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

DEPURAÇÃO DO SISTEMA DE APOIO À DECISÃO PARA UTILIZAÇÃO NO AGRONEGÓCIO

TRABALHO DE GRADUAÇÃO

Leandro Hernandes Mânica

Santa Maria, RS, Brasil
2010

DEPURAÇÃO DO SISTEMA DE APOIO À DECISÃO PARA UTILIZAÇÃO NO AGRONEGÓCIO

Leandro Hernandes Mânica

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

Orientadora: Profª Oni Reasilvia de Almeida Oliveira Sichonany

Trabalho de graduação n° 292
Santa Maria, RS, Brasil
2010

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

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

DEPURAÇÃO DO SISTEMA DE APOIO À DECISÃO PARA UTILIZAÇÃO NO AGRONEGÓCIO

elaborado por
Leandro Hernandes Mânica

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

COMISSÃO EXAMINADORA:

Oni Reasilva de Almeida Oliveira Sichonany, Msc. (UFSM)
(Presidente/Orientador)

Roseclea Duarte Medina, Dra. (UFSM)

Iria Brucker Roggia, Msc. (UFSM)

Santa Maria, 7 de dezembro de 2010.

AGRADECIMENTOS

Agradeço a Deus que me deu conforto nos momentos de dificuldades e forçar para superá-los.

Agradeço imensamente minha família, pelo incentivo e apoio em todos os momentos da minha vida. Agradeço em especial ao meu pai, Adelar Mânica (*in memoriam*), que me ensinou valores como o respeito, dignidade e honestidade, os quais levarei comigo onde estiver.

À professora Oni Reasilvia de Oliveira Sichonany, por ter proposto este trabalho de graduação e ter acreditado que eu poderia realizá-lo, pela orientação, auxiliando nos momentos de dúvidas, e pela motivação que me transmitia a cada encontro.

Às professoras Roseclea Duarte Medina e Iria Brucker Roggia, por colaborarem, através de seus conhecimentos, com importantes sugestões a este trabalho.

A todos meus amigos da Casa do Estudante da UFSM, os distantes e os ainda presentes, que se tornaram parte da minha família e com os quais convivi agradáveis momentos.

RESUMO

Trabalho de Graduaçao
Ciência da Computação
Universidade Federal de Santa Maria

DEPURAÇÃO DO SISTEMA DE APOIO À DECISÃO PARA UTILIZAÇÃO NO AGRONEGÓCIO

Autor: Leandro Hernandes Mânica
Orientadora: Oni Reasilvia de Almeida Sichonany
Data e Local: Santa Maria, 7 de dezembro de 2010.

A depuração constitui uma das atividades fundamentais no desenvolvimento de qualquer programa, seu objetivo é a correção de comportamentos errôneos. A observação de defeitos pode ser durante um teste ou depois do *software* ser liberado. Logo, é de interesse tanto do desenvolvedor quanto do usuário que esses problemas sejam identificados e corrigidos. Para isso, existem técnicas e ferramentas que auxiliam o programador a realizar essa tarefa. Este trabalho apresenta um estudo sobre depuração de *software*, a aplicação de abordagens e técnicas de depuração em um caso de uso, o sistema de apoio à decisão para a utilização no agronegócio (SADA) – um sistema para o gerenciamento agrícola. Os resultados do trabalho são as falhas encontradas no SADA e como elas foram corrigidas.

Palavras-chave: Depuração. Técnicas de depuração. SADA.

ABSTRACT
Undergraduation Work
Computer Science
Federal University of Santa Maria

DEBUGGING OF DECISION SUPPORT SYSTEM FOR USE IN AGRIBUSINESS

Author: Leandro Hernandes Mânica
Adviser: Oni Reasilvia de Almeida Sichonany
Date and Place: Santa Maria, December 7, 2010.

Debugging is one of the fundamental activities in the development of any program, your goal is to correct erroneous behaviors. The observation of defects can be during a test or after the software is released. It is, therefore, of interest to both the developer and the user that these problems are identified and corrected. For this, there are techniques and tools that help the programmer to accomplish this task. This paper presents a study on software debugging, the application of approaches and techniques for debugging in a use case, the decision support system for use in agribusiness (SADA) - a system for agricultural management. The results of this work are failures found in the SADA and how they were corrected.

Keywords: Debugging. Debugging techniques. SADA.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
PDA	Personal Digital Assistent
PNG	Portable Network Grapics
PPGEA	Programa de Pós-Graduação em Engenharia Agrícola
SADA	Sistema de Apoio à Decisão para utilização no Agronegócio
URL	Uniform Resource Locator

SUMÁRIO

1 INTRODUÇÃO

Diversas áreas estão sendo beneficiadas pelo atual desenvolvimento tecnológico. Inclusive a agricultura, tem recorrido a sistemas que auxiliem no gerenciamento da propriedade. Esses sistemas utilizam dispositivos que coletam informações sobre a plantação, umidade do solo, ou funcionamento dos maquinários.

Através da análise das informações obtidas, várias aplicações podem ser desenvolvidas. Assim, por exemplo, pode-se desenvolver um sistema que auxilie na detecção do mau funcionamento da máquina agrícola pelo monitoramento dos dados e, a partir disso, ter um sistema de tomadas de decisão para a correção do problema.

Nesse contexto, foi desenvolvido o Sistema de Apoio à Decisão para utilização no Agronegócio (SADA), que é uma aplicação de gerenciamento das informações de desempenho de máquinas agrícolas, que estão armazenadas em um banco de dados. O SADA é parte integrante em um projeto entre o curso de Ciência da Computação e o Programa de Pós-Graduação em Engenharia Agrícola (PPGEA).

Por mais experiente que o desenvolvedor de *software* seja, é natural introduzir defeitos durante a construção de um programa. Além disso, à medida que esses se tornam maiores e mais complexos, aumenta a probabilidade de inserção de erros no código-fonte do programa.

