

**UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO**

**UM SISTEMA *WEB* DE INFORMAÇÕES
GEORREFERENCIADAS PARA ANÁLISE DE
REDES DE DISTRIBUIÇÃO DE ENERGIA
ELÉTRICA**

TRABALHO DE GRADUAÇÃO

Pedro Bastos Zorrilla

Santa Maria, RS, Brasil

2013

UM SISTEMA *WEB* DE INFORMAÇÕES GEORREFERENCIADAS PARA ANÁLISE DE REDES DE DISTRIBUIÇÃO DE ENERGIA ELÉTRICA

Pedro Bastos Zorrilla

Trabalho de Graduação apresentado ao Curso de Ciência da Computação
da Universidade Federal de Santa Maria (UFSM, RS), como requisito
parcial para a obtenção do grau de
Bacharel em Ciência da Computação

Orientador: Prof^a. Dr. Felipe Martins Müller

Co-orientadora: Prof. Dr^a. Luciane Neves Canha

**Trabalho de Graduação Nº 358
Santa Maria, RS, Brasil**

2013

**Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação**

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

**UM SISTEMA *WEB* DE INFORMAÇÕES GEORREFERENCIADAS
PARA ANÁLISE DE REDES DE DISTRIBUIÇÃO DE ENERGIA
ELÉTRICA**

elaborado por
Pedro Bastos Zorrilla

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Felipe Martins Müller, Dr.
(Presidente/Orientador)

Deise de Brum Saccol, Dr^a. (UFSM)

Eduardo Kessler Piveta, Dr. (UFSM)

Santa Maria, 18 de Fevereiro de 2013.

AGRADECIMENTOS

Aos meus pais, Diego Alberto Zorrilla Menendez e Darcy de Castro Bastos, por todo carinho e apoio, não medindo esforços para que eu chegasse até esta etapa de minha vida.

A minha namorada Danielle Vargas, pelo apoio incondicional e pela paciência nas horas em que me fiz ausente para realização desse trabalho, além das inúmeras correções de português sugeridas.

Ao professor Felipe e a professora Luciane, pela paciência na orientação e incentivo que se fizeram presentes no decorrer desse trabalho.

Ao PPGEE - UFSM (Programa de Pós Graduação em Engenharia Elétrica - UFSM) e ao CEESP (Centro de Estudos em Energia e Sistemas de Potência) pela infraestrutura física e pelas contribuições dos professores do grupo.

Ao Rafael Milbrat, por ter me iniciado no mundo da pesquisa e sempre me guiar nesse árduo caminho de seguir na área de processamento de energia elétrica, mesmo não sendo engenheiro eletricitista. A também pelas mais diversas contribuições para realização e conclusão desse trabalho.

A ANEEL (Agência Nacional de Energia Elétrica) pelo programa P&D ANEEL e à CEEE-D (Companhia Estadual de Energia Elétrica - Distribuição) pelo suporte financeiro e fornecimento dos dados que tornaram esse trabalho possível.

A todos os professores do curso, que foram tão importantes na minha vida acadêmica e no desenvolvimento desta monografia.

Aos amigos e colegas, pelo incentivo e pelo apoio constantes.

‘O único lugar aonde o sucesso vem antes do trabalho é no dicionário.’

— ALBERT EINSTEIN

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

UM SISTEMA *WEB* DE INFORMAÇÕES GEORREFERENCIADAS PARA ANÁLISE DE REDES DE DISTRIBUIÇÃO DE ENERGIA ELÉTRICA

AUTOR: PEDRO BASTOS ZORRILLA

ORIENTADOR: FELIPE MARTINS MÜLLER

CO-ORIENTADORA: LUCIANE NEVES CANHA

Local da Defesa e Data: Santa Maria, 18 de Fevereiro de 2013.

A energia elétrica é uma das formas de energia mais importantes para a sociedade e o desenvolvimento econômico. Com a demanda por energia cada vez mais crescente, as redes de distribuição são cada vez mais importantes para o sistema elétrico e desempenham um papel fundamental na melhoria da qualidade da energia. Com isso em vista, elas deverão sofrer uma intensa modernização nos próximos anos, principalmente com o surgimento das chamadas *Smart Grids* (redes inteligentes de energia). Nesse contexto os sistemas de computação e comunicação desempenharão um papel fundamental para a melhoria da qualidade da energia distribuída. É nesse cenário que o presente trabalho apresenta as etapas para o desenvolvimento de um Sistema de Informações Georreferenciadas (SIG) web, com a finalidade de prover uma melhor análise de alimentadores em redes de distribuição de energia elétrica. Uma infraestrutura de classes foi proposta para tornar a inclusão de novos algoritmos para análise da rede elétrica mais rápida e prática. Foram utilizadas a arquitetura de sistema em três camadas bem como tecnologias a fim de tornar o sistema flexível, escalável e independente de sistemas operacionais específicos, além de ser necessário somente um computador (ou até mesmo *tablets* ou *smartphones*) com algum navegador web instalado para utilizá-lo. A interface do sistema se baseia na tecnologia de mapas fornecida pela *API* do *Google Maps*, exibindo a topologia do alimentador em interface georreferenciada. Como resultados, este trabalho apresenta o *software* desenvolvido, demonstrando suas telas e outras interfaces com o usuário final, desde a configuração de quais equipamentos elétricos serão exibidos na interface georreferenciada, até a execução de algoritmos para análise elétrica.

Palavras-chave: Sistema de Informações Georreferenciadas. SIG. Arquitetura em três camadas.

ABSTRACT

Undergraduate Final Work
Computer Science Course
Federal University of Santa Maria

DEVELOPMENT OF A WEB GEOREFERENCED INFORMATION SYSTEM (WEBGIS) INTO THREE LAYERS SOFTWARE ARCHITECTURE FOR ANALYSIS OF PRIMARY DISTRIBUTION NETWORKS

AUTHOR: PEDRO BASTOS ZORRILLA

ADVISOR: FELIPE MARTINS MÜLLER

COADVISOR: LUCIANE NEVES CANHA

Defense Place and Date: Santa Maria, February 18th, 2013.

Electricity is one of the most important forms of energy to society and economic development. With demand for energy growing increasingly, distribution networks are more and more important to the electrical system and play a key role in improving power quality. They will experience intense modernization over the coming years, especially with the emergence of the Smart Grid initiatives. In this context, computer and communication systems will play a vital role in improving the quality of distributed energy. It is against this background this paper presents the steps for developing a Web Geographic Information System (WebGIS), in order to provide analysis of medium voltage feeders on electricity distribution networks. A class infrastructure has been proposed to make adding new algorithms for power analysis faster and more practical. It has also been used the three-tier software architecture allied to technologies in order to make the system flexible, scalable and independent of specific operating systems, as well as being only needed a computer (even tablets or smartphones) with a web browser installed in order use the software. The software interface is based on the maps technology provided by Google Maps API, displaying the feeder's interface in a georeferenced way. As the results, this work presented the software developed, demonstrating screens and other interfaces with the end user, starting with the configuration of which electrical equipment are going to be displayed in the georeferenced interface, until the execution of power systems's analysis algorithms.

Keywords: WebGis,GIS,three-tier software architecture.

LISTA DE FIGURAS

Figura 2.1 – Arquitetura de sistemas em três camadas. Adaptada de (HORSTMANN, 2005, Capítulo 27)	17
Figura 2.2 – Diagrama de uma requisição na arquitetura em três camadas	18
Figura 2.3 – Estrutura e fluxo de uma requisição/resposta típica do protocolo HTTP (HOCK-CHUAN, 2013)	19
Figura 2.4 – Injeção de dependência via construtor e método de escrita, respectivamente	20
Figura 2.5 – Componentes do <i>framework</i> GWT (PAWLAK et al., 2009)	22
Figura 3.1 – Diagrama de casos de uso do sistema	25
Figura 3.2 – Desenho dos principais componetes de um alimentador. Adaptado de (Rede inteligente, 2009)	27
Figura 3.3 – Diagrama das principais classes do sistema	28
Figura 3.4 – Esquema lógico do banco de dados relacional do sistema	30
Figura 3.5 – Parte do arquivo XML de configuração do Hibernate (hibernate.cfg.xml)	31
Figura 3.6 – Classe <i>AsdDAO</i> e parte de seus métodos persistentes	32
Figura 3.7 – Classe <i>AlgoritmoAbstrato</i> com seus métodos e atributos	33
Figura 3.8 – Interface <i>Navegavel</i>	34
Figura 3.9 – Interfaces <i>AcaoPrefixada</i> e <i>AcaoPosFixada</i> , respectivamente	35
Figura 3.10 – Parte principal da classe <i>Navegador</i>	36
Figura 3.11 – Ilustração de como o algoritmo para somar as correntes funciona	37
Figura 3.12 – Classe <i>CalculaSomaCorrentesAcaoPos</i> , responsável por somar as correntes acumuladas até o nó atual e retornar o somatório ao seu nó pai	38
Figura 3.13 – Classe <i>CalcularSomaCorrentesAlgoritmo</i> , responsável por realizar o algoritmo de calculo do somatório das correntes	39
Figura 3.14 – Parte da classe <i>Usuario</i> e parte de seu DTO <i>UsuarioCli</i> , destacando-se a implementação do método da interface <i>ClasseCliTransformadorInterface</i>	40
Figura 3.15 – Atributos e parte dos métodos da classe <i>AsdServiceImpl</i>	41
Figura 3.16 – Interfaces <i>AsdService</i> e <i>AsdServiceAsync</i> , respectivamente	42
Figura 3.17 – Trecho de código da chamada remota do método <i>getAlimentadoresIds</i>	43
Figura 4.1 – Tela inicial de login do sistema	46
Figura 4.2 – Interface para o carregamento de um alimentador	47
Figura 4.3 – Tela principal de exibição do alimentador de média tensão	48
Figura 4.4 – legenda de ícones dos equipamentos	49
Figura 4.5 – Tela principal exibindo somente transformadores, e o menu de configuração de exibição dos equipamentos elétricos	50
Figura 4.6 – <i>Popup</i> de exibição dos atributos elétricos de uma chave seccionadora	51
Figura 4.7 – Diálogo de configuração de quais atributos inseridos por algoritmos serão exibidos	52
Figura 4.8 – Menu para execução dos algoritmos, exemplificando a execução do algoritmo <i>Caminho mais longo</i>	53

LISTA DE ABREVIATURAS E SIGLAS

AJAX	<i>Asynchronous JavaScript and XML</i>
API	<i>Application Program Interface</i>
CEEE-D	<i>Companhia Estadual de Energia Elétrica - Distribuição</i>
CRUD	<i>Create Read Update and Delete</i>
DTO	<i>Data Transfer Object</i>
GIS	<i>Geographic Information System</i>
GWT	<i>Google Web Toolkit</i>
HTTP	<i>Hypertext Transfer Protocol</i>
IOC	<i>Inversão de Controle</i>
JEE	<i>Java Enterprise Edition</i>
JDBC	<i>Java Database Connectivity</i>
JSR	<i>Java Specification Requests</i>
RPC	<i>Remote Procedure Call</i>
SQL	<i>Structured Query Language</i>
SIG	<i>Sistema de Informações Georeferenciadas</i>
XML	<i>eXtensible Markup Language</i>

SUMÁRIO

1 INTRODUÇÃO	11
1.1 Objetivos	13
1.2 Organização do texto	13
2 REVISÃO BIBLIOGRÁFICA	15
2.1 SIG em sistemas de energia elétrica	15
2.2 Arquitetura de sistemas em três camadas e tecnologias usadas	16
2.2.1 Spring	19
2.2.2 Hibernate	20
2.2.3 GWT (<i>Google Web Toolkit</i>)	21
2.3 Conclusão do capítulo	23
3 DESENVOLVIMENTO	24
3.1 Visão geral e diagramas UML	24
3.1.1 Diagrama de casos de uso	25
3.1.2 Diagrama de classes	26
3.2 Implementação	29
3.2.1 Modelagem do banco de dados	29
3.2.2 Mapeamento objeto-relacional	31
3.2.3 Estrutura para execução de algoritmos de análise em redes elétricas	33
3.2.3.1 Classe <i>AlgoritmoAbstrato</i>	33
3.2.3.2 Interface <i>Navegavel</i>	34
3.2.3.3 A classe <i>Navegador</i> e as interfaces <i>AcaoPreFixada</i> e <i>AcaoPosFixada</i>	35
3.2.3.4 Exemplo de implementação de algoritmo	36
3.2.4 Comunicação cliente-servidor com GWT	39
3.2.4.1 Definição das classes DTO (<i>Data Transfer Objects</i>)	39
3.2.4.2 Parte servidor do mecanismo RPC	41
3.2.4.3 Parte cliente do mecanismo RPC	42
3.3 Conclusão do capítulo	43
4 RESULTADOS	45
4.1 Telas	45
4.1.1 Tela de <i>login</i>	45
4.1.2 Tela principal de exibição do alimentador	46
4.2 Outras interfaces	50
4.2.1 <i>Popup</i> de exibição dos atributos elétricos	51
4.2.2 Diálogo para configuração de quais atributos elétricos serão exibidos	51
4.2.3 Execução de algoritmos de análise em redes elétricas	53
4.3 Conclusão do capítulo	54
5 CONCLUSÕES	55
REFERÊNCIAS	56

1 INTRODUÇÃO

A energia elétrica é uma das formas de energia mais importantes para a sociedade e o desenvolvimento econômico. O sistema de energia, para efeito de análise, pode ser dividido em três grandes blocos: geração, transmissão e distribuição. A primeira etapa de geração é onde a energia potencial de uma queda d'água, por exemplo, é transformada em energia elétrica por meio de um gerador, como no caso da geração em hidrelétricas, que é a forma mais comum no Brasil. Após a elevação da tensão, é iniciada a fase de transmissão: essa energia é transportada através de condutores para as subestações. Essas subestações normalmente reduzem a tensão para uma faixa de 13,8kV, e a partir delas inicia-se a distribuição, a qual é seccionada ainda em dois tipos de rede: rede de distribuição de média e baixa tensão, sendo a primeira a rede que tem como fonte uma subestação e estende-se até os transformadores de distribuição, onde a tensão tipicamente de 13,8kV é reduzida para valores entre 120V e 380V, e a segunda a rede desses transformadores até os consumidores finais.

Com o passar dos anos, os sistemas de computação estão se tornando cada vez mais presentes no nosso dia a dia. Com o avanço tecnológico e a crescente necessidade de se obter soluções mais rápidas e ao mesmo tempo eficientes, a demanda por esses sistemas tem crescido exponencialmente. Na área de processamento de energia elétrica essa realidade não é diferente. No Brasil, a crise energética do ano 2000 até 2001 causou um rombo de cerca de R\$ 45,2 bilhões aos cofres públicos, tendo sido estimado que os consumidores finais fossem responsáveis por arcar cerca de 60% desses custos, através de reajustes tarifários (BANCILLON, 2009). Em outubro de 2012, uma falha humana no nordeste deixou moradores dos seus nove estados, além de parte do Tocantins e do Pará, sem energia elétrica (NEVES; MIDIA, 2012; AMATO, 2012). Aliando isso ao aumento cada vez maior da demanda e a crescente preocupação com um planeta sustentável, a rede de energia elétrica atual é muito complexa e mal adaptada às necessidades do século vinte e um (GUNGOR et al., 2011).

É nesse cenário que uma nova proposta de rede de energia surge, a chamada rede inteligente, do inglês *Smart Grid*. Essas redes inteligentes serão caracterizadas pelo uso de modernas tecnologias de informação e comunicação, promovendo a integração de fontes de energia renováveis à rede, a utilização de medidas remotas e em tempo real

provenientes de medidores inteligentes e também a comunicação em duas vias entre a concessionária de energia e o consumidor final (GUNGOR et al., 2011; PANAJOTOVIC; JANKOVIC; ODADZIC, 2011; PARIKH; NIELSEN, 2009). Nesse contexto de redes inteligentes, os sistemas de informação e comunicação têm grande importância para a viabilidade das mesmas (PANAJOTOVIC; JANKOVIC; ODADZIC, 2011). Um desses sistemas de computação que se mostrarão presentes na operação das novas redes inteligentes são os Sistemas de Informações Georreferenciadas (SIG), como exemplificam os trabalhos em Parikh e Nielsen (2009), Damiano et al. (2010) e Romero et al. (2011). Os sistemas de Informações Georreferenciadas são ferramentas muito úteis para exibição e análise de informações (SYLLIGNAKIS; ADAMAKIS; PAPAZOGLU, 2007).

