazena verdadeiro ou falso;
g) Abastecimento: armazena a referência da entidade Abastecimento;
h) Veiculo: armazena a referência da entidade Veiculo;
i) Fornecedor: armazena a referência da entidade Fornecedor;
j) Movimento: armazena a referência da entidade Movimento;
k) NotaFiscal: armazena a referência da entidade NotaFiscal;
l) MovimentoItemId: armazena a referência da entidade MovimentoItemId;
m) Material: armazena a referência da entidade Material.

O Quadro 10 apresenta o dicionário de dados da classe “Abastecimento”.

Entidade: Abastecimento		
Atributo	**Tipo**	**Descrição**
Id	int	Sequência única do abastecimento
sequenciaAutorizacao	int	Sequência da autorização
codigoCliente	int	
quantidadeAbastecida	double	
valorUnitarioAbastecido	double	
tipoCombustivel	TipoCombustivel	
CNPJ	String	CNPJ completo do fornecedor
Placa	String	Placa do veículo
dataRealizada	Date	
codigoNotaFiscal	int	Número da nota fiscal
Enviado	boolean	Abastecimento eviado ao *web service*
anoRegistro	int	Ano de exercício que foi gerado

Quadro 10 - Dicionário de dados da classe “Abastecimento”

O Quadro 11 apresenta o dicionário de dados da classe “AbastecimentoLog”.

Entidade: AbastecimentoLog		
Atributo	**Tipo**	**Descrição**
Id	Int	Sequência do log
Data	Date	Data que ocorreu a tentativa
Erro	String	Código do erro
Abastecimento	Abastecimento	Relacionamento com o abastecimento

Quadro 11 – Dicionário de dados da classe “AbastecimentoLog”

O Quadro 12 apresenta o dicionário de dados da classe “Movimento”.

Entidade: Movimento		
Atributo	**Tipo**	**Descrição**
Id	int	Sequência do movimento
dataMovimento	Date	Data do movimento
tipoMovimento	int	Representa um tipo de movimento
Veiculo	Veiculo	Relacionamento com o veículo
Fornecedor	Fornecedor	Relacionamento com o fornecedor
Movimento	Movimento	Relacionamento com outro movimento
movimentoAnula	Movimento	Relacionamento com movimento de anulação
notaFiscal	NotaFiscal	Relacionamento com a nota fiscal
codigoAbstecimento	int	Código do abastecimento

Quadro 12 - Dicionário de dados da classe “Movimento”

O Quadro 13 apresenta o dicionário de dados da classe “MovimentoItem”.

Entidade: MovimentoItem		
Atributo	**Tipo**	**Descrição**
Id	MovimentoItemId	Sequência do item
Quantidade	double	Quantidade movimentada
Material	Material	Relacionamento com o material
valorUnitario	double	Valor unitário movimentado

Quadro 13 - Dicionário de dados da classe “MovimentoItem”

O Quadro 14 apresenta o dicionário de dados da classe “MovimentoItemId”.

Entidade: MovimentoItemId		
Atributo	**Tipo**	**Descrição**
idMovimento	int	Sequência do movimento
idItem	int	Sequência do item

Quadro 14 - Dicionário de dados da classe “MovimentoItemId”

O Quadro 15 apresenta o dicionário de dados da classe “Cliente”.

Entidade: Cliente		
Atributo	**Tipo**	**Descrição**
anoEntidade	int	Ano do exercício
codigoCliente	int	Código do cliente

Quadro 15 - Dicionário de dados da classe “Cliente”

O Quadro 16 apresenta o dicionário de dados da classe “EnderecoServico”.

Entidade: EnderecoServico		
Atributo	**Tipo**	**Descrição**
Id	int	Sequência dos endereços
Cliente	int	Código do cliente
nomeCliente	String	Nome do cliente
Endereço	String	Endereço de acesso do *web service*

Quadro 16 - Dicionário de dados da classe “EnderecoServico”

O Quadro 17 apresenta o dicionário de dados da classe “Fornecedor”.

Entidade: Fornecedor		
Atributo	**Tipo**	**Descrição**
Id	int	Sequência do fornecedor
Código	int	Código do fornecedor
CNPJ	String	Cnpj completo do fornecedor
Denominação	String	Razão social

Quadro 17 - Dicionário de dados da classe “Fornecedor”

O Quadro 18 apresenta o dicionário de dados da classe “Material”.

Entidade: Material		
Atributo	**Tipo**	**Descrição**
Id	int	Sequência do material
Descrição	String	Descrição do material
tipoCombustivel	TipoCombustivel	Tipo do combustível

Quadro 18 - Dicionário de dados da classe “Material”

O Quadro 19 apresenta o dicionário de dados da classe “NotaFiscal”.

Entidade: NotaFiscal		
Atributo	**Tipo**	**Descrição**
Id	int	Sequência da nota fiscal
Numero	int	Número da nota fiscal
Tipo	String	Tipo da nota fiscal

Fornecedor	Fornecedor	Relacionamento com o fornecedor
CNPJ	String	Cnpj do fornecedor
Pessoa	String	Tipo do fornecedor
dataEmissao	Date	Data da nota fiscal
valorTotal	double	Valor da nota fiscal

Quadro 19 - Dicionário de dados da classe “NotaFiscal”

O Quadro 20 apresenta o dicionário de dados da classe “Veiculo”.

Entidade: Veiculo		
Atributo	**Tipo**	**Descrição**
Id	int	Sequência do veículo
Descrição	String	Descrição do mateiral
Placa	String	Placa do veículo

Quadro 20 - Dicionário de dados da classe “Veiculo”