Assim, da codificação de um programa de computador até suas execuções realizadas pelo usuário final, é bem provável que defeitos sejam detectados e, conseqüentemente, precisem ser corrigidos. Quando um defeito é detectado, não se sabe exatamente onde ele aconteceu, sabe-se apenas que o mesmo existe. O processo de localização e correção destes defeitos é chamado de depuração (PRESSMAN, 2006).

Este trabalho propõe a realização da depuração do SADA. Para isso, será necessário fazer um estudo sobre depuração, assim como do código do SADA, e as APIs e bibliotecas que são utilizadas nele. Também serão apresentadas as abordagens e técnicas de depuração.

No capítulo 2 é apresentado conceito de depuração, o SADA e duas APIs Google, Google *Chart* e Google *Maps*. Na seção de depuração, é definido o seu significado, em que momentos ela ocorre e como o processo de depuração é feito. Também são apresentadas algumas dificuldades em se depurar, abordagens e técnicas utilizadas e os depuradores, ferramentas que auxiliam na depuração. Na seção seguinte, é apresentado o SADA, com os seus casos de uso. As duas últimas seções são sobre APIs Google, API Google *Chart* e API Google *Maps*, em que é feito uma breve apresentação de como as utilizar.

O capítulo 3 aborda como foi desenvolvido o trabalho. Ou seja, como foi realizado o processo de depuração no SADA.

Os resultados obtidos são apresentados no capítulo 4, que mostra uma série de falhas e a sua correção.

2 REVISÃO DE LITERATURA

2.1 Depuração

É comumente definida como a tarefa de localizar erros ou defeitos que provocam falhas em um programa, e após, projetar e implementar mudanças no programa que deverão corrigir os defeitos (BEIZER, 1990). No contexto desse trabalho uma falha é definida como um desvio na especificação de um projeto, que pode ser uma saída incorreta ou pode provocar a parada da execução do programa.

A depuração deve ser realizada sempre que ocorre uma falha. No entanto, defeitos podem ser revelados em diferentes fases do ciclo de vida de um *software*. Por isso os processos de depuração podem ocorrer durante a codificação, depois de testes e durante a manutenção (CHAIM, 2001).

A depuração durante a codificação é uma atividade complementar à da implementação. Já a depuração depois de testes é ativada pelo teste sistemático bem sucedido, ou seja, aquele que revelou defeitos, podendo se beneficiar das informações coletadas durante essa fase (SOMMERVILLE, 2005). A manutenção, que consiste na alteração do código, pode ser causada por um defeito revelado após o *software* ter sido liberado, ou pela necessidade de se acrescentar novas funcionalidades nele. Nesses casos, nova depuração se faz necessária.

A relevância da atividade de depuração tem dirigido esforços em duas direções: no entendimento do processo de depuração e no desenvolvimento de técnicas que aumente a sua produtividade. Para isso, vários experimentos foram desenvolvidos com o objetivo de entender o processo de depuração de *software* e estabelecer um modelo de depuração (DELAMARO et al, 2007).

2.1.1 O Processo de Depuração

A depuração se inicia quando é verificada uma falta de correspondência entre o que é observado e a especificação de um projeto. Isto pode acontecer quando o desenvolvedor realiza testes no *software* antes de seu lançamento, ou quando ele é utilizado pelo usuário após seu lançamento. Em muitos casos, um engenheiro de

software, ao avaliar os resultados de um teste, se defronta com um indício sintomático do problema, ou seja, a manifestação externa do erro e a causa interna do erro não têm nenhuma relação óbvia entre si (PRESSMAN, 2006). O processo de depuração tenta ligar o sintoma à causa, levando assim à correção do erro.

O processo de depuração consiste de três atividades sucessivas: a) (re-) manifestação da falha; b) localização do defeito, e; c) correção do defeito (Figura 2.1).

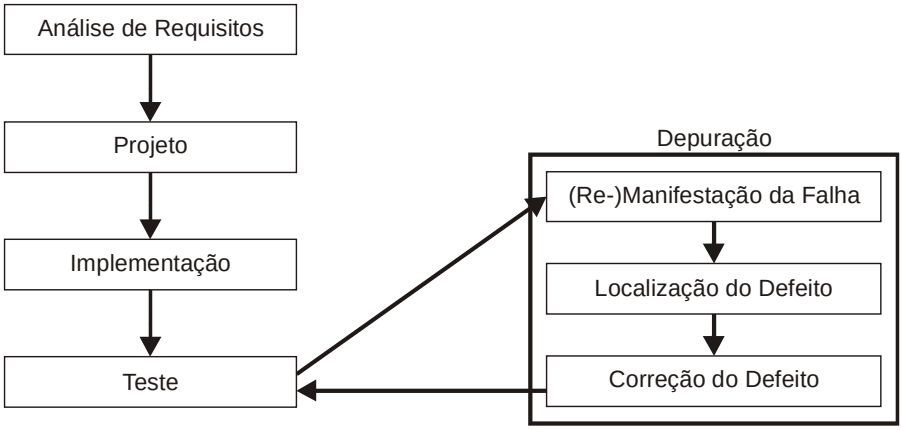

Figura 2.1 – Etapas da depuração

Durante a re-manifestação da falha, a falha que foi detectada, enquanto se realizava o teste de *software* ou durante o seu uso, é reproduzida. Durante a localização dos defeitos, o código-fonte do programa é percorrido a fim de encontrar os lugares dos defeitos. E, finalmente, durante a correção de um defeito, realizam-se as mudanças necessárias no código-fonte para a correção do defeito. Depois o programa modificado é testado novamente (LIAN et al, 1997).

O processo de depuração sempre terá um dentre dois resultados:

a. A causa será encontrada, corrigida e removida; ou
b. A causa não será descoberta.

Neste último caso a pessoa que executa a depuração pode suspeitar de uma causa, projetar um caso de teste para ajudar a validar a hipótese e trabalhar na direção da correção do erro de forma iterativa.