Liu (2010) aponta que com a demanda energética cada vez mais crescente, as redes de distribuição são muito importantes para o sistema elétrico e desempenham um papel fundamental na melhoria da qualidade da energia. Objetivando uma melhora nessas e em outras questões, elas deverão sofrer uma intensa modernização nos próximos anos através da instalação de novos equipamentos que permitam o monitoramento e operação de forma remota em um número cada vez maior de nós da rede de distribuição. Neste sentido os sistemas de computação que permitem a análise e operação destas redes terão uma grande importância. De acordo com Chao et al. (2010), os SIG podem diminuir significativamente a sobrecarga de trabalho em redes de distribuição de energia elétrica, tornando mais eficiente o despacho das equipes de manutenção, melhorando a taxa de tratamento de incidentes e reduzindo as perdas causadas por falhas. Em outros trabalhos como Rao et al. (2008), um SIG é apontado como uma poderosa ferramenta para visualização de uma rede de distribuição, permitindo ao engenheiro do sistema visualizar as áreas com baixas tensões e maiores perdas, além de proporcionar uma melhor análise de novos clientes e subestações para atender ao aumento da demanda. É nesse cenário que o presente trabalho se torna justificado.

Com isso em mente, este trabalho tem como foco o desenvolvimento de um Sistema de Informações Georreferenciadas (SIG) para análise e processamento de informações topológicas de um alimentador em redes de distribuição. As informações geradas pelo sistema também poderão servir como apoio para análise e até mesmo, num futuro, a própria operação do sistema de distribuição, visando a melhora nos indicadores de qualidade da energia distribuída. Este trabalho utilizou para os testes dados de um alimentador

(sub-rede de uma subestação) cedidos pela Companhia Estadual de Energia Elétrica – Distribuição (CEEE-D). A partir desses dados foi construída uma interface para que todas as operações de visualização e interação com o sistema estejam disponíveis através da Web, através das tecnologias Spring, Hibernate e GWT (*Google Web Toolkit*).

1.1 Objetivos

Esse trabalho tem como principal objetivo o desenvolvimento de um SIG em três camadas com interface web, usando ferramentas e *frameworks* de código-livre. Como requisitos desse sistema, se destacam:

- O sistema deverá ser capaz de mostrar e analisar informações referentes a um diagrama unifilar de um alimentador de uma rede de distribuição.
- As Informações elétricas e topológicas deverão ser exibidas em interface agradável com apoio da base cartográfica do Google Maps.
- O sistema deverá dar suporte para o desenvolvimento e uso de algoritmos de análise de alimentadores em redes de energia elétrica, através da utilização de uma plataforma de classes padrão para a implementação e uso dos mesmos.
- O sistema deve ser capaz de ser executado em qualquer dispositivo que tenha acesso a internet e a algum navegador web. Isto torna o sistema independente de sistemas operacionais específicos, além de minimizar a quantidade de programas instalados necessários para a sua execução.

1.2 Organização do texto

Esse trabalho está organizado da seguinte maneira: O capítulo 2 apresenta uma fundamentação teórica sobre os Sistemas de Informações Georreferenciadas e apresenta alguns trabalhos que indicam os usos dos mesmo em sistemas de energia elétrica. Ainda nesse capítulo é apresentada a arquitetura de sistemas em três camadas, bem como as tecnologias utilizadas nesse trabalho para implementação dessa arquitetura (Spring *framework*, Hibernate e GWT (*Google Web Toolkit*)). O capítulo 3 apresenta os principais passos seguidos para o desenvolvimento desse sistema. O capítulo 4 apresenta os resulta-

dos obtidos dos passos realizados no capítulo 3, e ilustra as interfaces do SIG desenvolvido. Por fim, o capítulo 5 apresenta as conclusões e possíveis trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

Nesse capítulo será apresentada a fundamentação teórica e a contextualização de trabalhos relacionados aos temas:

1. SIG, principalmente os quais foram desenvolvidos e/ou utilizados para aplicação em sistemas de energia elétrica.
2. Tópicos referentes ao desenvolvimento de sistemas de informação utilizando o modelo de arquitetura de sistemas em três camadas (camada de dados, camada de negócio e camada de apresentação), bem como as tecnologias utilizadas nesse trabalho para esse modelo de sistema.

2.1 SIG em sistemas de energia elétrica

Kasturi et al. (1989) definiu um SIG como um sistema de computação que permite a seus usuários coletar, manipular e analisar grandes volumes de dados georreferenciados. Nesses sistemas, essas informações são representadas por linhas, pontos ou polígonos em mapas (CHAURASIA; THAKUR, 2009). Com a evolução das tecnologias de computação e comunicação, esses SIG's tradicionalmente disponíveis apenas para computadores pessoais estão evoluindo para sistemas mais robustos e acessíveis em qualquer parte do mundo com informações georreferenciadas disponíveis através da Internet (FENG, 2010; FU; SUN, 2010, pp. 1). Vários trabalhos vêm tirando proveito dessas novas tecnologias, entre eles podemos destacar alguns que foram mais recentemente realizados.

Monteiro et al. (2005) desenvolveu uma metodologia para minimizar os custos de instalação de novas linhas de transmissão, por meio da automatização do processo de busca do melhor caminho para roteamento de linhas ponto a ponto, utilizando também um sistema de informações georreferenciadas. Essa metodologia foi implementada na linguagem de programação C++ e integrada ao SIG comercial ArcView (atual ArcGIS Desktop).

O SIG ArcGIS também foi utilizado por Chaurasia et al. (2009) para Indexar consumidores de uma rede de distribuição com a sua localização exata ao longo de um alimentador, através do uso de GPS e de imagens de alta qualidade via satélite, proporcionando acesso a informações, tais como encontrar o transformador que um consumidor

específico está utilizando. Em seu trabalho também foi utilizada a ferramenta Google Earth para auxílio na digitalização de imagem das casas dos consumidores.

Já Rao et al. (2008) utilizou os softwares POWERGIS e POWERNET para a construção de um diagrama unifilar de uma rede de distribuição indiana, ilustrando também uma base de dados com cerca de 45 mil clientes. Nesse trabalho são apontadas as principais vantagens da utilização de um SIG em áreas como análise e planejamento de sistemas de distribuição.

Em Triplett et al. (2010), foi utilizada uma Infraestrutura de Medição Avançada, que pode ser resumida como um conjunto de medidores inteligentes capazes de monitorar, comunicar e otimizar o uso de energia (SELVAM et al., 2012), em conjunto a um SIG, para calcular as perdas mensais ao longo de uma rede de distribuição. Foram estimadas as perdas em contas de usuários, condutores secundários, transformadores de distribuição, condutores primários e subestação (incluindo bancos de reguladores e alimentadores de saída) com o uso de dados georreferenciados.

2.2 Arquitetura de sistemas em três camadas e tecnologias usadas

Os programas de computador antigamente eram desenvolvidos para uma única máquina, atrelando todas as tarefas e funcionalidades do mesmo a uma arquitetura de hardware específica, além de tornar o trabalho de manutenção da aplicação complicado e trabalhoso. Principalmente com o surgimento e popularização da internet, hoje em dia esses programas evoluíram para grandes sistemas, tornando-se uma das estruturas mais complexas já construídas por seres humanos (ROZANSKI; WOODS, 2011, pp. 5).

Com a crescente complexidade desses sistemas, uma das arquiteturas que vem sendo bastante utilizada é a arquitetura em três camadas, como pode ser visto em vários trabalhos da literatura, como em (ZECEVIC, 1998; QIU; GOOI, 2000; FENG, 2010; ROMERO et al., 2011; LIDUO; YAN, 2010). Essa arquitetura visa tornar uma aplicação mais bem organizada, através da separação do sistema em três camadas distintas, objetivando minimizar a interdependência das mesmas e assim facilitar o trabalho de manutenção do sistema como um todo, além de encapsular em cada camada uma preocupação específica do sistema (QINGWU; JIAXUAN, 2009; LIDUO; YAN, 2010; HU; YU, 2012).

As três camadas recebem os nomes de camada de dados, camada de negócios e camada de apresentação, e sua estrutura pode ser observada na Figura 2.1. A camada

de dados é a responsável por fornecimento dos dados e atributos relacionados a uma determinada requisição. A camada de negócios nada mais é responsável por uma série de processamentos desses dados, necessários para responder a requisição e a apresentação final dos mesmos. Por fim, a camada de apresentação é necessária para formatação de como esses dados devem ser mostrados para o cliente final. Os componentes existentes em camadas diferentes devem ser fracamente acoplados, garantindo sua independência. Assim, um componente de negócios em um servidor de aplicação pode oferecer seus serviços a vários tipos de clientes, cada um com uma camada de apresentação diferente.

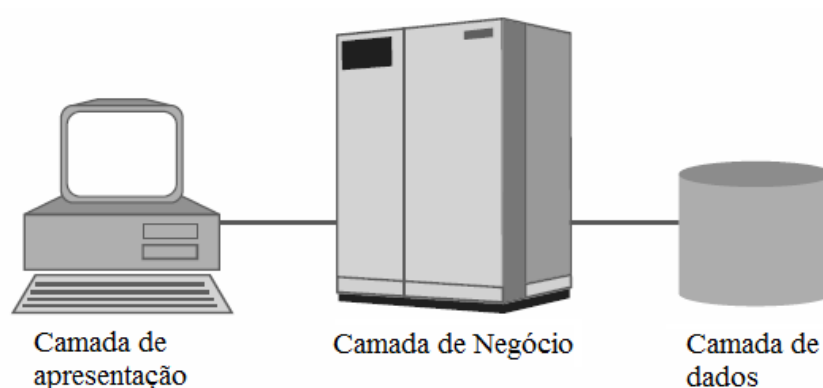


Figura 2.1 – Arquitetura de sistemas em três camadas. Adaptada de (HORSTMANN, 2005, Capítulo 27)

O fluxo de uma requisição nessa arquitetura pode ser observado na Figura 2.2. O cliente (camada de apresentação) envia uma requisição ao servidor de aplicação (camada de negócios). Essa requisição é então processada e, caso a requisição envolva criar, pesquisar, atualizar ou remover algum dado persistente, o banco de dados é consultado. Essas operações são chamadas de CRUD (Create, Read, Update and Delete) na literatura (PEREIRA; AGUIAR; SANTOS, 2010). Com os dados necessários retornados do banco de dados, o servidor de aplicação aplica uma lógica para manipulação desses dados (daí o nome lógica de negócio) e envia o resultado dessa requisição de volta para o cliente.

Um dos protocolos mais utilizados nesse contexto é o protocolo HTTP (*Hypertext Transfer Protocol*), que é um protocolo de aplicação para sistemas de informação distribuídos multimídia, baseando-se no modelo de requisição/resposta (Cliente/Servidor). Ele vem sendo usado através da Internet desde a década de noventa, e suas principais características são: (1) possibilitar a comunicação entre clientes e servidores, (2) ser bastante flexível permitindo a transmissão de quaisquer tipo de dados e (3) ser um protocolo

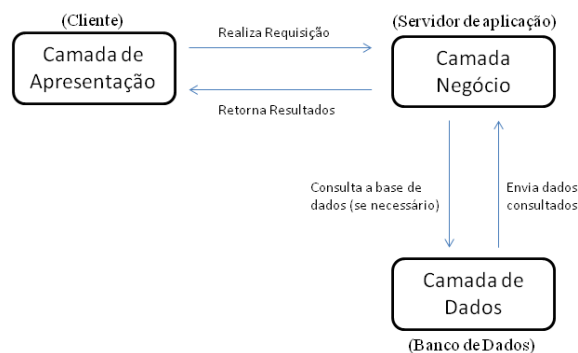


Figura 2.2 – Diagrama de uma requisição na arquitetura em três camadas

stateless, ou seja, os servidores não armazenam dados relativos aos clientes que fazem as requisições (como se um cliente já fez um login, por exemplo), permitindo uma fácil alocação de recursos. As mensagens desse protocolo seguem o padrão US-ASCII e consistem de requisições originadas de clientes (normalmente através de navegadores web) para servidores, e respostas desses servidores de volta aos clientes (YU; WU; YIN, 2003).

De acordo com Shuai et al. (2008), as estatísticas apontam que a imensa maioria das respostas de requisições HTTP enviadas por servidores é composta de arquivos de imagem e principalmente páginas HTML. A Figura 2.3 ilustra uma requisição/resposta típica do protocolo HTTP. Clientes através de navegadores web e aplicações realizam uma requisição, a qual no exemplo abaixo solicita o conteúdo da pagina */home.html* do *host xyz.com*. Após o servidor HTTP processar essa requisição, ele envia o resultado de volta para o cliente. Nesse exemplo, o cabeçalho dessa resposta ilustra que conteúdo da requisição foi encontrado, tem 105 bytes de tamanho e é em formato de texto em html.

Existem diversos servidores HTTP no mercado, mas um dos mais populares é o Apache. O projeto Apache servidor HTTP é um projeto de esforço contínuo para desenvolver e manter um servidor de resposta para requisições HTTP gratuito para modernos sistemas operacionais incluindo UNIX e Windows, além de ser originado e mantido pela Apache Software Foundation. Ele foi desenvolvido na linguagem de programação C e tem sido o servidor web mais popular da internet desde abril de 1996 (Apache HTTP, 2012).

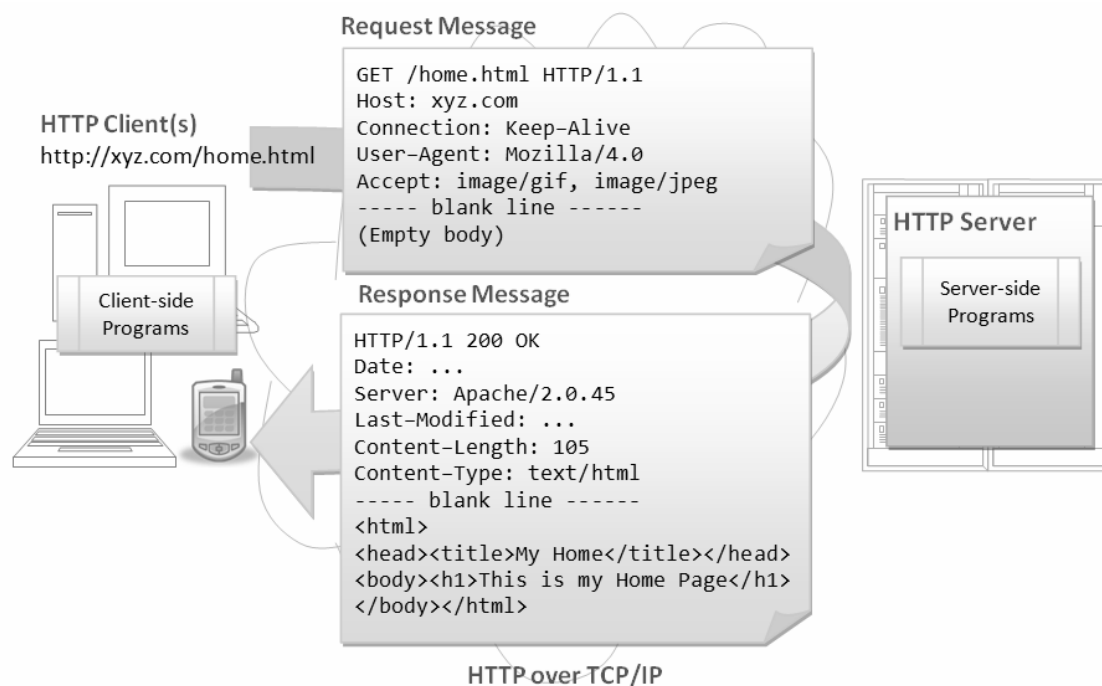


Figura 2.3 – Estrutura e fluxo de uma requisição/resposta típica do protocolo HTTP (HOCK-CHUAN, 2013)

2.2.1 Spring

Uma das técnicas para otimizar a produção no desenvolvimento de aplicações é diminuir a complexidade desse processo. Essa tem sido uma forte área de estudos, com o intuito de transformar a maneira com que programadores Java desenvolvem aplicações para *Web*. Nesse contexto, o uso do *framework* Spring é uma das soluções que vem sido predominantemente utilizada no contexto de java (ARTHUR; AZADEGAN, 2005; JOHNSON, 2005).

O Spring é um *framework* Java que provê uma infraestrutura de fácil compreensão para auxiliar no desenvolvimento de aplicações. Dois dos principais conceitos que esse *framework* implementa são chamados de Inversão de controle (IOC) e de Injeção de dependência. Segue a definição de ambos esses conceitos (JOHNSON et al., 2012):

- O conceito de inversão de controle tem por base, passar a um *framework* externo o controle da aplicação. Quando necessário, o *framework* passa ao usuário o controle da aplicação para realização de uma determinada tarefa e após a conclusão da mesma, esse controle é retornado para o *framework*. Um dos principais exemplos de Inversão de Controle são as interfaces com usuários. Normalmente uma inter-

face desse tipo tem um programa principal rodando um *loop* infinito e somente é entregue o controle ao usuário quando o mesmo faz uma entrada de texto ou clica em algum botão. Essa interação é tratada pelos *Listeners*, e então o *loop* principal é reativado.