Entre as atividades do processo de depuração, encontrar a localização dos defeitos é a tarefa mais difícil (MYERS, 1976). Isto acontece principalmente com programas grandes, pois diversos passos podem ser necessários até identificar o defeito. Reparar o defeito é relativamente simples, podendo ser mais complexo se for em conseqüência de um erro de projeto (ARAKI et al, 1991).

2.1.2 Dificuldades em Depurar

A depuração geralmente é uma tarefa difícil e trabalhosa (PRESSMAN, 2006). A seguir são descritos alguns aspectos que contribuem para essa característica. As dificuldades na depuração variam de acordo com o tipo de *software* que é desenvolvido.

a) O sintoma e a causa podem ser geograficamente remotos. Ou seja, o sintoma pode aparecer em uma parte do programa, enquanto a causa pode, de fato, estar localizada num lado bastante remoto;
b) O sintoma pode desaparecer (temporariamente) quando outro erro é corrigido;
c) O sintoma pode, de fato, ser causado por não-erros, por exemplo, como imprecisões de arredondamento. Isto pode acontecer em cálculos envolvendo variáveis do tipo real de precisão simples (tipo *float* em algumas linguagens de programação, como C);
d) O sintoma pode ser causado por um erro humano que não é facilmente rastreado;
e) O sintoma também pode ser resultado de problemas de temporização, e não de problemas de processamento. Pode ocorrer em sistemas que usam memória compartilhada ou uma interface de passagem de mensagens, um caso típico é um sistema produtor-consumidor que operam em velocidades diferentes. O consumidor pode acessar informações desatualizadas porque o produtor delas não as atualizou (SOMMERVILLE, 2005);
f) Pode ser difícil de reproduzir com precisão as condições de entrada. Por exemplo, uma aplicação de tempo real em que a entrada é indeterminada;
g) O sintoma pode ser intermitente. Isso é particularmente comum em sistemas embutidos que acoplam *hardware* e *software* de forma inerente;
h) O sintoma pode ser devido em causas que estão distribuídas por uma série de tarefas que são executadas em diferentes processadores (CHEUNG et al, 1990);

Além disso, poder ser que a pessoa responsável pela depuração de um programa não seja a mesma que o desenvolveu, ou que o programador original não se lembre mais das funções implementadas.

Juntamente com essas dificuldades apresentadas e também do fato de que mais de cinqüenta por cento do custo total de desenvolvimento de programas serem gastos na fase de testes e depuração (PRESSMAN, 2006), a depuração tem sido uma das áreas menos desenvolvidas em engenharia de *software* (ARAKI, 1991).

2.1.3 Abordagens à Depuração

As abordagens à depuração foram desenvolvidas com o objetivo de disciplinar o processo de depuração (ARAKI, 1991). Em geral, existem várias abordagens para a depuração de programas, algumas delas são:

a) Baseada na análise de programas: Nesta abordagem, o programador primeiramente lê o código-fonte do programa, analisa-o, e tenta entendê-lo. No processo se deve verificar se o programa é consistente com as especificações. Se uma discrepância ocorrer entre as especificações e o código, o programador pode tentar localizar o defeito no código e corrigi-lo (SEVIORA, 1987).
b) Baseada na entrada e saída do programa: Terminada a codificação, o programa é compilado e executado com entradas selecionadas. Com isso, se a saída do programa não é a esperada, tenta-se identificar em que parte do programa está o defeito. O programador pode gerar diversas hipóteses a respeito do defeito e sua localização, então executar novamente o programa para, assim, eliminar algumas hipóteses e examinar as partes suspeitas para identificar o defeito. Esta abordagem é uma das mais utilizadas (VIRAVAN, 1994).
c) Baseada no rastreamento interno do programa: Em certas circunstâncias, a única ferramenta disponível ao programador de depuração é o rastreador. O programa é depurado a partir do código e exemplos de saída. Na maioria dos casos, o programador conta com o auxílio de ferramentas, depuradores, que permitem coletar informações sobre o estado das variáveis do programa em determinado momento durante a execução. Com essa informação o programador pode localizar os lugares onde ocorreu o defeito.

Cada abordagem requer um tipo de conhecimento. Para a abordagem de análise de programa, utiliza-se o conhecimento do problema de decomposição e construção do programa para entender suas partes e suas interações. Já na abordagem baseada na entrada e saída, é necessário o conhecimento de diagnóstico para a elaboração das hipóteses sobre a localização dos defeitos. Finalmente a abordagem baseada no rastreamento interno do programa, o programador confia no seu conhecimento a respeito do que rastrear e de como analisar as informações coletadas (CRUZES, 1999).

2.1.4 Técnicas de Depuração

A maioria das técnicas ajuda na avaliação de hipóteses sobre o comportamento do programa e a formulação de hipóteses sobre as localizações dos defeitos e suas causas (CRUZES, 1999). Essencialmente, as técnicas de depuração, utilizadas hoje, são as mesmas desenvolvidas há pelo menos trinta anos (DELAMARO, 2007).

A seguir são descritas algumas técnicas que podem ser utilizadas no processo de depuração depois do teste, apresentando suas vantagens e desvantagens.

2.1.4.1 Força Bruta

A categoria de depuração por força bruta é o método mais fácil de implementar e o menos eficiente para isolar a causa de um erro. Aplica-se esse método quando tudo mais falha. Nesta técnica são inseridos comandos de impressão em lugares onde se suspeita que possa estar o erro. A partir disso, é esperado entender o comportamento do programa através das informações mostradas (PRESSMAN, 2006).

2.1.4.2 Asserções

Depuração com asserções é baseada na inclusão de parte das especificações do programa no seu código-fonte, de maneira que a ação é ativada toda vez que a especificação parcial do programa é violada durante a execução. Dependendo do resultado, a ação pode ser de abortar um processo, ou retornar os números das linhas do código-fonte que não atenderam as assertivas, junto com outras informações úteis a depuração. Algumas linguagens fornecem mecanismos para codificar essas restrições (STAA, 2000).