- Já na injeção de dependências, segundo Johnson (2004, pp. 131) o principal objetivo é tirar dos objetos da aplicação a responsabilidade de procurar por objetos e recursos que colaboram com eles ou dos quais eles dependem. Ao invés disso, um *container* de inversão de controle externo deve ser o responsável por essa tarefa. Existem dois principais tipos de injeção de dependências: por construtores e por métodos de escrita. Esses dois tipos são ilustrados na Figura 2.4. No primeiro tipo, a dependência de um objeto *MovieFinder* pelo objeto *MovieLister* é sanada através da injeção do objeto *MovieFinder* como parâmetro diretamente no método construtor do objeto *MovieLister*. Já no segundo, o objeto *MovieLister* possui um método de escrita que fará a injeção do objeto *MovieFinder*.

<pre>public class MovieLister { private MovieFinder finder; //Construtor public MovieLister(MovieFinder finder) { this.finder = finder; } }</pre> <p style="text-align: center; color: red;">Injeção por Construtor</p>	<pre>public class MovieLister { private MovieFinder finder; public void setFinder(MovieFinder finder) { this.finder = finder; } }</pre> <p style="text-align: center; color: red;">Injeção por Setter</p>
--	--

Figura 2.4 – Injeção de dependência via construtor e método de escrita, respectivamente

2.2.2 Hibernate

O modelo de programação utilizando JDBC diretamente para acesso e gerenciamento de dados persistentes é propenso a erros, visto que limitava os conceitos da orientação a objetos, forçando os programadores a trabalhar com conceitos relacionais em códigos java (JOHNSON, 2005). Uma das soluções propostas para esse problema é o mapeamento objeto-relacional. Esse mapeamento utiliza metadados com o intuito de fornecer persistência automática e transparente de objetos Java para tabelas em banco de dados relacionais. Com esse mapeamento, é possível melhorar a produtividade e aumentar a capacidade de manutenção de programas, fornecendo um melhor desempenho

no processo de desenvolvimento, além de permitir o uso de diferentes bancos de dados através de uma única interface (WU; YIN, 2010).

Um dos mais populares e utilizados *framework* que implementam mapeamento objeto-relacional é o Hibernate (JOHNSON, 2005; WU; YIN, 2010). Além de ser um *framework* de *software* livre, ele possui a HQL (*Hibernate Query Language*) que é muito parecida com o SQL. No entanto, a HQL é voltada para consultas orientadas a objetos, e compreende noções de herança, polimorfismo e associações (KINGAND et al., pp. 269). Com o intuito de otimizar o desempenho de operações persistentes, o Hibernate utiliza a técnica de *cache*, que consiste na utilização de algoritmos para armazenamento temporário de entidades persistentes principalmente em memória principal (RAM) (JING; FAN, 2011).

Existem duas maneiras principais de se realizar o mapeamento objeto-relacional com o *framework* Hibernate: através de arquivos XML ou anotações Java. No mapeamento por anotação, o mapeamento é feito diretamente no código fonte da classe, sendo a mesma marcada como persistente através da anotação *@Entity*. Já no mapeamento por XML, a classe é mapeada através de um arquivo XML externo à classe e vinculado à mesma através do arquivo de configuração do *Hibernate*.

2.2.3 GWT (*Google Web Toolkit*)

Com o surgimento da web 2.0, introduziu-se um novo conceito de aplicações voltadas para internet: elas abandonariam a ideia de que navegadores seriam apenas terminais burros para exibir dados preparados pelo servidor, e sim tornariam-se clientes com maiores recursos para interação entre cliente-servidor, através da tecnologia AJAX (*Asynchronous JavaScript and XML*). Com o uso dessa tecnologia, as aplicações web tornaram-se mais parecidas com as aplicações para *desktops* que possuem arquitetura distribuída. Em contrapartida, com o crescimento da complexidade de aplicações AJAX aliado ao fato de que essa tecnologia nunca teve como foco ser utilizada estritamente em aplicações *web* de larga escala, faltam às mesmas ferramentas como IDEs (*Integrated Development Environments*) que dão suporte ao desenvolvimento dessas aplicações da mesma maneira que aplicações *desktop*. Aliando tudo isso ao fato que navegadores interpretam código *javascript* de maneira diferente, o desenvolvimento dessas aplicações de larga escala se

tornou um problema bastante complexo (PAWLAK et al., 2009).

Com esses problemas em mente, a Google lançou um *kit* de desenvolvimento chamado GWT (*Google Web Toolkit*). GWT é um conjunto de ferramentas AJAX de código livre para o desenvolvimento de aplicações de interfaces gráficas mais ricas puramente em linguagem Java (PAWLAK et al., 2009; GOOGLE, 2012). A Figura 2.5 mostra os componentes mais importantes desse *kit* de desenvolvimento.

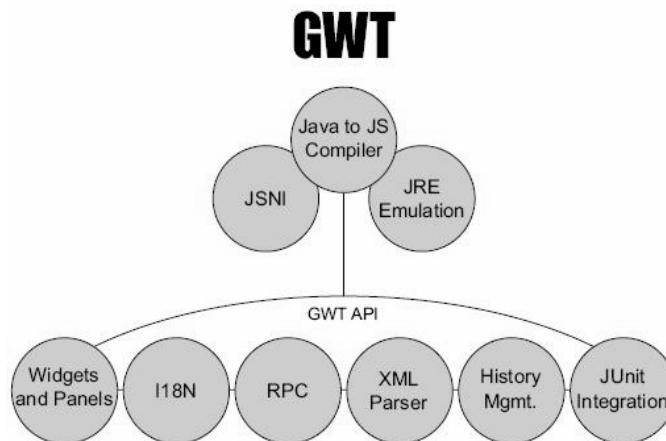


Figura 2.5 – Componentes do *framework* GWT (PAWLAK et al., 2009)

O GWT possui um compilador otimizado que traduz código Java em código *JavaScript* e *HTML*, permitindo aos desenvolvedores evitarem a complexidade das tecnologias que transformam a escrita de aplicações AJAX propensa a erros. Isso se concretiza através da capacidade de depuração da aplicação no chamado *Hosted Mode*. Esse modo permite a depuração em tempo de execução da camada de apresentação, fazendo a correspondência em tempo de execução do código *JavaScript* para o trecho de código Java correspondente, permitindo a depuração de códigos na camada de apresentação (PAWLAK et al., 2009; MELIA et al., 2008; SONG; LI; FANG, 2010; XIANJUN; XUXIAN; HONGQUAN, 2011; GOOGLE, 2012). Além disso, o GWT tem a capacidade de gerar código *JavaScript* otimizado para diversos navegadores, bem como possui o mecanismo RPC (*Remote Procedure Call*) para comunicação assíncrona de maneira simples entre a camada de apresentação e a camada da lógica de negócios (PAWLAK et al., 2009; GOOGLE, 2012).

2.3 Conclusão do capítulo

Nesse capítulo inicialmente foram apresentados os conceitos de um SIG, e então foram ilustrados alguns trabalhos que utilizaram esses conceitos para melhorar a análise de sistemas de transmissão e principalmente distribuição de energia elétrica.

Em seguida foi descrita a arquitetura de sistemas em três camadas, apresentando sua ideia principal (separação lógica do sistema em três camadas lógicas: dados, negócio e apresentação), além de dissertar um pouco sobre o protocolo *HTTP*, o qual é considerado um dos mais importantes no contexto dessa arquitetura de aplicação. Por fim, algumas ferramentas utilizadas nesse trabalho com o intuito de facilitar o desenvolvimento e manutenção dos sistemas em três camadas (*Spring framework*, *Hibernate* e *GWT*) foram explanadas e tiveram seu uso justificado.

3 DESENVOLVIMENTO

Este capítulo inicialmente descreve a visão geral de como o sistema foi implementado, além de mostrar os diagramas de caso de uso e de classes do sistema, e em seguida apresentar os passos realizados para sua implementação.

3.1 Visão geral e diagramas UML

O principal objetivo do sistema a ser implementado é que as informações referentes a um alimentador de média tensão sejam exibidas em uma interface georreferenciada. Para isso, um dos primeiros passos tomados foi descrever as funcionalidades desse sistema, bem como as classes utilizadas para sua construção. Em seguida, foi realizada a modelagem relacional das entidades necessárias em um ambiente de distribuição primária de energia elétrica (subestação original, trechos de rede, transformadores de distribuição, etc). Após essa modelagem, foi adquirido um arquivo *xls* com os dados de uma subestação da concessionária de distribuição de energia elétrica CEEE-D. Nesse arquivo estavam localizadas todas as informações elétricas (equipamentos utilizados, valores das grandezas elétricas nominais, etc.) e topológicas (coordenadas da localização de postes e equipamentos, comprimento de trechos de redes, entre outros) de um alimentador de média tensão. Com o intuito de tornar o acesso a esses dados mais rápido e estruturado, eles foram importados no banco de dados relacional PostgreSQL. Para fazer a correlação do conteúdo do *xls* com a modelagem lógica proposta, foi utilizado o *framework* Hibernate para persistência dos dados.

Já com os dados mapeados, foi implementada uma plataforma para execução de algoritmos de análise da rede (fluxo de potência, estimação de estado, entre outros) com a finalidade de tornar a inclusão de novos algoritmos para análise das redes de distribuição mais rápidas e simples. Essa estrutura baseou-se na construção de classes concretas e classes abstratas, bem como na definição de interfaces para padronizar o modo como esses algoritmos operariam. Por fim, a estrutura para a comunicação entre cliente e servidor foi definida, utilizando o mecanismo de ARPC (*Asynchronous Remote Procedure Call*) do *kit* de desenvolvimento GWT.

3.1.1 Diagrama de casos de uso

Diagramas de casos de uso UML são importantes para visualizar, especificar e documentar o comportamento de um elemento. Eles são aplicados para demonstrar o comportamento desejado do sistema a ser desenvolvido sem especificar como o comportamento é implementado logicamente. Esse diagrama é uma das peças mais importantes para comunicação entre os analistas de sistemas e o cliente final, sendo uma peça fundamental no levantamento de requisitos do sistema (SENGUPTA; BHATTACHARYA, 2006). A Figura 3.1 demonstra o diagrama de casos de uso para o sistema desenvolvido.

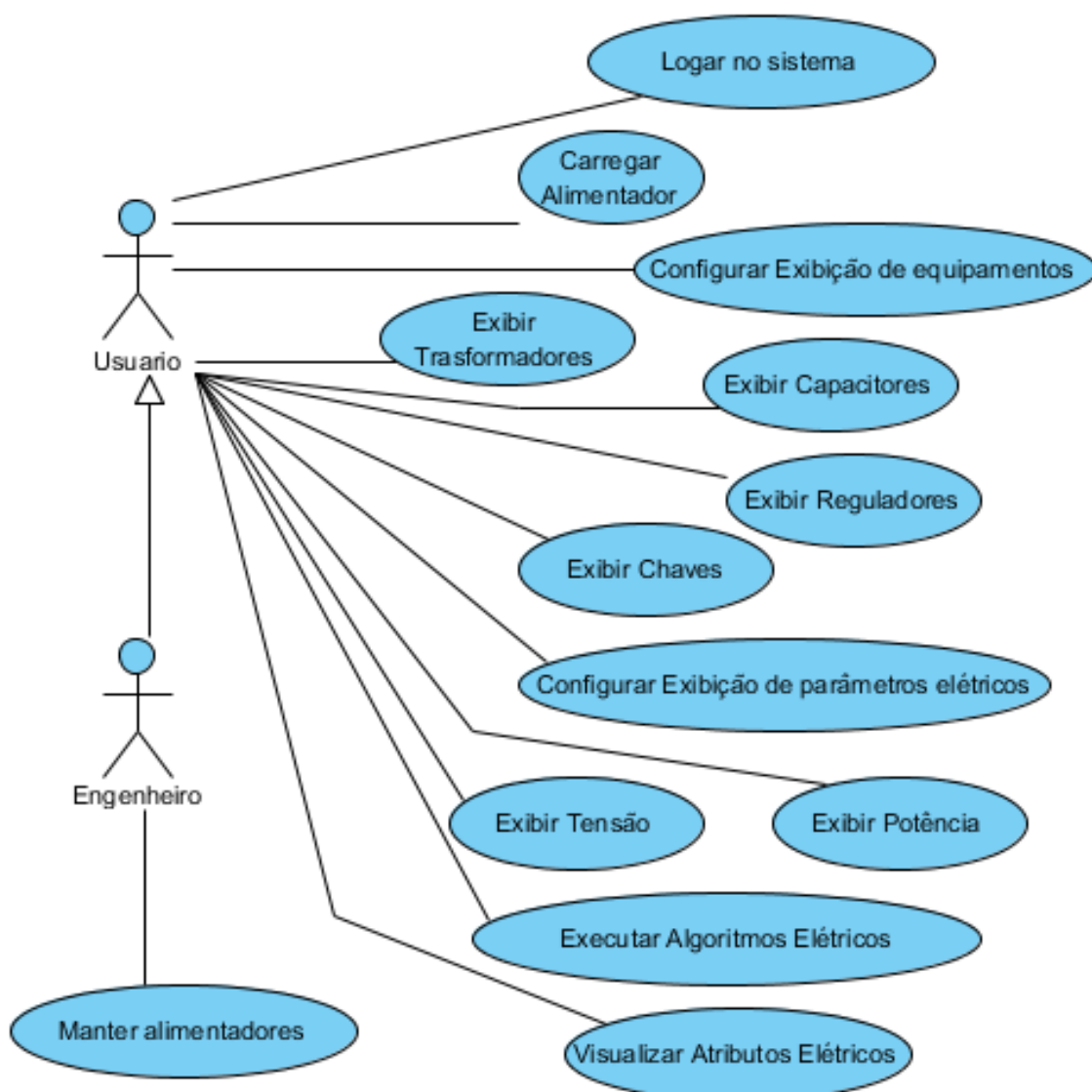


Figura 3.1 – Diagrama de casos de uso do sistema

Os atores no sistema são usuários e engenheiros. O engenheiro pode realizar todas as operações que um usuário, com adição de cadastrar novos alimentadores. As operações

permitidas por um usuário são:

- Logar no sistema: O usuário fará uma autenticação para acessar o sistema, fornecendo um nome de usuário e senha.
- Carregar Alimentador: O usuário poderá carregar a topologia de um alimentador já previamente cadastrado no sistema, exibindo seus componentes (trechos de rede e equipamentos elétricos) numa interface georreferenciada.
- Configurar Exibição de equipamentos: O usuário poderá configurar quais equipamentos deverão aparecer na interface georreferenciada, tendo a opção de exibir ou não os equipamentos: transformadores de distribuição, chaves seccionadoras, capacitores e reguladores de tensão. A configuração deverá ser independente para cada equipamento.
- Configurar Exibição de parâmetros elétricos: O usuário poderá configurar a exibição de parâmetros elétricos, exibindo ou não atributos como tensão ou potência.
- Executar Algoritmos de análise: O usuário poderá executar algoritmos para análise da rede elétrica. Esses algoritmos deverão ser implementados pelos engenheiros responsáveis.
- Visualizar Atributos Elétricos: O usuário poderá clicar no ícone de um equipamento ou trecho de rede para visualizar os atributos elétricos desse objeto (tensão, potência, etc.)

3.1.2 Diagrama de classes

O diagrama de classes é o diagrama mais comumente encontrado na modelagem de sistemas orientados a objetos. Um diagrama de classes mostra um conjunto de classes, interfaces, colaborações e seus relacionamentos para modelar a visão estática do projeto de um sistema (BOOCH; RUMBAUGH; JACOBSON, 1998, Capítulo 8).

Para modelar as classes de um sistema, é necessária uma análise inicial dos objetos do mundo real que essas classes irão representar. Em um alimentador de uma rede de distribuição primária de energia elétrica, os principais componentes são: subestação de distribuição, trechos de rede e postes (esses últimos podem conter equipamentos). Um desenho de como esses componentes são estruturados é apresentado na Figura 3.2.

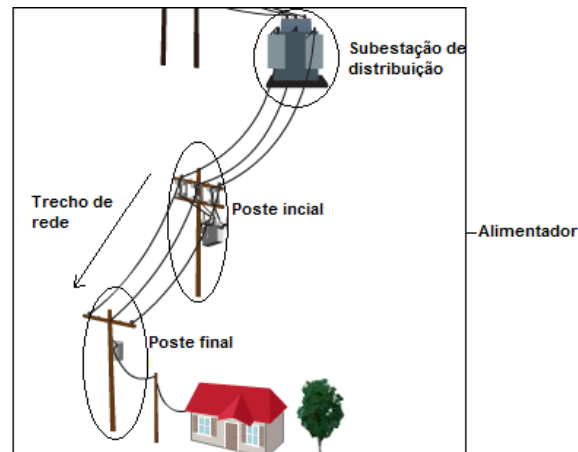


Figura 3.2 – Desenho dos principais componentes de um alimentador. Adaptado de (Rede inteligente, 2009)

A partir da subestação de distribuição, a energia é transportada através dos trechos de rede. Esses trechos são caracterizados por conter dois postes, (poste inicial e poste final), e cada poste pode conter um equipamento elétrico (no caso da Figura 3.2, o poste inicial contém um transformador de distribuição). Todo esse conjunto citado é chamado de alimentador.