2.1.4.3 *Breakpoints*

Um *breakpoint* é um ponto no fluxo de execução do programa onde a execução normal do programa é suspensa e as informações sobre o estado atual do programa são salvas. Através disso, o programador pode examinar e modificar o estado de um programa. Esta técnica não requer nenhum código extra no programa, evitando a inserção de erros ao adicionar novos comandos (CHEUNG, 1990).

2.1.4.4 *Backtracking*

É uma técnica bastante comum à depuração que pode ser usada com sucesso em programas de pequeno porte. Inicia-se no local em que o sintoma foi descoberto, o código-fonte é rastreado para trás (manualmente ou através de ferramentas) até que o local da causa seja encontrado. Não é indicado para programas grandes, pois à medida que o número de linhas do código aumenta, o número de potenciais caminhos de *backtrancking* pode se tornar muito alto inviabilizando o seu uso (AGRAWAL, 1991).

2.1.4.5 *Tracing*

Essa é a técnica mais utilizada na prática. O sucesso dessa técnica se deve a três fatores: seu uso requer apenas treinamento básico; o custo em tempo de depuração é razoável; e sua disponibilidade é ampla porque está presente em qualquer ambiente de programação. Esse tipo de depuração envolve o rastreamento de eventos e a sua inspeção do estado do programa no momento em que ocorre um evento (DELAMARO, 2007).

O objetivo dessa técnica é rastrear um determinado ponto do programa durante a execução e inspecionar o valor de variáveis escolhidas, ou seja, examinar o estado parcial do programa. Essas tarefas são facilitadas com o uso de depuradores.

2.1.4 Depuradores

Um depurador é um programa de computador utilizado para testar outros programas, com o objetivo de auxiliar o programador na detecção de erros durante o processo de depuração (DELAMARO et al, 2007).

Um depurador simbólico, ou depurador de código-fonte, tipicamente mostra a posição no código-fonte, de um programa ou sistema, onde a execução foi interrompida. Caso seja um depurador de linguagem de máquina, ele mostra a linha onde ocorreu o problema através de desmontagem – processo de conversão código de máquina em código escrito em linguagem de montagem. Depuradores comumente estão presentes em Ambientes Integrados de Desenvolvimento (IDEs – do inglês *Integrated Development Environments*).

Os depuradores oferecem um conjunto de funcionalidades, que geralmente são: execução passo a passo, suspender a execução do programa para analisar seu estado, acompanhar os valores das variáveis e inserir pontos de paradas (*breakpoints*). Alguns oferecem opções mais avançadas, tais como alterar o estado, fluxo ou variáveis em tempo de execução e a capacidade de voltar na execução (*backtracking*).

A seguir são apresentados alguns depuradores.

2.1.4.1 GNU *Debugger*

O GNU *Debugger*, mais conhecido como GDB, é o depurador de código livre do projeto GNU. Ele pode ser usado para depuração de muitos sistemas escritos em linguagem de programação C, C + +, FORTRAN e muitas outras (GDB, 2010). Esses programas podem estar executando na mesma máquina que o GDB (nativo) ou em outra máquina (remoto). O GDB pode ser executado em sistemas do tipo UNIX e Windows variantes Microsoft.

O GDB oferece o monitoramento e alteração de valores de variáveis internas do sistema, e pode chamar funções de forma independente do fluxo do programa. Suporta *backtrackig*. Ele não possuiu uma interface gráfica, o usuário interage através do interpretador de comandos. Porém, muitas IDEs o utilizam para a depuração, implementando uma interface.

2.1.4.2 JSwat

O JSwat é um depurador gráfico Java que foi escrito utilizando a Java Platform Debugger Architecture, uma coleção de APIs para depurar programas escritos em java. Ele está disponível gratuitamente em ambas as formas, o código-fonte e o executável (JSWAT, 2010). Além da interface gráfica, existe uma versão de console e sua operação é muito semelhante ao jdb, que é o depurador incluído com o Kit de Desenvolvimento Java.

Dentre as suas funcionalidades estão a inserção de *breakpoints*, invocação de métodos e acompanhamento de valores de variáveis, entre outros.

2.1.4.3 IBM *Rational Purify*

O IBM *Rational Purify* é um programa depurador de memória utilizado por desenvolvedores de software para detectar erros de acesso à memória em programas, especialmente aqueles escritos em C ou C + +. Não é uma ferramenta gratuita (IBM, 2010).

Ele permite a verificação dinâmica durante a execução de um programa. Possui verificação estática, ou análise estática de código, que envolve a detecção de erros no código-fonte sem precisar a compilação ou execução, apenas por descobrir inconsistências lógicas.

Também tem funcionalidades de monitoramento, *breakipoints*, e outras comuns aos depuradores.

2.2 SADA

O Sistema de Apoio à Decisão para a utilização no Agronegócio (SADA) é um conjunto de aplicações desenvolvidas para auxiliar no gerenciamento do agronegócio. É constituído por duas partes: o T-SADA, responsável pela transmissão dos dados coletados por sensores instalados em uma máquina agrícola, e o G-SADA, responsável pelo gerenciamento desses dados (SICHONANY, 2010).

O T-SADA é um sistema de coleta de dados (através de sensores colocados em pontos específicos de uma máquina agrícola), armazenamento dos dados coletados (em um depósito de dados) e a transmissão a um servidor de Banco de Dados. Essa coleta é executada em intervalos de tempo definidos (SANTOS, 2010).

As informações geradas pelo G-SADA têm como base de decisão os dados provenientes das consultas a um banco de dados que contém as informações coletadas sobre o funcionamento de máquinas agrícolas, tais como, velocidade, posição, consumo de combustível, etc.

A partir dos dados no Banco de Dados, podem ser desenvolvidas funcionalidades que vão desde o gerenciamento da informação até cálculos estatísticos. Desta forma, o operador pode ser informado se ocorrer algum procedimento errôneo no manejo com a máquina, de forma que possa ser corrigido o mais rápido possível. O operador será informado através do envio de mensagens para um dispositivo móvel, tal como um celular, *smartphone* ou PDA.