Tendo sido observada essa estrutura de um alimentador, buscou-se modelar as classes que representariam essas entidades de maneira apropriada, e a Figura 3.3 mostra o diagrama das principais classes do sistema.

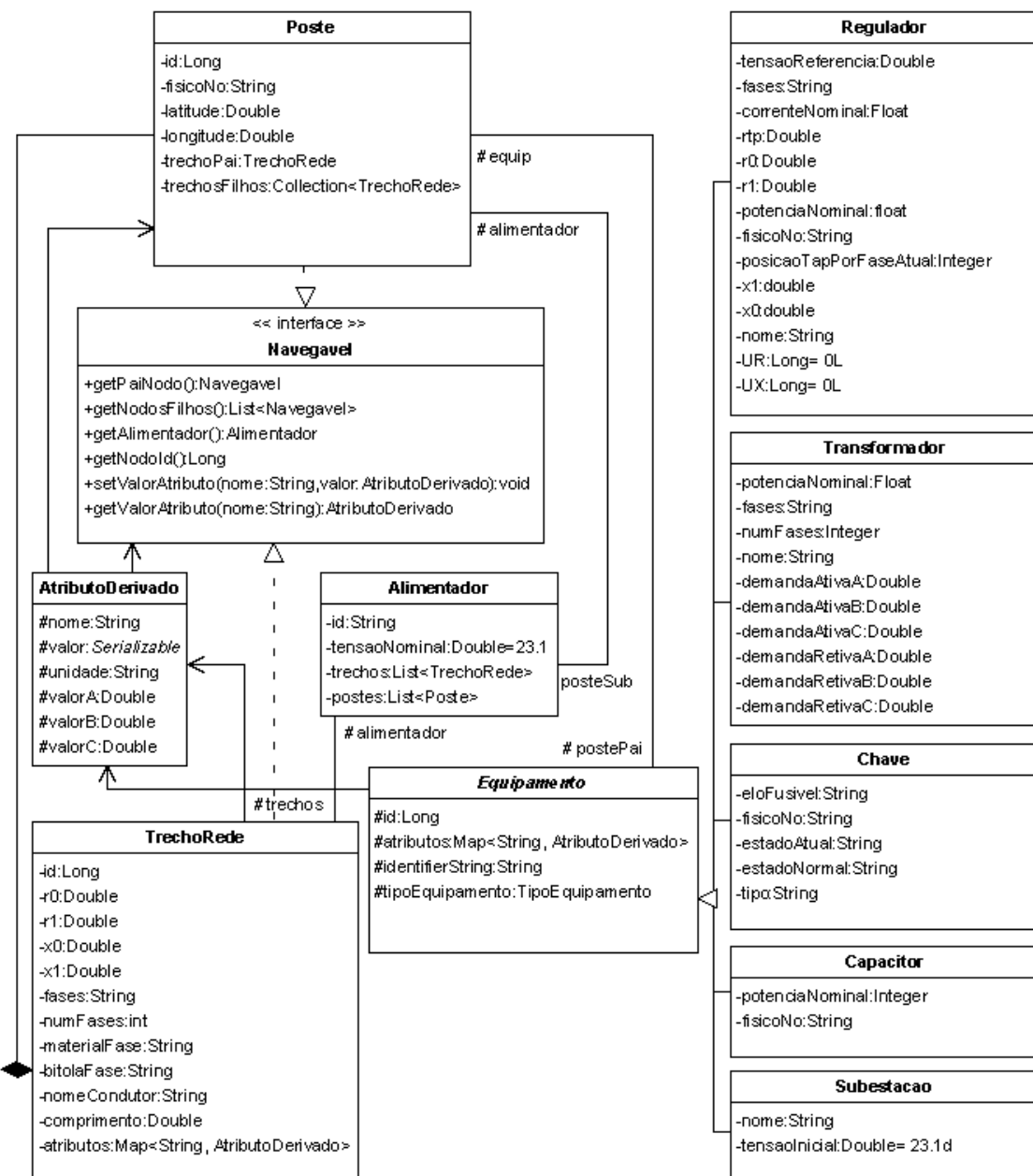


Figura 3.3 – Diagrama das principais classes do sistema

É interessante ressaltar algumas partes desse diagrama. A classe Alimentador, a qual representa a principal abstração da modelagem de todo o sistema, é composta basicamente de uma lista de trechos de redes (arcos) e outra de postes (nós), além de um poste para indexar o poste inicial (onde se encontra a subestação). Como as redes de distribuição brasileiras são tipicamente radiais ou no máximo possuem pouquíssimas malhas, foi possível representar a hierarquia de todo o alimentador através de uma árvore genérica, onde o poste onde se encontra a subestação do alimentador é considerado a raiz da árvore, e a partir dele é possível fazer a navegação entre postes e trechos de rede até os

postes finais do alimentador (nós folhas).

3.2 Implementação

Nesta seção são abordados os principais passos para implementação do sistema, passando pela modelagem do banco de dados até a estrutura para comunicação entre cliente-servidor disponibilizado pelo *GWT*.

3.2.1 Modelagem do banco de dados

Com o intuito de padronizar a representação de um alimentador de média tensão, foi proposta uma modelagem lógica para o banco de dados relacional, também objetivando tornar o sistema menos dependente da modelagem específica que cada concessionária de energia utiliza. Por exemplo, um transformador de distribuição pode ser modelado contendo somente potência nominal para uma concessionária, já outra pode incluir o número de fases em que o transformador opera. Essa pequena diferença pode vir a trazer problemas no momento do manuseio desses dados e para diminuir o impacto dessas diferentes modelagens, decidiu-se utilizar o mínimo de dados possíveis na modelagem do sistema. Dessa maneira pode-se observar o diagrama lógico da modelagem do banco de dados na Figura 3.4.

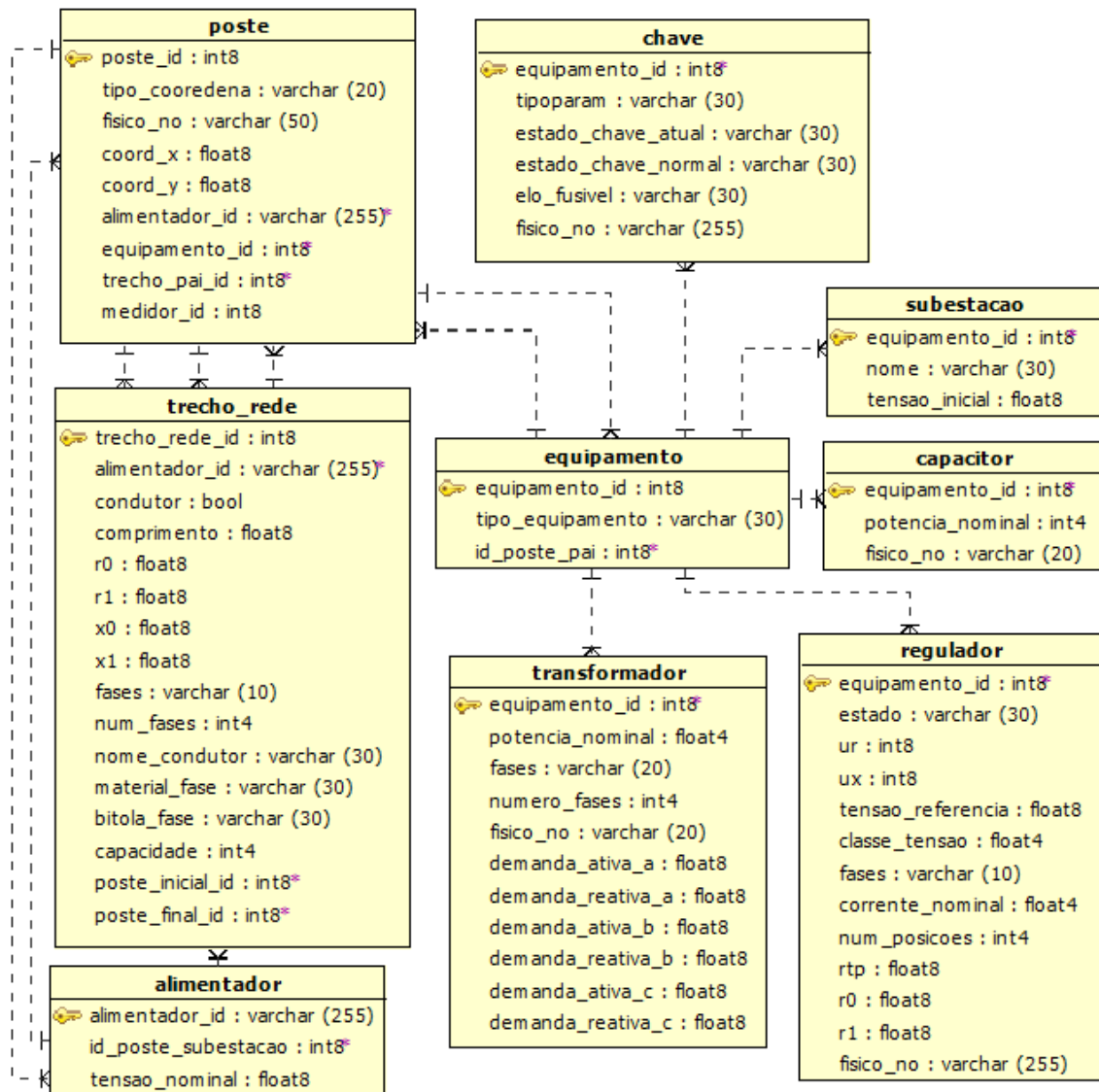


Figura 3.4 – Esquema lógico do banco de dados relacional do sistema

É interessante notar que as tabelas *chave*, *subestação*, *capacitor*, *regulador* e *transformador* tem como chave primária uma chave estrangeira da tabela *equipamento*. Isso faz com que entradas dessas tabelas não possam existir sem uma entrada na tabela *equipamento*. Como o poste pai é a única propriedade comum a todos os tipos de equipamento (no contexto do banco de dados), optou-se pelo o uso da implementação da herança como uma entidade por classe em conjunto ao uso do discriminador na entidade pai (*equipamento*).

3.2.2 Mapeamento objeto-relacional

O mapeamento objeto-relacional foi concebido a partir do diagrama do banco de dados. Optou-se pelo uso de arquivos *xml* para efetuar esse mapeamento, e a Figura 3.5 ilustra o arquivo de configuração do Hibernate (*hibernate.cfg.xml*) utilizado no sistema. Nesse arquivo foram configuradas as propriedades para utilização de cache do Hibernate ("cache.provider_class", "cache.use_query_cache" e "cache.use_second_level_cache"), as propriedades de configuração do *driver* JDBC para acesso ao banco ("dialect", "connection.driver_class", "connection.url", "connection.username" e "connection.password") e por fim a correlação entre as classes mapeadas e seus arquivos de mapeamento (tags *xml* <mapping>).

```

1 <hibernate-configuration>
2   <session-factory name="sessionFactory">
3     <property name="cache.provider_class">
4       org.hibernate.cache.EhCacheProvider
5     </property>
6     <property name="cache.use_query_cache">true</property>
7     <property name="cache.use_second_level_cache">true</property>
8     <property name="format_sql">true</property>
9     <property name="hbm2ddl.auto">update</property>
10    <property name="show_sql">>false</property>
11    <property name="dialect">org.hibernate.dialect.PostgreSQLDialect</property>
12    <property name="connection.driver_class">org.postgresql.Driver</property>
13    <property name="connection.url">jdbc:postgresql://localhost/asd</property>
14    <property name="connection.username">asd</property>
15    <property name="connection.password">asd</property>
16    <property name="hibernate.dialect">
17      org.hibernate.dialect.PostgreSQLDialect
18    </property>
19    <mapping resource="br/com/server/model/hbm/Alimentador.hbm.xml"
20      class="br.com.server.model.Alimentador"/>
21    <mapping resource="br/server/model/hbm/Poste.hbm.xml"
22      class="br.com.server.model.Poste"/>
23    <mapping resource="br/server/model/hbm/TrechoRede.hbm.xml"
24      class="br.com.server.model.TrechoRede"/>
25    <mapping resource="br/server/model/hbm/Equipamento.hbm.xml"
26      class="br.com.server.model.Equipamento"/>
27    <mapping resource="br/server/model/hbm/Usuario.hbm.xml"
28      class="br.com.server.model.Usuario"/>
29  </session-factory>
30 </hibernate-configuration>

```

Figura 3.5 – Parte do arquivo XML de configuração do Hibernate (*hibernate.cfg.xml*)

Com o mapeamento objeto-relacional e configuração do Hibernate realizados, foi implementado um DAO (*Data Access Object*) para as operações de persistência com o banco de dados. A Figura 3.6 ilustra os principais métodos que essa classe implementa para prover persistência às entidades mapeadas pelo Hibernate, e seus métodos mais im-

portantes são:

```

1 public class AsdDAO <T, PK extends Serializable> {
2     private SessionFactory factory;
3     public AsdDAO(SessionFactory factory) { this.factory = factory; }
4     public Session getSession() {
5         return SessionFactoryUtils.getSession(factory, true);
6     }
7     public void delete(T entity) {
8         this.getSession().delete(entity);
9     }
10    public Object findById(PK id, Class objectClass) {
11        return this.getSession().get(objectClass, id);
12    }
13    public T save(T entity) {
14        this.getSession().saveOrUpdate(entity);
15        return entity;
16    }
17    public Set<T> findAll(Class objectClass) {
18        return new HashSet(findByCriteria(objectClass));
19    }
20    public List<T> findByCriteria(Class objectClass,
21                                Criterion... criterion) {
22        Criteria crit = this.getSession().createCriteria(objectClass);
23        if (criterion != null) {
24            for (Criterion c : criterion) {
25                crit.add(c);
26            }
27        }
28        return crit.list();
29    }
30    // (...)
31 }

```

Figura 3.6 – Classe AsdDAO e parte de seus métodos persistentes

- *getSession*: Utiliza a classe *SessionFactoryUtils* fornecida pelo *framework* Spring para facilitar a criação e manipulação de sessões com o banco de dados.
- *delete*: Remove um objeto do banco de dados.
- *findById*: Retorna um objeto de acordo com o identificador (chave primária) do mesmo no banco de dados.
- *findAll*: Retorna todos os objetos de uma determinada classe que o banco de dados contém.
- *findByCriteria*: Retorna todos os objetos que atendem a um determinado critério de uma determinada classe que o banco de dados contém. A API *Criteria* do Hibernate procura abstrair as cláusulas *where* da linguagem SQL.

Tendo sido observado o grande volume de trechos de rede e postes referentes ao alimentador de média tensão, em conjunto a uma demasiada demora na leitura das informações a partir do banco de dados, utilizou-se a opção *batch-size* nos arquivos de mapeamento dos equipamentos, postes e trechos de redes, visto que essas entidades são as que possuem maior número de tuplas. Essa opção força o Hibernate a agrupar um conjunto de entidades em um mesmo *select*, com o intuito de tornar a recuperação delas a partir do banco mais rápida. Dessa forma pode-se verificar uma melhora no desempenho do carregamento de todo o alimentador.

3.2.3 Estrutura para execução de algoritmos de análise em redes elétricas

Com a infraestrutura física dos dados pronta, iniciou-se a construção de uma estrutura de classes para execução de algoritmos de análise em redes elétricas. Nas próximas subseções será detalhada a estrutura de classes para execução de algoritmos de análise baseada na modelagem hierárquica de postes e trechos de rede proposta anteriormente junto ao diagrama de classes, na subseção *Diagrama de classes*.

3.2.3.1 Classe *AlgoritmoAbstrato*

A classe *AlgoritmoAbstrato* é uma classe abstrata base para padronizar os métodos que devem ser implementados em um novo algoritmo para análise a ser adicionado. A Figura 3.7 ilustra os métodos e atributos dessa classe abstrata.

```

1 public abstract class AlgoritmoAbstrato {
2
3     protected ConjuntoParametros conjuntoParametros =
4         new ConjuntoParametros();
5     protected abstract Serializable execute(Alimentador alimentador,
6                                             Serializable params);
7     protected abstract AlgoritmoInfo getAlgoritmoInfo();
8     protected ConjuntoParametros getConjuntoParametrosPadrao() {
9         return new ConjuntoParametros();
10    }
11    protected void setConjuntoParametros(
12        ConjuntoParametros conjuntoParametros) {
13        this.conjuntoParametros = conjuntoParametros;
14    }
15    protected Serializable getConjuntoParametros() {
16        return conjuntoParametros;
17    }
18 }

```

Figura 3.7 – Classe *AlgoritmoAbstrato* com seus métodos e atributos

A classe *ConjuntoParametros* nada mais é que uma abstração para um *Map<String, Parametro>* onde a classe *Parametro* define um valor, um tipo (*Boolean*, *Double* ou *String*) e um *label* (nome). Esses parâmetros são necessários, pois cada algoritmo elétrico pode possuir uma passagem inicial de parâmetros que são necessários para sua correta execução. O método *execute* é o método principal do algoritmo elétrico, sendo responsável pela execução dos passos para realização do algoritmo elétrico propriamente dito. Esse método é abstrato e novos algoritmos devem obrigatoriamente implementá-lo. Outro método abstrato é método *getAlgoritmoInfo()*, responsável por prover um objeto contendo informações sobre o algoritmo (nome, descrição, entre outros).