O G-SADA foi concebido em linguagem de programação orientada a objeto Java, e se utilizou JavaScript para as funcionalidades de suas páginas HTML.

A figura 2.2 mostra a tela de entrada do G-SADA.

Figura 2.2 – Tela entrada do SADA

2.2.1 Usuários

O sistema possui três tipos de usuários que irão interagir:

- Administrador: é responsável por fazer a manutenção das informações das tabelas contidas no banco de dados, incluindo, alterando e excluindo dados,
- Gerente: pode enviar e receber informações, além de poder executar as funções disponibilizadas ao administrador,
- Operador: recebe informações, através de um celular, enviadas pelo sistema sinalizando valores operacionais fora de padrão durante a operação agrícola.

2.2.2 Diagrama dos casos de uso

As figuras 2.3 e 2.4 mostram os casos de uso (funções) do G-SADA (SICHONANY, 2010).

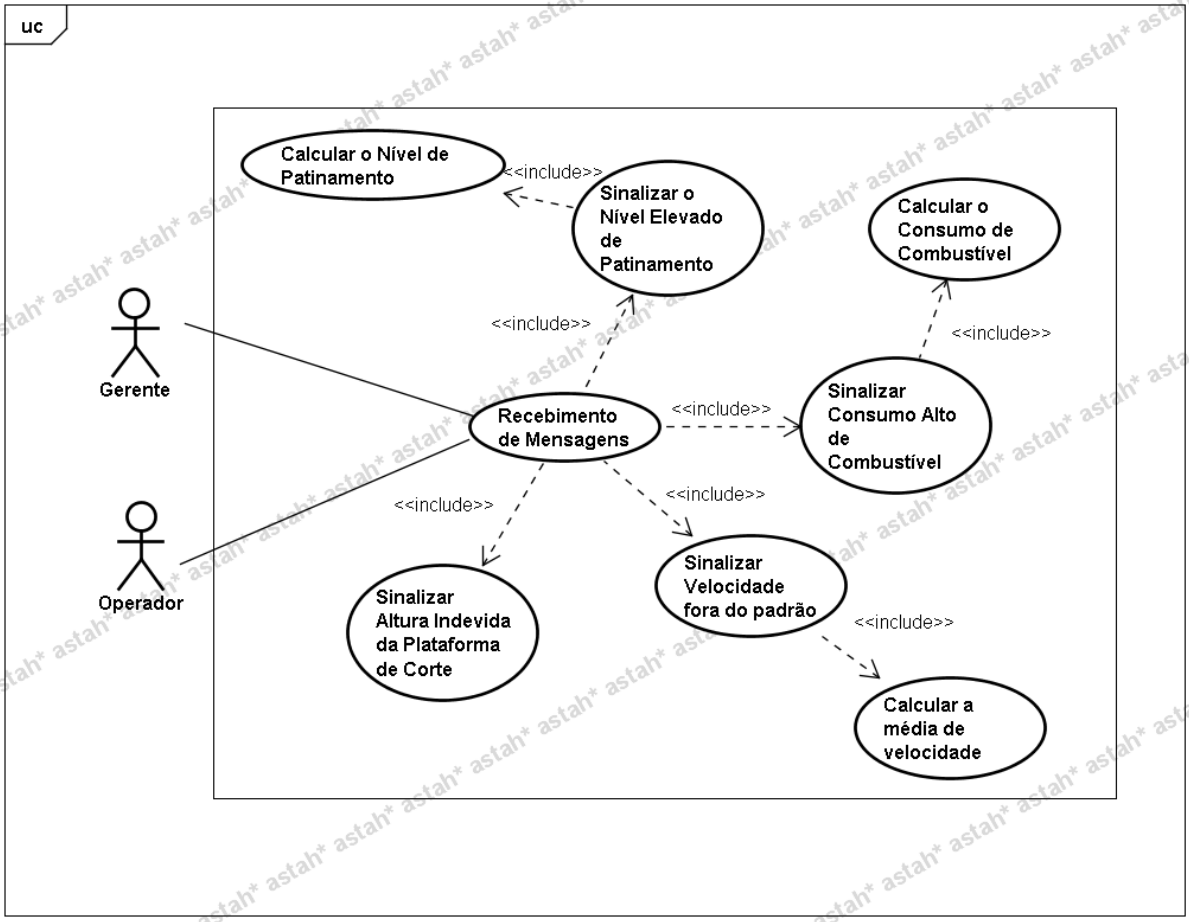

Figura 2.3 – Diagrama de Casos de Uso – Envios de Mensagens

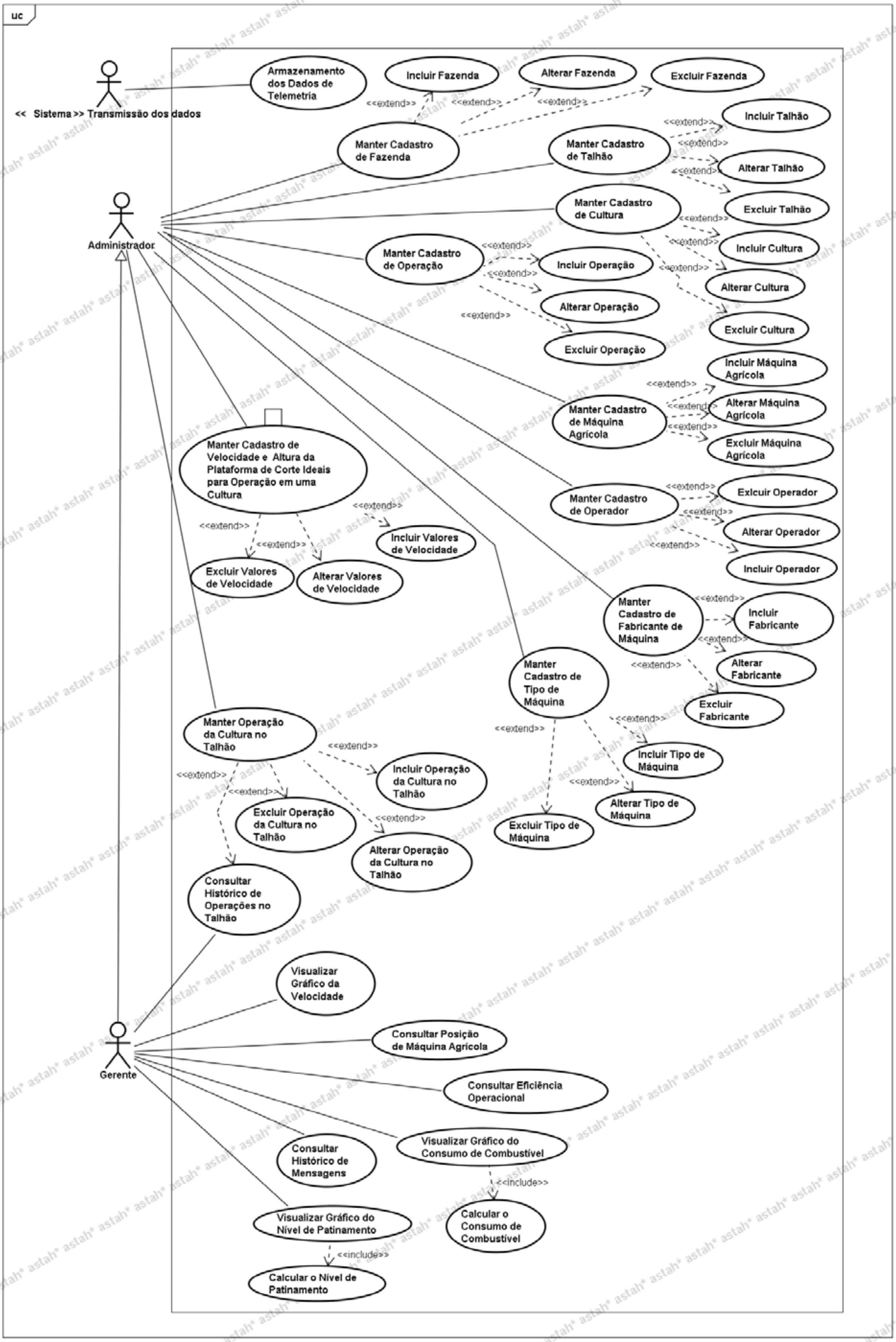

Figura 2.4 – Diagrama de Casos de Uso - Manutenção de Cadastros e Consultas.

A Figura 2.3 apresenta as funções de envios de mensagens e na figura 2.4, as funções de manutenção de cadastros e consultas.