3.2.3.2 Interface *Navegavel*

Na interface *Navegavel* são assinados os métodos para navegação uniforme da árvore do alimentador, através dos postes e trechos de rede (também chamadas de nós). A Figura 3.8 mostra os métodos que essa interface define.

```

1 public interface Navegavel {
2
3     public Navegavel getPaiNode();
4     public List<Navegavel> getNodosFilhos();
5     public Long getNodeid();
6     public void setValorAtributo(String nome, AtributoDerivado valor);
7     public AtributoDerivado getValorAtributo(String nome);
8     public Alimentador getAlimentador();
9
10 }
```

Figura 3.8 – Interface *Navegavel*

O método *getPaiNode()* deve retornar o nó que está acima na hierarquia da árvore do alimentador e o método *getNodosFilhos()* os nós que estão abaixo. Como essa hierarquia é representada por uma árvore genérica, um nó é caracterizado por ter um pai e diversos filhos. Os métodos *setValorAtributo(String nome, AtributoDerivado valor)* e *getValorAtributo(String nome)* são interfaces de alteração e acesso aos atributos elétricas presentes no nó, respectivamente. Por fim o método *getAlimentador()* permite o acesso direto ao alimentador de média tensão.

3.2.3.3 A classe *Navegador* e as interfaces *AcaoPreFixada* e *AcaoPosFixada*

Com o intuito de padronizar também as operações pré-fixadas (antes da passagem no nó) e pós-fixadas (depois da passagem do nó), foram definidas as interfaces *AcaoPreFixada* e *AcaoPosFixada* e as definições de seus métodos são mostradas na Figura 3.9, respectivamente.

```

1 public interface AcaoPreFixada {
2
3     Object executePre(Navegavel n, Object params);
4
5 }

```

```

1 public interface AcaoPosFixada {
2
3     Object executePos(Navegavel n, Object params);
4     Object operaResultados(Object params, Object resultadoParcial,
5                             Object resultadoRamo);
6 }

```

Figura 3.9 – Interfaces *AcaoPrefixada* e *AcaoPosFixada*, respectivamente

Na interface *AcaoPreFixada* o único método definido é o *executePre(Navegavel n, Object params)*. Esse método representa a ação (conjunto de operações) que serão realizadas antes do percurso completo da árvore e recebe como parâmetro o nó (*Navegavel n*) em que a ação será realizada, bem como o(s) parâmetro(s) necessários para a mesma (*Object params*). A interface *AcaoPosFixada* define o método *executePos(Navegavel n, Object params)* que é praticamente igual ao método anteriormente citado, com a ressalva que a ação é executada após o percurso da árvore do alimentador. Essa última interface ainda define o método *operaResultados(Object params, Object resultadoParcial, Object resultadoRamo)*, responsável por fazer a operação entre o resultado parcial (resultado das ações pós-fixadas do nó folha até o nó atual) e o resultado de um ramo (resultado de um dos irmãos do nó atual).

Essas classes acima citadas não teriam muita utilidade sozinhas, e na realidade são usadas pela classe *Navegador*. Esta classe fornece um método para o percurso da árvore do alimentador executando ações pré-fixadas e pós-fixadas de maneira uniforme e automática. A Figura 3.10 mostra a implementação desse e outro método privado desta classe.

```

1 public class Navegador {
2
3     public static Object percorreArvore(
4         Navegavel source, Object params,
5         AcaoPreFixada pre, AcaoPosFixada pos) {
6         Collection<Navegavel> navegou = new LinkedList<Navegavel>();
7         params = pre.executePre(source, params);
8         Object result = null;
9         for (Navegavel n : getDestinos(source, navegou)) {
10             Object rRamo = navega(n, params, pre, pos, new LinkedList<Navegavel>());
11             result = pos.operaResultados(params, result, rRamo);
12         }
13         return pos.executePos(source, result);
14     }
15
16     private static Object navega(
17         Navegavel nodo, Object params, AcaoPreFixada pre,
18         AcaoPosFixada pos, Collection<Navegavel> navegou) {
19         params = pre.executePre(nodo, params);
20         navegou.add(nodo);
21         Object result = null;
22         for (Navegavel n : getDestinos(nodo, navegou)) {
23             Object rRamo = navega(n, params, pre, pos, navegou);
24             result = pos.operaResultados(params, result, rRamo);
25         }
26         return pos.executePos(nodo, result);
27     }
28     // (...)
29 }

```

Figura 3.10 – Parte principal da classe *Navegador*

É através do método *percorrerArvore* que ocorre o percurso da árvore do alimentador, varrendo todos os filhos a partir do nó passado como parâmetro, executando as ações pré e pós fixadas também passadas como parâmetros, e em seguida chama o método *navega*. Este último método tem a mesma função do método anterior, porém continua a navegação recursivamente até não serem mais encontrados nós filhos.

3.2.3.4 Exemplo de implementação de algoritmo

Com essa infraestrutura proposta anteriormente, se torna muito mais fácil a inclusão de novos algoritmos para análise da rede. Essa subseção pretende ilustrar um exemplo prático do algoritmo para calcular o somatório das correntes ao longo do alimentador, o qual é bastante utilizado em diversos outros algoritmos (fluxo de potência, estimação de estado, entre outros). A Figura 3.11 mostra a ideia geral desse algoritmo.

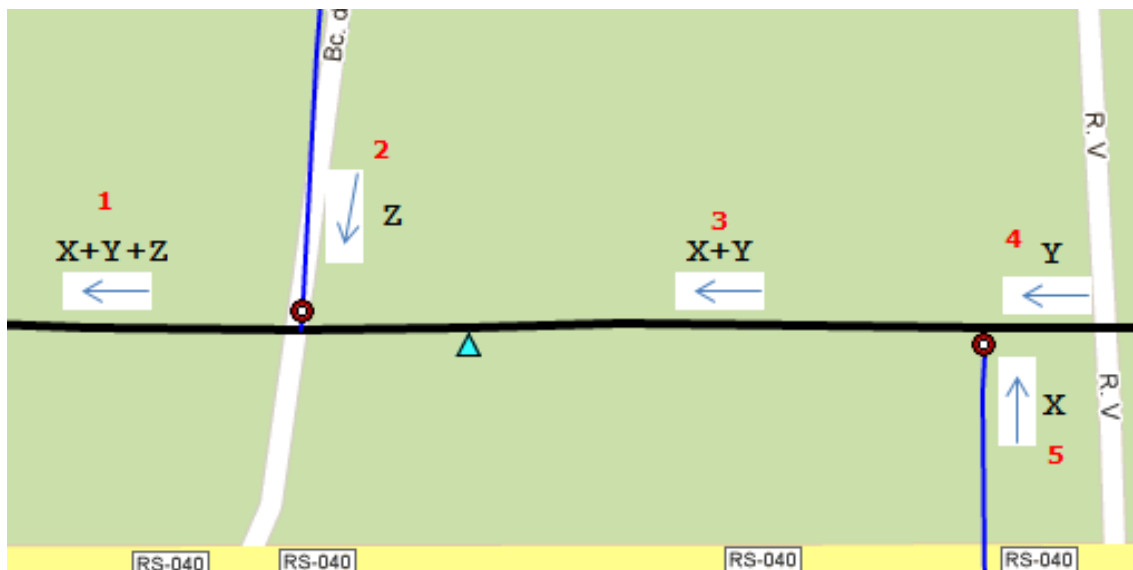


Figura 3.11 – Ilustração de como o algoritmo para somar as correntes funciona

Inicialmente o algoritmo deve percorrer toda a árvore do alimentador no sentido de navegação do poste inicial do alimentador até os postes finais (pré-fixado, contrário ao apresentado pelas setas da Figura 3.11, no sentido de 1 para 5), sem fazer nenhuma ação em específico. Após o percurso de toda a extensão do alimentador, o percurso passa a ser dos nós finais até o inicial (de maneira pós-fixada, no sentido das setas apresentadas na Figura 3.11), onde as correntes (X, Y e Z) no ponto de intersecção entre os trechos (poste) são somadas à corrente acumulada dos outros ramos.

Dessa forma, para implementar esse algoritmo através da estrutura de classes proposta, o primeiro passo é criar uma classe que implemente a interface *AcaoPosFixada*, visto que as operações necessárias para realizar o somatório das correntes são realizadas na volta do percurso normal do alimentador (pós-fixadamente). A Figura 3.12 mostra a classe *CalculaSomaCorrentesAcaoPos*, responsável por realizar a soma das correntes acumuladas até então com a corrente do nó em questão.

```

public class CalculaSomaCorrentesAcaoPos implements AcaoPosFixada {

    public static final String NOME_ATTR = "Corrente";

    public Object executePos(Navegavel nav, Object params) {
        AtributoDerivado attrCorrente = (AtributoDerivado) params;
        if (attrCorrente == null) {
            attrCorrente = new AtributoDerivado(NOME_ATTR, new Double[]{0d, 0d, 0d}, "A");
        }
        if (nav.getValorAtributo(NOME_ATTR) != null) {
            AtributoDerivado attrNav = nav.getValorAtributo(NOME_ATTR);
            attrCorrente.setValorA(attrCorrente.getValorA() + attrNav.getValorA());
            attrCorrente.setValorB(attrCorrente.getValorB() + attrNav.getValorB());
            attrCorrente.setValorC(attrCorrente.getValorA() + attrNav.getValorC());
        }
        return attrCorrente;
    }

    public Object operaResultados(Object params, Object resultadoParcial, Object resultadoRamo) {
        if (resultadoParcial == null && resultadoRamo == null) {
            return new AtributoDerivado(NOME_ATTR, new Double[]{0d, 0d, 0d}, "A");
        }
        if (resultadoParcial == null) {
            return resultadoRamo;
        }
        if (resultadoRamo == null) {
            return resultadoParcial;
        }
        AtributoDerivado res = new AtributoDerivado(NOME_ATTR, new Double[]{0d, 0d, 0d}, "A");
        AtributoDerivado r1 = (AtributoDerivado) resultadoParcial;
        AtributoDerivado r2 = (AtributoDerivado) resultadoRamo;
        res.setValorA(r1.getValorA() + r2.getValorA());
        res.setValorB(r1.getValorB() + r2.getValorB());
        res.setValorC(r1.getValorC() + r2.getValorC());
        return res;
    }
}

```

Figura 3.12 – Classe *CalculaSomaCorrentesAcaoPos*, responsável por somar as correntes acumuladas até o nó atual e retornar o somatório ao seu nó pai

o método `executePos()` é chamado após a navegação no nó em questão. Dessa maneira ele retorna para o nó pai, o valor das correntes acumuladas até então (*param*) somadas ao valor da corrente dele (*nav.getValorAtributo(NOME_ATTR)*). Já o método *operaResultados* é responsável por acumular o valor da soma de cada um dos seus filhos (*resultadoRamo*) ao somatório obtido até o nó atual (*resultadoParcial*).

Com a classe da ação pós-fixada implementada, deve-se criar então uma classe que derive da classe abstrata *AlgoritmoAbstrato*, implementando no mínimo os métodos abstratos desta última, a fim de incluir esse novo algoritmo de maneira automática no menu da aplicação. A Figura 3.13 ilustra a classe *CalcularSomaCorrentesAlgoritmo*, herdando a classe abstrata e implementando os métodos obrigatórios.

```

1 public class CalcularSomaCorrentesAlgoritmo extends AlgoritmoAbstrato {
2
3     @Override
4     protected Serializable execute(Alimentador alimentador, Serializable params) {
5         return (Serializable) Navegador.percorreArvore(
6             alimentador.getPosteSubEstacao(),
7             new CalculaSomaCorrentes());
8     }
9
10    @Override
11    protected AlgoritmoInfo getAlgoritmoInfo() {
12        return new AlgoritmoInfo("Calcular soma das correntes",
13            "Esse algoritmo calcula a soma das correntes de todos os nós do " +
14            "alimentador");
15    }
16 }

```

Figura 3.13 – Classe CalcularSomaCorrentesAlgoritmo, responsável por realizar o algoritmo de cálculo do somatório das correntes

O método *execute* é a interface padrão para realização da execução propriamente dita do algoritmo e o método *getAlgoritmosInfo()* é necessário para identificação do algoritmo, a fim de identificá-lo no menu desses algoritmos.

3.2.4 Comunicação cliente-servidor com GWT

O GWT implementa a comunicação entre cliente-servidor de maneira assíncrona e elegante, através do mecanismo ARPC ou chamada de procedimento remoto assíncrono. Para tal mecanismo funcionar corretamente, é necessário a implementação de classes nos ambientes cliente e servidor. As subseções seguintes abordarão esse processo.

3.2.4.1 Definição das classes DTO (*Data Transfer Objects*)

Um *DTO*, também chamado de um objeto de transferência ou um objeto de valor, é um objeto que abstrai um conjunto de valores, permitindo que clientes remotos possam solicitar e receber todo o conjunto desses valores com uma única chamada remota (PANTALEEV; ROUNTEV, 2007). Inserido no contexto de aplicações GWT que utilizam o Hibernate para mapeamento objeto-relacional, esses objetos se tornam uma das alternativas para viabilizar essa integração. Isso se dá ao fato de que o *Hibernate* modifica o código das classes mapeadas (trocando uma coleção *Set<Usuario>* por *org.hibernate.collection.PersistentSet<Usuario>*, por exemplo), tornando impossível a serialização de uma classe persistente através do mecanismo RPC do GWT.

Dessa maneira todas as classes mapeadas pelo *Hibernate* tiveram uma classe DTO relacionada implementada, nomeadas como o nome da classe mapeada com adição do sufixo *Cli*. Essas classes também devem implementar a interface *ClasseCliTransformadorInterface<T extends IsSerializable>*. Esta interface define um único método (*toClasseCliente()*) responsável pela transformação da classe persistente para a classe DTO (*Cli*) que a representa. A Figura 3.14 mostra parte da classe mapeada *Usuario* e parte de seu DTO (*UsuarioCli*), destacando também a implementação do método de transformação.

```
public class Usuario implements IsSerializable,
    ClasseCliTransformadorInterface<UsuarioCli> {

    private String nome;
    private String login;
    private String senha;

    @Override
    public UsuarioCli toClasseCliente() {
        UsuarioCli usuarioCli = new UsuarioCli(nome, login, senha);
        return usuarioCli;
    }
    // (...)
}

public class UsuarioCli implements IsSerializable {

    private String nome;
    private String login;
    private String senha;

    public UsuarioCli(String nome, String login, String senha) {
        this.nome = nome;
        this.login = login;
        this.senha = senha;
    }
    // (...)
}
```

Figura 3.14 – Parte da classe *Usuario* e parte de seu DTO *UsuarioCli*, destacando-se a implementação do método da interface *ClasseCliTransformadorInterface*

É interessante notar que o tipo da classe de retorno do método *toClasseCliente* é definido somente em tempo de compilação (utilizando a característica de programação genérica introduzida a partir do Java 1.5) e deve implementar a interface *IsSerializable* fornecida pelo kit GWT, tornando o *DTO* (*UsuarioCli*, no exemplo) serializável pelo mecanismo RPC do GWT.

3.2.4.2 Parte servidor do mecanismo RPC

Uma das classes mais importantes de um sistema mecanismo RPC do GWT é a classe que implementa os métodos que serão chamados remotamente através do cliente, também chamada de *classe de serviços*. Nesse trabalho a classe de serviço é chamada *AsdServiceImpl* e a Figura 3.15 exibe os atributos e alguns métodos que a mesma possui. Foi exibida a implementação somente do método *logInUsuario* pois os outros métodos possuem códigos de implementação muito extensos, tornando inviável anexá-los à figura.

```

1 public class AsdServiceImpl extends RemoteServiceServlet implements
2     AsdService {
3
4     private static ExcelServiceVia excelServiceVia;
5     private static AsdDAO dao;
6     private static Map<String, Alimentador> mapaAlimentadoresRodando =
7         new HashMap<>();
8
9     public UsuarioCli logInUsuario(String login, String senha) {
10         Usuario user = (Usuario) dao.findById(login, Usuario.class);
11         return user.toClasseCliente();
12     }
13     public Set<String> getAlimentadoresId() throws
14         UsuarioNaoLogadoException {...}
15     public AlimentadorCli getAlimentadorCli(String alimentadorId) throws
16         UsuarioNaoLogadoException {...}
17     public Alimentador getAlimentador(String alimentadorId) {...}
18     public List<String> getStringsInfoWindow(
19         String className, Serializable id) throws
20         UsuarioNaoLogadoException, AlimentadorNaoCarregadoException {...}
21     public Serializable executarAlgoritmo(
22         AlgoritmoInfoCli algoritmoInfoCli, Serializable params) throws
23         UsuarioNaoLogadoException, AlimentadorNaoCarregadoException {...}
24     public void persistirAlimentadores() {...}
25     // (...)
26 }

```

Figura 3.15 – Atributos e parte dos métodos da classe *AsdServiceImpl*

Essa classe deve obrigatoriamente herdar a classe *RemoteServiceServlet*, além de implementar a interface definida no cliente *AsdService*. Essa extensão, juntamente com a implementação da interface, são parte das etapas para permitir a chamada remota dos métodos dessa classe. O atributo *excelServiceVia* é um objeto da classe que implementa a importação dos dados para o banco é utilizado pelo método *persistirAlimentadores*. O atributo *dao* é o objeto da classe *AsdDAO* utilizado na grande maioria dos métodos dessa classe de serviços, como pode ser visto explicitamente no método *logInUsuario*.

3.2.4.3 Parte cliente do mecanismo RPC

Tendo as classes persistentes juntamente com suas classes *DTO* desenvolvidas e a classe de serviços já implementada, o GWT padroniza o acesso a esses métodos remotos através da definição de duas interfaces no cliente, uma com os métodos iguais (assinatura e tipo de retorno) à classe de serviços no servidor e outra com esses mesmos métodos com a ressalva da adição de um novo parâmetro (*AsyncCallback<>*) e retornando *void*. No sistema desenvolvido as interfaces síncrona e assíncrona são *AsdService* e *AsdServiceAsync*, respectivamente, e as estruturas das mesmas podem ser observadas na Figura 3.16.

```

1 @RemoteServiceRelativePath("AsdService")
2 public interface AsdService extends RemoteService {
3     public void persistirAlimentadores();
4     public AlimentadorCli getAlimentadorCli(String id);
5     public Set<String> getAlimentadoresId();
6     public List<String> getStringsInfoWindow(String className,
7                                             Serializable id);
8     public Serializable executarAlgoritmo(
9         AlgoritmoInfoCli algoritmoInfoCli,
10        Serializable params) throws Exception;
11    public List<NodoInfo> getParametrosEletricosNodosParaMostrar(
12        Constantes.TipoParametroEletrico params);
13    public List<AlgoritmoInfoCli> getAlgoritmosInfo();
14    public UsuarioCli loginUsuario(String login, String senha);
15 }
16 public interface AsdServiceAsync {
17     void getAlimentadorCli(String alimentador,
18                           AsyncCallback<AlimentadorCli> async);
19     void persistirAlimentadores(AsyncCallback<Void> async);
20     void getAlimentadoresId(AsyncCallback<Set<String>> async);
21     void getStringsInfoWindow(String className, Serializable id,
22                               AsyncCallback<List<String>> async);
23     void executarAlgoritmo(AlgoritmoInfoCli algoritmoInfoCli,
24                           Serializable params,
25                           AsyncCallback<Serializable> async);
26     void getAlgoritmosInfo(
27         AsyncCallback<List<AlgoritmoInfoCli>> async);
28     void getParametrosEletricosNodosParaMostrar(
29         Constantes.TipoParametroEletrico params,
30         AsyncCallback<List<NodoInfo>> async);
31     void loginUsuario(String login, String senha,
32                       AsyncCallback<UsuarioCli> async);
33 }

```

Figura 3.16 – Interfaces *AsdService* e *AsdServiceAsync*, respectivamente

A interface *AsdService* define os métodos que poderão ser invocados remotamente, e a interface *AsdServiceAsync* deve definir esses mesmos métodos, porém re-

tornando *void* e com um parâmetro a mais. Na prática a chamada remota ocorre através dessa segunda interface, utilizando também a interface *AsyncCallback* fornecida pelo GWT. Essa última interface permite que a execução de um método remoto seja assíncrona, ou seja, não bloqueie a continuação da execução do servidor. A Figura 3.17 ilustra uma chamada remota fazendo uso das interfaces *AsdServiceAsync* e *AsyncCallback*

```
//(...)
AsdServiceAsync asdAsync = (AsdServiceAsync) GWT.create(AsdService.class);
asdAsync.getAlimentadoresId(new AsyncCallback<Set<String>>() {
    public void onFailure(Throwable throwable) {
        Window.alert("Lista de alimentadores não pode ser carregada!");
        throwable.printStackTrace();
        Utils.setCarregando(false);
    }

    public void onSuccess(Set<String> alimentadoresIds) {
        for (String s : alimentadoresIds) {
            ComboBoxAlimentadores.getInstance().addItem(s, s);
        }
        mostrarDialogEscolherAlimentadores();
        Utils.setCarregando(false);
        carregouAlimentadores = true;
    }
});
//(...)
```

Figura 3.17 – Trecho de código da chamada remota do método *getAlimentadoresIds*

Esse exemplo é típico de todas as chamadas remotas. o método remoto é invocado a partir da interface assíncrona que define em tempo de compilação o tipo de retorno do método remoto. Caso ocorra algum erro na chamada remota, o método *onFailure* da interface *AsyncCallback* será executado. Se tudo ocorrer normalmente, o método *onSuccess* é executado contendo como parâmetro o retorno do método executado no servidor.

3.3 Conclusão do capítulo

No atual capítulo esse trabalho apresentou uma visão geral dos passos do desenvolvimento do sistema proposto. Nessa etapa foram especificados o diagrama de casos de uso (descrevendo as funcionalidades do sistema) e diagrama de classes (especificando as classes mais importantes do sistema). Junto ao diagrama de classes, foi ilustrado como um alimentador numa rede de distribuição real é estruturado, a fim de tornar mais compreensível a modelagem de classes proposta. Após isso, inciou-se a descrição mais detalhada de como o sistema foi implementado, descrevendo as etapas: (1) modelagem física do

banco de dados, destacando as tabelas e atributos das mesmas, (2) realização do mapeamento objeto-relacional a partir da modelagem do banco proposta, incluindo a descrição do DAO (*Data Access Object*) utilizado para acesso e atualização dos dados persistentes da aplicação, (3) descrição da plataforma, para inclusão de maneira sucinta e intuitiva, novos algoritmos para análise em redes elétricas, utilizando as classes *AlgoritmoAbstrato*, *Navegador* e as interfaces *Navegavel*, *AcaoPreFixada* e *AcaoPosFixada*, bem como a exemplificação de um algoritmo teste proposto para ilustrar o uso dessa plataforma, (4) Descrição da arquitetura da comunicação entre cliente e servidor através do mecanismo ARPC (*Asynchronous Remote Procedure Call*) do GWT, desde a justificação do uso de classes DTO (*Data Transfer Objects*) para viabilizar o uso do framework *Hibernate* em conjunto a esse mecanismo ARPC, até a descrição das interfaces definidas no lado do cliente e a implementação na classe de serviços no lado servidor.

4 RESULTADOS

Esse capítulo apresenta os resultados obtidos a partir da realização das etapas propostas para o desenvolvimento do sistema, destacando suas principais interfaces de usuário para exibições das informações elétricas do alimentador de média tensão em um ambiente georreferenciado.

4.1 Telas

O sistema teve duas telas principais como resultado. A primeira é a tela de login, responsável por fazer a autenticação do usuário a utilizar o sistema. Em seguida é carregada a segunda tela, responsável pela exibição dos equipamentos e atributos do alimentador propriamente dito. As subseções a seguir descreverão essas duas telas em maiores detalhes.

4.1.1 Tela de *login*

A primeira tela que o sistema exibe ao acessar o mesmo é a tela para *login*. Essa tela fornece uma interface gráfica para autenticação de um usuário e senha junto ao banco de dados do sistema e ilustração da mesma pode ser vista na Figura 4.1.

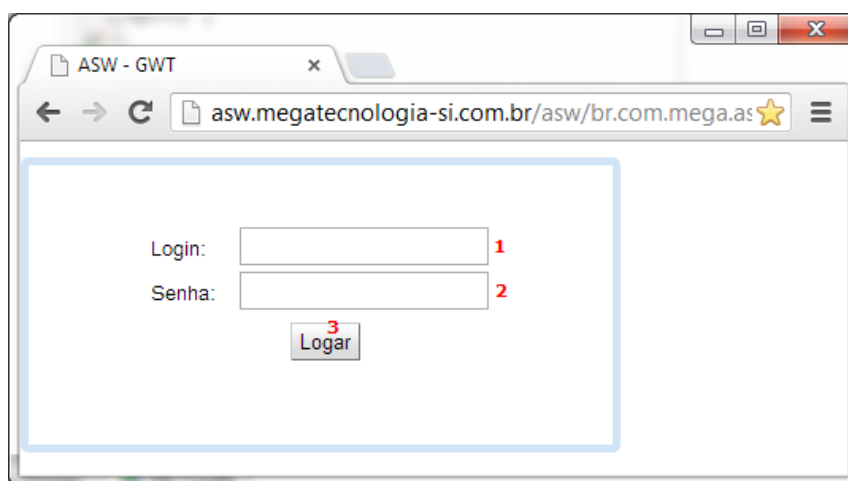


Figura 4.1 – Tela inicial de login do sistema

O campo *login* (1) deve ser preenchido com o login do usuário e o campo *senha* (2), com a senha pessoal desse usuário. Após esse preenchimento, pode-se clicar no botão *OK* (3) ou pressionar a tecla *ENTER* para iniciar a verificação das credenciais no

banco de dados. Em seguida ocorre o carregamento da tela principal do sistema, caso o login e senha sejam válidos.

4.1.2 Tela principal de exibição do alimentador

Após ter sido realizado o login é possível carregar um alimentador presente no banco de dados. Para isso é necessário acessar o menu *Arquivo* (1) e clicar no submenu *Carregar Alimentador* e então um *Combobox* (2) é exibido para seleção do alimentador a ser carregado. A Figura 4.2 mostra essa interface.

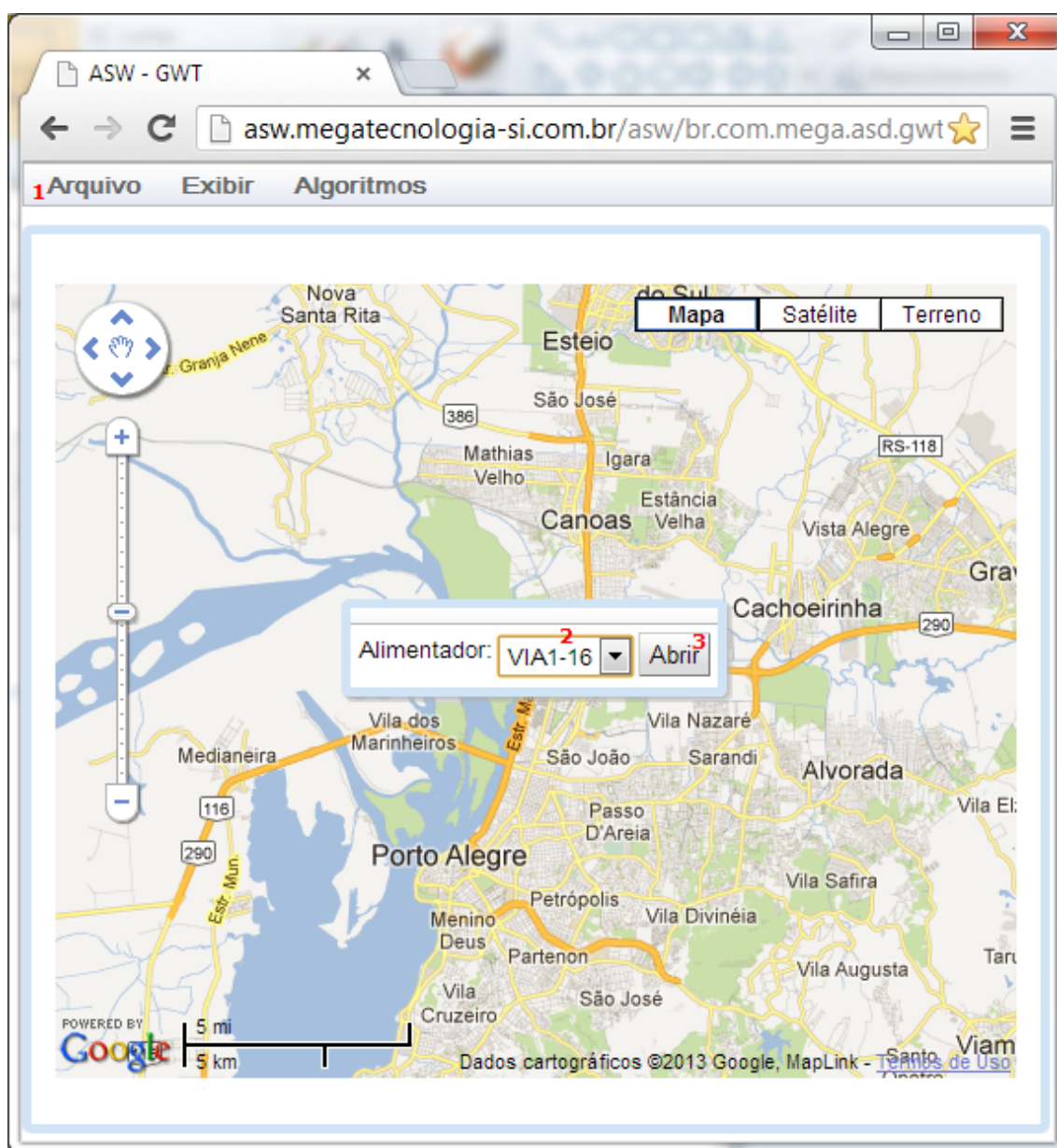


Figura 4.2 – Interface para o carregamento de um alimentador

Selecionado o alimentador a ser carregado, clica-se no botão *abrir* (3) para iniciar o carregamento do mesmo. Após esse carregamento é exibida a interface principal com trechos de rede e equipamentos do alimentador de média tensão num mapa georreferenciado. A Figura 4.3 demonstra essa tela principal.

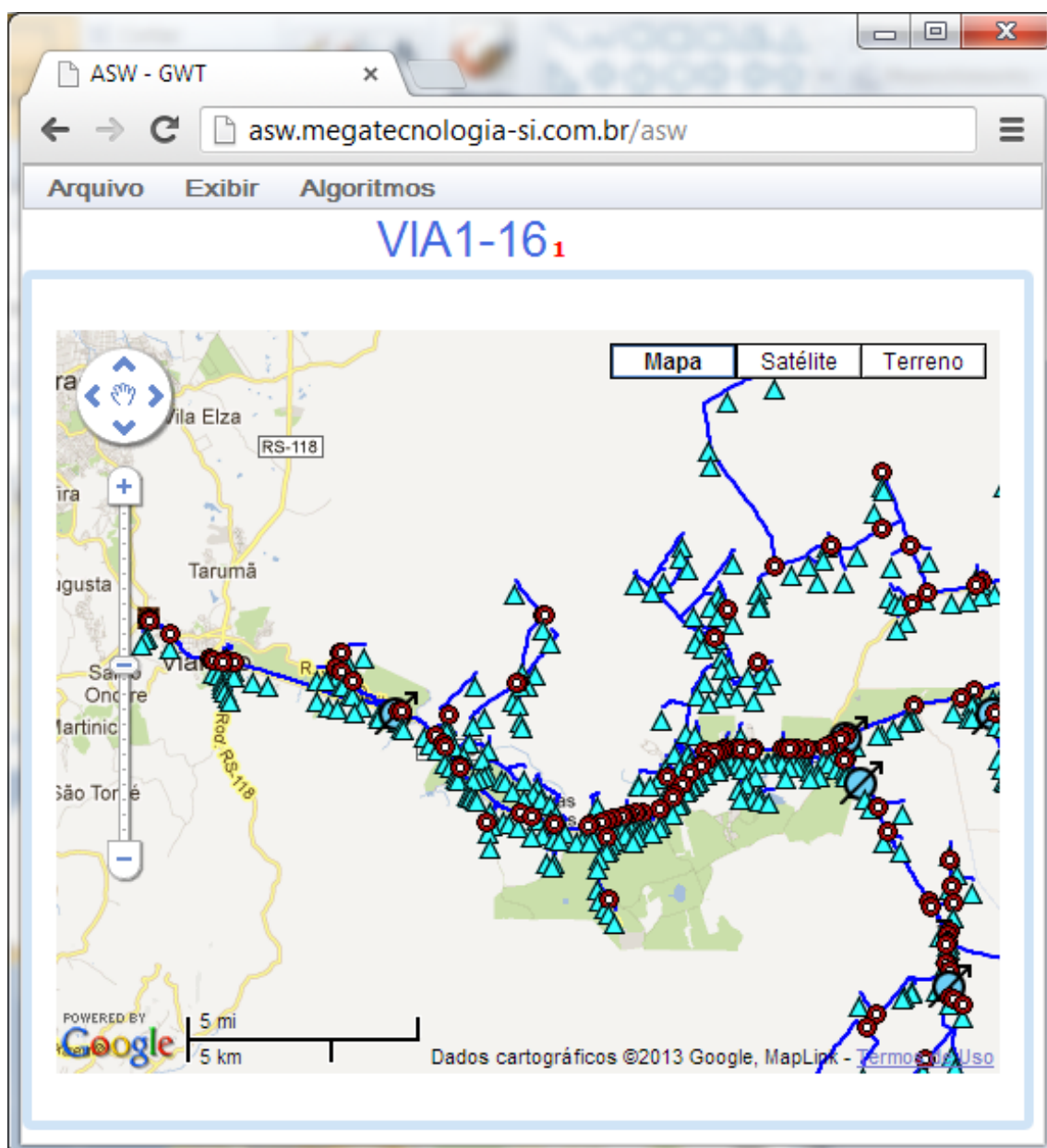


Figura 4.3 – Tela principal de exibição do alimentador de média tensão

O *label* (1) na parte superior da tela exibe o nome do alimentador carregado. As linhas representam trechos de rede, e cada ícone representa um equipamento elétrico diferente acoplado em um poste. Os equipamentos são *chaves*, *transformadores*, *capacitores* e *reguladores de tensão* e a legenda dos ícones que representam cada um deles é apresentada na Figura 4.4.