2.2.2 Descrição dos casos de uso

A seguir estão as telas correspondentes aos casos de uso do G-SADA, uma breve definição e as operações de alguns casos. A figuras 2.5 mostra as opções de cadastros e a figura 2.6, as opções de consultas.

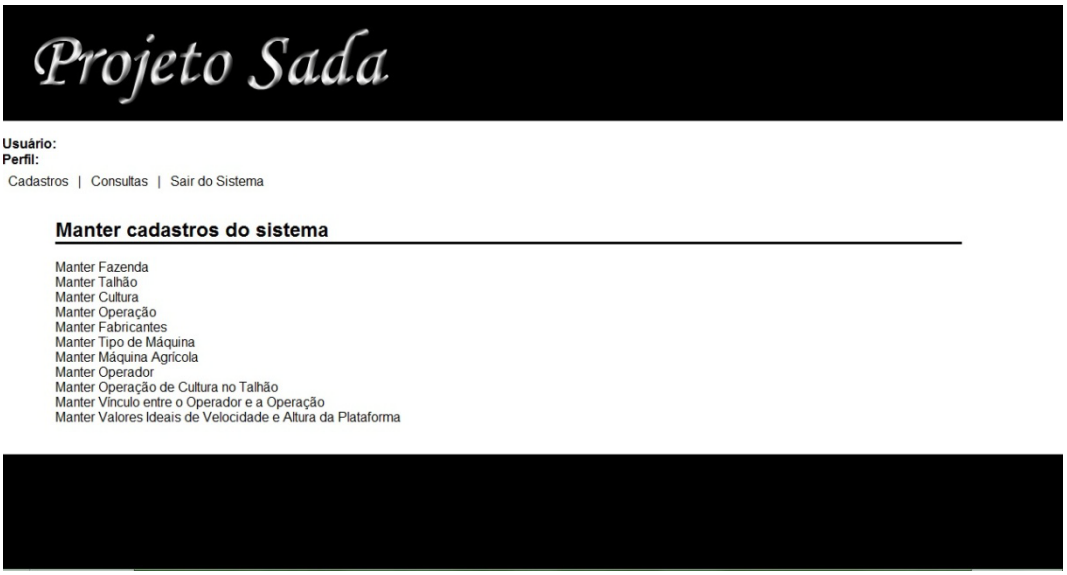

Figura 2.5 – Tela de cadastros

Figura 2.6 – Tela de consultas

Dependendo do tipo de usuário, ele pode ter acesso a cadastros ou a cadastros e consultas.

2.2.2.1 Cadastros

São operações de manutenção das informações nas tabelas do banco de dados. Os casos são:

- Manter cadastro de fazenda,
- Manter cadastro de talhão,
- Manter cadastro de cultura,
- Manter cadastro de operação,
- Manter cadastro da máquina agrícola,
- Manter cadastro do operador,
- Manter cadastro de fabricante de máquina,
- Manter cadastro de tipo de máquina,
- Manter operação da cultura no talhão,
- Manter vínculo o operador e a operação,
- Manter cadastro de velocidade e altura da plataforma ideal para operação em uma cultura.

É possível realizar a inclusão, alteração ou a remoção do cadastro de cada um dos itens mencionados anteriormente.

Os usuários do tipo administrador e gerente estão habilitados a realizar operações de cadastros.