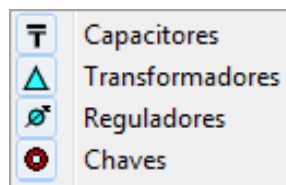


Figura 4.4 – legenda de ícones dos equipamentos

Ainda é possível configurar quais tipos de equipamentos deverão ser mostrados na interface georreferenciada, possibilitando ao usuário desde uma visão limpa da extensão do alimentador, exibindo somente seus trechos de rede, como uma visão completa do mesmo, exibindo também seus equipamentos. A Figura 4.5 ilustra essa possibilidade de personalização de exibição através do menu *Exibir*, marcando ou não os submenus que representam cada tipo de equipamento. Essa figura mostra um exemplo de menu configurado para exibição apenas dos transformadores de distribuição.

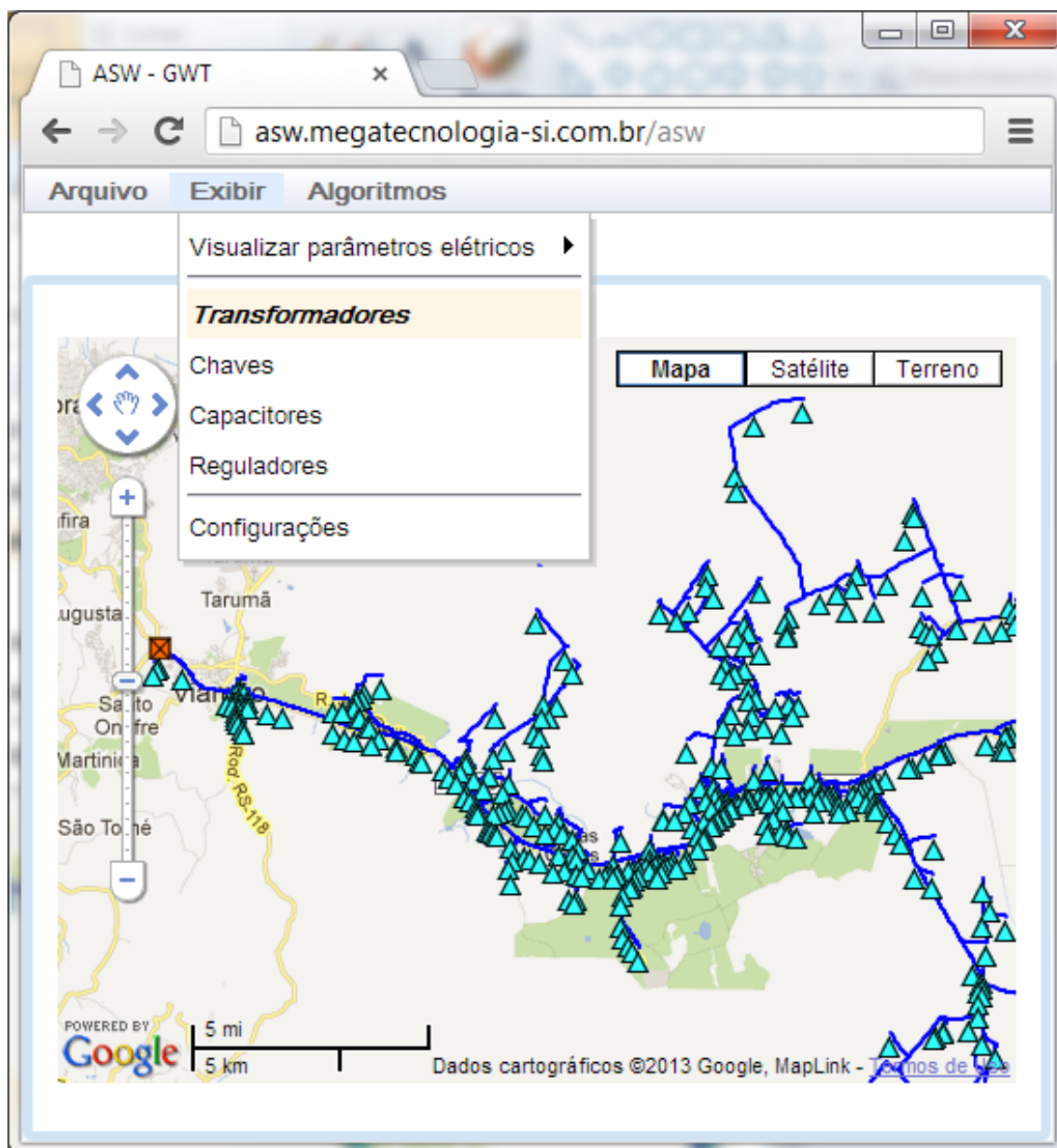


Figura 4.5 – Tela principal exibindo somente transformadores, e o menu de configuração de exibição dos equipamentos elétricos

Equipamentos que serão exibidos têm seus submenus em negrito e fundo de uma cor diferenciada. Dessa maneira é fácil a identificação de quais equipamentos estão configurados para serem mostrados, e quais não estão.

4.2 Outras interfaces

Nessa seção serão apresentadas as outras interfaces do sistema.

4.2.1 *Popup* de exibição dos atributos elétricos

Ao clicar-se nos ícones que representam os equipamentos ou em um determinado trecho de rede, um *popup* (1) é exibido com as informações daquele determinado trecho ou equipamento. A Figura 4.6 exibe o *popup* aberto com as propriedades de uma chave.

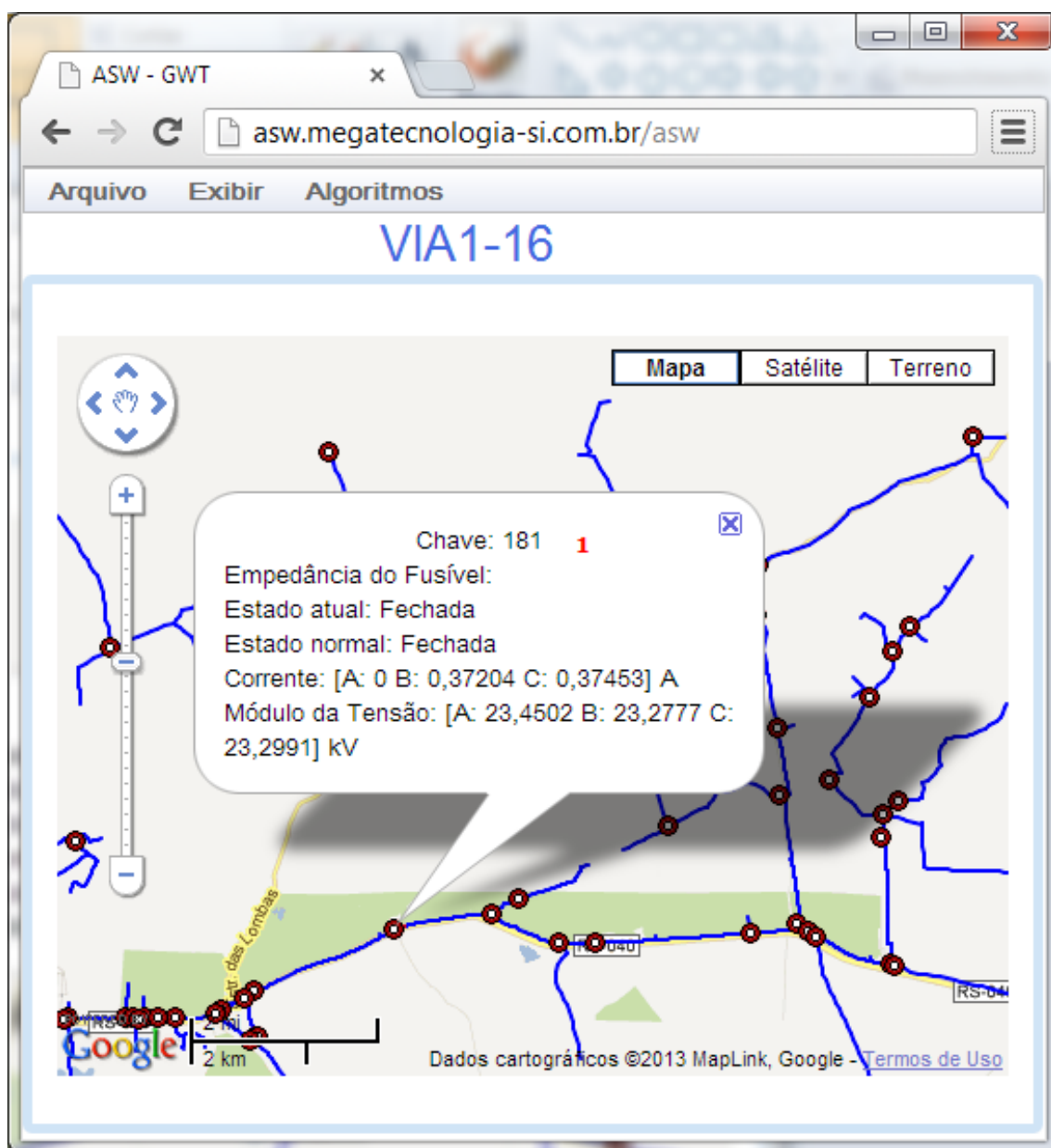


Figura 4.6 – *Popup* de exibição dos atributos elétricos de uma chave seccionadora

4.2.2 Diálogo para configuração de quais atributos elétricos serão exibidos

Existem atributos que são utilizados em todos os equipamentos e/ou trechos de rede, normalmente inseridos por um algoritmo de análise em redes elétricas. Exemplos

são os atributos *Corrente* e *Módulo da Tensão* da Figura 4.6 na subseção anterior. Para configuração de quais atributos devem ou não serem exibidos para cada tipo de equipamento, existe um diálogo desenvolvido para esse propósito, e sua estrutura pode ser verificada na Figura 4.7.

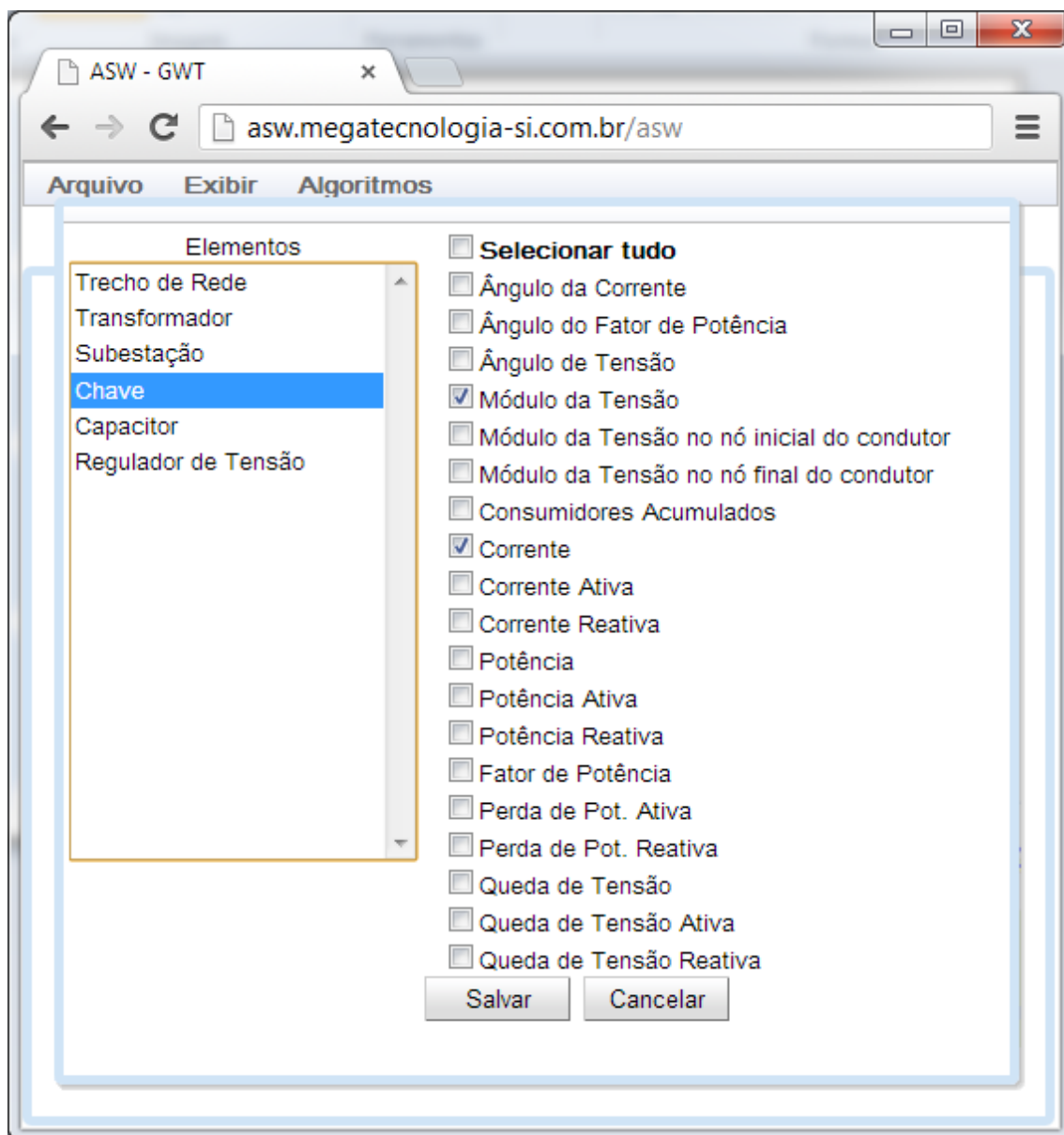


Figura 4.7 – Diálogo de configuração de quais atributos inseridos por algoritmos serão exibidos

Essa configuração pode ser feita independentemente para cada tipo de equipamento ou trecho de rede. A lista da esquerda contém os itens representando cada tipo de equipamento ou trecho de rede, e a lista à direita contém os atributos que são comumente utilizados por eles. Na 4.7 é exemplificada a configuração da exibição dos atributos *Corrente* e *Módulo da Tensão* inseridos por algoritmos nas chaves seccionadoras.

4.2.3 Execução de algoritmos de análise em redes elétricas

Os algoritmos elétricos para análise são disponibilizados através do menu *Algoritmos* (1) da tela principal. Os submenus (2) desse menu contém os algoritmos para análise implementados no servidor, e são montados em tempo de execução do cliente, junto ao carregamento da tela principal. A Figura 4.8 ilustra os menus para execução desses algoritmos bem como o resultado da execução do algoritmo *Caminho mais longo*.