A figura 2.7 mostra o caso Manter cadastro de fazenda. Nesta figura a fazenda Buricaci já está cadastrada, então para esta fazenda é possível alterar seus dados ou excluí-la.

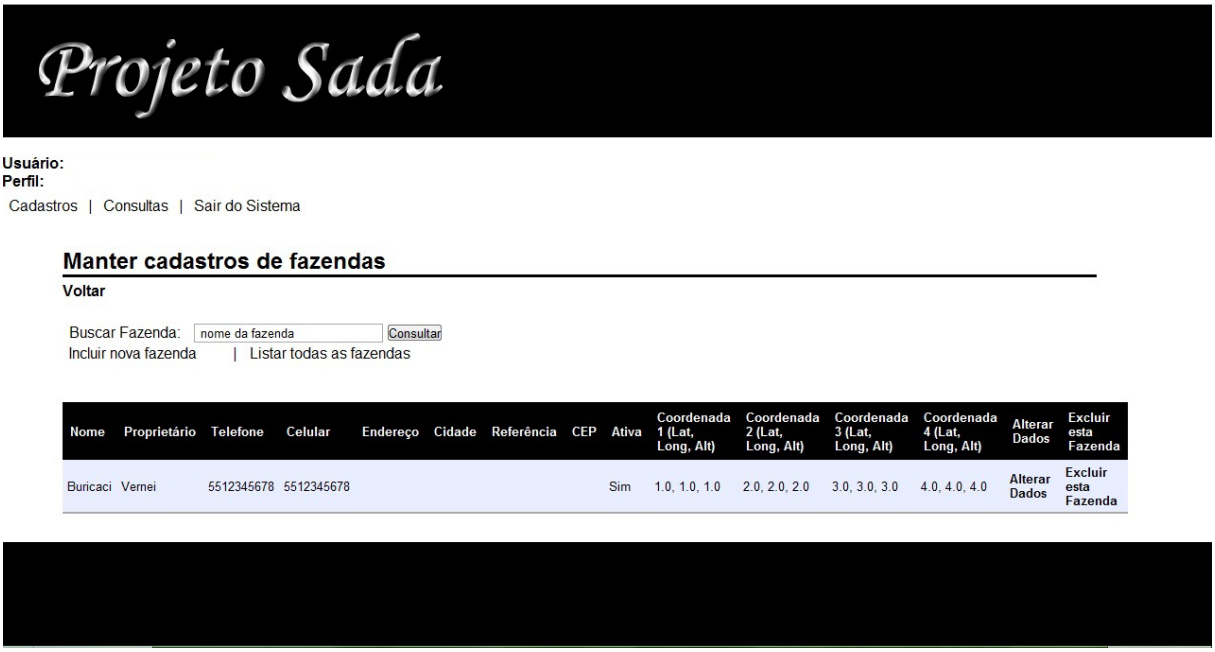

Figura 2.7 – Manter cadastro de fazendas.

Para cadastrar uma nova fazenda, deve-se clicar em “Incluir nova fazenda” e preencher os campos contendo as informações referentes à nova fazenda, como mostra a figura 2.8.

Figura 2.8 – Incluir nova fazenda.

2.2.2.2 Consultas

As consultas mostram informações, valores resultantes de cálculos ou geram gráficos a partir dos dados armazenados no banco de dados.

As seguintes consultas estão disponíveis:

- Consultar histórico de operações no talhão,
- Consultar consumo de combustível,
- Visualizar o gráfico do consumo de combustível,
- Calcular o nível de patinamento,
- Visualizar o gráfico do nível de patinamento,
- Consultar eficiência operacional,
- Consultar a posição de uma máquina agrícola,
- Visualizar o gráfico da velocidade,
- Consultar o histórico de mensagens,
- Calcular a média de velocidade.

Para cada função é possível escolher um conjunto de opções que se deseja usar como filtro. Como, por exemplo, a fazenda, o talhão, tipo de operação, a cultura, etc.

O usuário do tipo gerente é o único a poder realizar consultas.

2.2.2.3 Mensagens de alerta

Mensagens enviadas pelo SADA para ao operador para alertar quando for verificado que os valores operacionais estão fora do padrão estabelecido. As mensagens podem ser das seguintes funções de alertas:

- Sinalizar o nível elevado de patinamento,
- Sinalizar o consumo alto de combustível,
- Sinalizar velocidade fora do padrão,
- Sinalizar altura indevida da plataforma de corte.

2.3 APIs Google

A empresa Google oferece uma variedade de APIs e serviços *online* para desenvolvedores de programas e sistemas *web*. Entre eles o Google *Maps*, Google *Toolbar*, Google *Chart*, etc. A seguir são abordados dois desses serviços, o Google *Chart* e Google *Maps*. Embora essas APIs possuam uma vasta referência, os tópicos abordados neste trabalho se limitam aos atributos utilizados na geração dos gráficos e mapas para o SADA.

2.3.1 Google *Chart*

A API do Google *Chart* é uma ferramenta simples que permite gerar gráficos e incorporá-los em uma página *web*. Para isso, inserem-se os dados e parâmetros de formatação em uma requisição HTTP, e o Google retorna uma imagem no formato *portable nettwork graphics* (PNG) contendo o gráfico (GOOGLE CODE, 2010).

Essa API pode gerar vários tipos de gráficos. Alguns exemplos são: gráfico tipo pizza, gráficos de linha, gráficos de barras, gráficos gerados a partir de equações, gráfico de dispersão, etc. Ainda é possível combinar dois tipos de gráficos.

Todas as informações referentes ao que se deseja gerar, tais como, os dados do gráfico, o tamanho, cores e rótulos, fazem parte da URL. Para que o gráfico seja exibido como uma imagem diretamente em uma página web, basta adicionar esta URL como sendo uma imagem dentro da tag HTML `<img src= “…” >`.