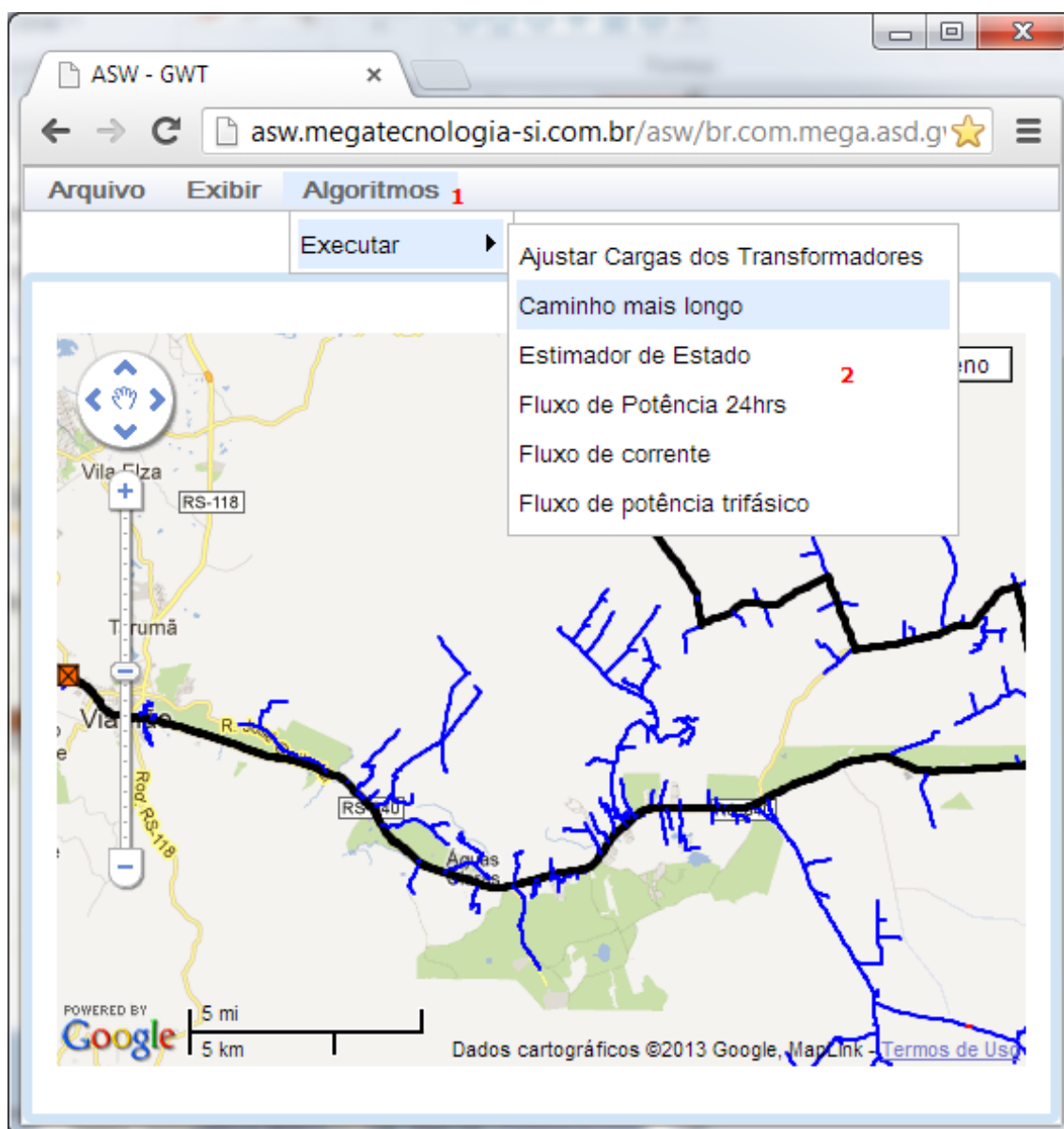


Figura 4.8 – Menu para execução dos algoritmos, exemplificando a execução do algoritmo *Caminho mais longo*

Esse algoritmo tem como resultado a pintura em preto dos trechos de rede que fazem parte do caminho raiz. O caminho raiz é definido como o maior caminho percorrido

(em número de navegações, não em extensão) do poste contendo a subestação até um poste que represente um nó final.

4.3 Conclusão do capítulo

Neste capítulo foram apresentados os resultados obtidos à partir do desenvolvimento do sistema proposto. Foram apresentadas as telas e outras interfaces do sistema, ilustrando a exibição do alimentador em interface georreferenciada, os menus para configuração de exibição dos equipamentos elétricos, o diálogo para personalização de quais atributos cada equipamento elétrico ou trecho de rede deve exibir, e por fim o menu para execução dos algoritmos de análise. Assim, pode-se ter uma visão de como os passos propostos e realizados no desenvolvimento, tiveram resultados satisfatórios em relação à aos objetivos desse trabalho.

5 CONCLUSÕES

Com o surgimento de diversas novas tecnologias computacionais, muitas áreas, principalmente comerciais, têm tirado proveito dessas tecnologias para melhorar a qualidade do serviço ou do processo como um todo. Tendo em vista que as redes de distribuição são muito importantes para o sistema elétrico, elas também são responsáveis pelas maiores perdas e o maior número de falhas. Objetivando uma melhora nestas em outras questões, elas deverão sofrer uma intensa modernização nos próximos anos através da instalação de novos equipamentos que permitem o monitoramento e operação de forma remota. Neste sentido, os sistemas de computação que permitem a análise e operação destas redes terão uma grande importância.

Com essa crescente importância dos sistemas de computação e comunicação nas redes de distribuição de energia elétrica, principalmente com as iniciativas de redes inteligentes (*Smart Grids*), esse trabalho apresentou as etapas para o desenvolvimento de um Sistema de Informações Georreferenciadas em três camadas, objetivando fornecer uma melhor análise de alimentadores em uma rede de distribuição de média tensão. Com o sistema foi possível verificar a topologia do alimentador de média tensão 16 da subestação VIA1 da Companhia Estadual de Distribuição de Energia Elétrica (CEEE-D), bem como verificar os dados e atributos dos trechos de rede e equipamentos do mesmo. Com a separação do sistema nas três camadas (dados, negócio e apresentação) em conjunto à utilização da tecnologia Java e *frameworks* como GWT (*Google Web Toolkit*) e *Google Maps API*, foi possível a construção de uma aplicação de internet com interface georreferenciada de um alimentador de média tensão. Além disso o sistema pode ser acessado através de qualquer computador que tenha acesso a internet e qualquer navegador web instalado, graças a capacidade de geração automática de códigos javascript pelo GWT para cada um deles.

Como trabalho futuro e tendo em vista a nova realidade de redes inteligentes, seria interessante o desenvolvimento de um módulo na camada de negócio para comunicação com os mais diversos tipos de equipamentos conectados ao longo do alimentador, a fim de monitorar as grandezas elétricas da cada um deles em tempo real. Essa comunicação poderia ser implementada diretamente com os equipamentos ou através de um sistema supervisor, neste último facilitando o acesso aos dados desses dispositivos.

REFERÊNCIAS

AMATO, F. **Apagão no Nordeste 'teve falha humana', diz diretor-geral da Aneel**. G1 (online). publicado 30/10/2012 - Disponível em <<http://g1.globo.com/economia/noticia/2012/10/apagao-no-nordeste-foi-provocado-por-falha-humana-diz-aneel.html>> - Acessado em novembro de 2012., 2012.

APACHE HTTP Server Project. Disponível em <<http://httpd.apache.org/>> - Acessado em dezembro de 2012.

ARTHUR, J.; AZADEGAN, S. Spring framework for rapid open source J2EE Web application development: a case study. In: SOFTWARE ENGINEERING, ARTIFICIAL INTELLIGENCE, NETWORKING AND PARALLEL/DISTRIBUTED COMPUTING, 2005 AND FIRST ACIS INTERNATIONAL WORKSHOP ON SELF-ASSEMBLING WIRELESS NETWORKS. SNPD/SAWN 2005. SIXTH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2005. p.90 – 95.

BANCILLON, D. **Apagão elétrico custou R\$ 45,2 bilhões aos brasileiros**. Jornal Correio Braziliense (online). publicado em 15/07/2009 - Disponível em <http://www.correiobraziliense.com.br/app/noticia/economia/2009/07/15/internas_economia,126861/index.shtml>, 2009.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The Unified Modeling Language User Guide**. 1.ed. [S.l.]: Addison Wesley, 1998.

CHAO, W. et al. Applications of GIS to Power Distribution Dispatching and analysis of technical questions. In: ELECTRICITY DISTRIBUTION (CICED), 2010 CHINA INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.1 –5.

CHAURASIA, P.; THAKUR, T. Consumer Indexing - A GIS based approach. In: POWER SYSTEMS, 2009. ICPS '09. INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2009. p.1 –5.

DAMIANO, A.; GATTO, G.; MARONGIU, I. Decision support system for biomass energy exploitation in smart grid planning. In: POWER ELECTRONICS ELECTRICAL DRIVES AUTOMATION AND MOTION (SPEEDAM), 2010 INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2010. p.1183 –1188.

FENG, Z. Research on the WebGIS Space Vector Data Based on the J2EE. In: MANAGEMENT AND SERVICE SCIENCE (MASS), 2010 INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.1 –4.

FU, P.; SUN, J. **Web GIS: principles and applications**. [S.l.]: ESRI Press, 2010.

GOOGLE. **Google Web Toolkit 2.2 Developer's Guide**. Disponível em <<https://developers.google.com/web-toolkit/doc/2.2/DevGuide>> - Acessado em dezembro de 2012: [s.n.], 2012.

GUNGOR, V. et al. Smart Grid Technologies: communication technologies and standards. **Industrial Informatics, IEEE Transactions on**, [S.l.], v.7, n.4, p.529 –539, nov. 2011.

HOCK-CHUAN, C. **How to Install Apache Tomcat 7 and Get Start with Java Servlet Programming**. Disponível em http://www3.ntu.edu.sg/home/ehchua/programming/howto/Tomcat_HowTo.html - Acessado em Janeiro de 2013.

HORSTMANN, C. **Big Java**. 2.ed. [S.l.]: John Wiley & Sons, 2005.

HU, X.; YU, Z. Research on distributed system based on middleware. In: SYSTEM OF SYSTEMS ENGINEERING (SOSE), 2012 7TH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2012. p.289–291.

JING, W.; FAN, R. The research of Hibernate cache technique and application of EhCache component. In: COMMUNICATION SOFTWARE AND NETWORKS (ICCSN), 2011 IEEE 3RD INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. p.160 –162.

JOHNSON, R. J2EE development frameworks. **Computer**, [S.l.], v.38, n.1, p.107 – 110, jan. 2005.

JOHNSON, R. et al. **Spring Framework Reference Documentation**. 3.2.0.ed. [S.l.: s.n.], 2012.

JOHNSON, R.; HOELLER, J. **Expert One-on-One J2EE Development Without EJB**. [S.l.]: John Wiley & Sons, 2004.

KASTURI, R. et al. Map data processing in geographic information systems. **Computer**, [S.l.], v.22, n.12, p.10 –21, dec. 1989.

KINGAND, G. et al. **Hibernate Reference Documentation**. 3.6.10.Final.ed. [S.l.: s.n.], 2004.

LIDUO, H.; YAN, C. Design and implementation of Web Content Management System by J2EE-based three-tier architecture: applying in maritime and shipping business. In: INFORMATION MANAGEMENT AND ENGINEERING (ICIME), 2010 THE 2ND IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.513 –517.

LIU, L. Integration design of Distribution SCADA and GIS. In: ELECTRICITY DISTRIBUTION (CICED), 2010 CHINA INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.1 –4.

MELIA, S. et al. A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA. In: WEB ENGINEERING, 2008. ICWE '08. EIGHTH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2008. p.13 –23.

MONTEIRO, C. et al. GIS spatial analysis applied to electric line routing optimization. **Power Delivery, IEEE Transactions on**, [S.l.], v.20, n.2, p.934 – 942, april 2005.

NEVES, C.; MIDIA, N. **Apagão**: bahia e outros estados ficam sem energia na noite desta quinta (25). Jornal correio24horas (online). publicado em 25/10/2012 - Disponível em <<http://www.correio24horas.com.br/noticias/detalhes/detalhes-1/artigo/apagao-moradores-de-diversos-bairros-de-salvador-ficam-sem-energia-na-noite-desta-quinta-feira-25/>> - Acessado em novembro de 2012., 2012.

PANAJOTOVIC, B.; JANKOVIC, M.; ODADZIC, B. ICT and smart grid. In: TELECOMMUNICATION IN MODERN SATELLITE CABLE AND BROADCASTING SERVICES (TELSIKS), 2011 10TH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. v.1, p.118 –121.

PANTALEEV, A.; ROUNTEV, A. Identifying Data Transfer Objects in EJB Applications. In: DYNAMIC ANALYSIS, 2007. WODA '07. FIFTH INTERNATIONAL WORKSHOP ON. **Anais...** [S.l.: s.n.], 2007. p.5.

PARIKH, P.; NIELSEN, T. Transforming traditional geographic information system to support smart distribution systems. In: POWER SYSTEMS CONFERENCE AND EXPOSITION, 2009. PSCE '09. IEEE/PES. **Anais...** [S.l.: s.n.], 2009. p.1 –4.

PAWLAK, P. et al. Social network application based on Google Web Toolkit. In: CAD SYSTEMS IN MICROELECTRONICS, 2009. CADSM 2009. 10TH INTERNATIONAL CONFERENCE - THE EXPERIENCE OF DESIGNING AND APPLICATION OF. **Anais...** [S.l.: s.n.], 2009. p.461 –464.

PEREIRA, O.; AGUIAR, R.; SANTOS, M. CRUD-DOM: a model for bridging the gap between the object-oriented and the relational paradigms. In: SOFTWARE ENGINEERING ADVANCES (ICSEA), 2010 FIFTH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.114 –122.

QINGWU, W.; JIAXUAN, Y. Architecture designation of distributed system based on Java and Delphi. In: POWER ELECTRONICS AND INTELLIGENT TRANSPORTATION SYSTEM (PEITS), 2009 2ND INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2009. v.3, p.228–230.

QIU, B.; GOOI, H. Web-based SCADA display systems (WSDS) for access via Internet. **Power Systems, IEEE Transactions on**, [S.l.], v.15, n.2, p.681 –686, may 2000.

RAO, M.; VARMA, B.; RADHAKRISHNA, C. Experiences on implementation of GIS based tools for analysis, planning and design of distribution systems. In: POWER AND ENERGY SOCIETY GENERAL MEETING - CONVERSION AND DELIVERY OF ELECTRICAL ENERGY IN THE 21ST CENTURY, 2008 IEEE. **Anais...** [S.l.: s.n.], 2008. p.1 –8.

REDE Inteligente: por que, como, quem, quando, onde? - publicado em 11/08/2009 (online) - Disponível em <http://www.redeinteligente.com/2009/08/11/rede-inteligente-por-que-como-quem-quando-onde/> - Acessado em fevereiro de 2013.

ROMERO, M. et al. Web based management system for power quality assessment and detection of critical zones. In: INNOVATIVE SMART GRID TECHNOLOGIES (ISGT EUROPE), 2011 2ND IEEE PES INTERNATIONAL CONFERENCE AND EXHIBITION ON. **Anais...** [S.l.: s.n.], 2011. p.1 –8.

ROZANSKI, N.; WOODS, E. **Software Systems Architecture**: working with stakeholders using viewpoints and perspectives. 2.ed. [S.l.]: Addison-Wesley, 2011. v.1.

SELVAM, C. et al. Advanced metering infrastructure for smart grid applications. In: RECENT TRENDS IN INFORMATION TECHNOLOGY (ICRTIT), 2012 INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2012. p.145 –150.

SENGUPTA, S.; BHATTACHARYA, S. Formalization of UML use case diagram-a Z notation based approach. In: COMPUTING INFORMATICS, 2006. ICOCI '06. INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2006. p.1 –6.

SHUAI, L.; XIE, G.; YANG, J. Characterization of HTTP behavior on access networks in Web 2.0. In: TELECOMMUNICATIONS, 2008. ICT 2008. INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2008. p.1 –6.

SONG, B.; LI, M.; FANG, F. Solution and Implementation of Network Teaching System Based on GWT. In: EDUCATION TECHNOLOGY AND COMPUTER SCIENCE (ETCS), 2010 SECOND INTERNATIONAL WORKSHOP ON. **Anais...** [S.l.: s.n.], 2010. v.1, p.354 –357.

SYLLIGNAKIS, J.; ADAMAKIS, C.; PAPAZOGLU, T. A GIS web - application for power system of crete. In: UNIVERSITIES POWER ENGINEERING CONFERENCE, 2007. UPEC 2007. 42ND INTERNATIONAL. **Anais...** [S.l.: s.n.], 2007. p.414 –418.

TRIPLETT, J.; RINELL, S.; FOOTE, J. Evaluating distribution system losses using data from deployed AMI and GIS systems. In: RURAL ELECTRIC POWER CONFERENCE (REPC), 2010 IEEE. **Anais...** [S.l.: s.n.], 2010. p.C1 –C1–8.

WU, P.; YIN, K. Application research on a persistent technique based on Hibernate. In: COMPUTER DESIGN AND APPLICATIONS (ICCDA), 2010 INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. v.1, p.V1–629 –V1–631.

XIANJUN, L.; XUXIAN, Y.; HONGQUAN, W. Research on the venture capital information platform of Shanxi province based on GWT service. In: COMPUTER RESEARCH AND DEVELOPMENT (ICCRD), 2011 3RD INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. v.1, p.426 –429.

YU, X.; WU, J.; YIN, X. Study on conformance testing of hypertext transfer protocol. In: COMMUNICATION TECHNOLOGY PROCEEDINGS, 2003. ICCT 2003. INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2003. v.1, p.178 – 181 vol.1.

ZECEVIC, G. Web based interface to SCADA system. In: POWER SYSTEM TECHNOLOGY, 1998. PROCEEDINGS. POWERCON '98. 1998 INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 1998. v.2, p.1218 –1221 vol.2.