As URLs, que utilizam essa API, devem começar com:

```
http://chart.apis.google.com/chart?
```

Após esse trecho, devem ir os parâmetros que especificam os dados do gráfico e da aparência. Os parâmetros são pares `nome = valor`, separados por um caractere e comercial (&), e os parâmetros podem estar em qualquer ordem, após o caractere “?”. Todos os gráficos requerem, no mínimo, os seguintes parâmetros: `cht` (tipo de gráfico), `chd` (dados) e `chs` (tamanho). No entanto, há

muito mais parâmetros para opções adicionais, e podem ser especificados tantos parâmetros adicionais quanto o tipo de gráfico suportar.

Parâmetros obrigatórios:

- **cht**: este parâmetro especifica qual é o tipo de gráfico, ou seja, formato pizza, linha, barras, etc.

  Sintaxe: `cht = <type>`, onde `<type>` deve ser substituído pelo tipo de gráfico que se quer.

- **chd**: este parâmetro indica em qual formato os dados estão sendo enviados. O mais comum é enviar no formato de texto simples. Esse formato permite que se especifiquem valores de ponto flutuante de zero a cem, inclusive. Valores abaixo de zero ou o caractere "_" são considerados nulos e valores acima de cem são truncados para cem.

  Sintaxe: `chd = <type>: Val, val, … | val, val, …`, onde `<type>` é o tipo de dado e `<val>` são os valores. As séries de valores são separadas pela barra vertical (caractere "|").

- **chs**: este parâmetro determina a largura e a altura, respectivamente, de um gráfico.

  Sintaxe: `chs = <width>x<height>`, onde `<width>` é a largura em pixels com e `<height>` é a altura. O valor máximo é de 1000 pixels, para ambos, e o resultado da multiplicação da altura pela largura não pode ultrapassar o valor de 300000.

Alguns parâmetros opcionais:

- **chco:** parâmetro que especifica a cor do gráfico.

  Sintaxe: `chco = <color>` onde `<color>` é uma cor no formato hexadecimal RRGGBB.

  Obs.: A sintaxe exata e significado podem variar conforme o tipo de gráfico.

- **chds:** parâmetro referente a escala do gráfico.

  Sintaxe: `chds = <min>,<max>`, onde `<min>` é o menor valor e `<max>` é o maior valor.

- **chxl:** este parâmetro especifica o rótulo de um determinado eixo.

  Sintaxe: `chxl = <axis_index>: <lable1> | <lable2> | …`, onde `<axis_index>` é o eixo conforme definido pelo parâmetro `chxt`, o primeiro

eixo é referido pelo valor zero, e `<lable1> | <lable2> | …`, são os rótulos dos eixos.

- **chxr:** parâmetro que especifica o intervalo do eixo.
  Sintaxe: `chxr = <axis_index>, <start_val>, <end_val>, <step>`, onde `<axis_index>` é o eixo conforme definido pelo parâmetro `chxt`, o primeiro eixo é referido pelo valor zero, `<start_val>` e `<end_value>` são números que definem o menor e o maior, respectivamente, valores do eixo e `<step>` é um número que define o intervalo entre os valores que são mostrados no eixo, sendo este último opcional.
- **chxt:** este parâmetro define quais eixos terão seus valores visíveis.
  Sintaxe: `chxt = <axis1>, <axis2>, …`, onde `<axis1>, <axis2>, …`, são os eixos.

A figura 2.9 a seguir é um exemplo básico gerado a partir da URL:

```
http://chart.apis.google.com/chart?chs=400x200&chd=t:60,40&
                    cht=p3&chl=A=60%|B=40%
```

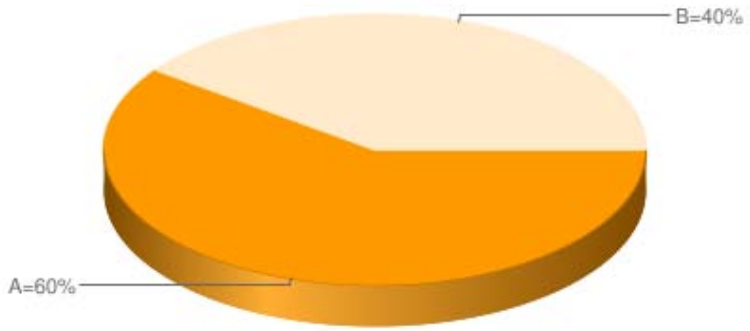

Figura 2.9 – Exemplo básico de gráfico.

Para o exemplo de gráfico da figura 2.9 os parâmetros são: chs define o tamanho do gráfico, possui os valores de 400 pixels de comprimento por 200 pixels de altura; chd, parâmetro que especifica o tipo de dado, no caso t indica que os dados são do tipo texto simples, e após vem os valores dos dados; cht é o parâmetro que se refere ao tipo de gráfico, o valor p3 indica que é um gráfico do tipo pizza em três dimensões; e chl recebe os valores dos rótulos, os rótulos são associados aos dados na ordem em que aparecem, isto é, o primeiro valor de chl se refere ao valor do primeiro dado de chd.

2.3.2 Google *Maps* JavaScript API

Esta API fornece diversos utilitários para manipulação de mapas, como os mostrados na página *web* http://maps.google.com, e adição de conteúdo aos mapas por meio de vários serviços, permitindo criar aplicativos de mapas.

A API permite que seja incorporado o Google *Maps* a páginas da *web*. Encontra-se atualmente na versão 3, sendo essa a versão oficial. Foi desenvolvida especialmente para ser mais rápida e apresentar maior compatibilidade com dispositivos móveis e também com aplicativos tradicionais para navegadores de computadores pessoais. Para os aplicativos desenvolvidos com a versão anterior, a versão 2, é recomendado fazer a migração do código para a versão 3, pois a versão 2 tem previsão de remoção, conforme políticas de remoção da Google (GOOGLE CODE, 2010).

O elemento fundamental de qualquer aplicativo da API do Google Maps V3 é o próprio "mapa". Isso é feito através do uso do objeto fundamental google.maps.Map e as noções básicas de operações de mapa.

Para carregar a Google *Maps* API deve-se incluir o seguinte trecho nas páginas HTML:

```
<script type=”text/javascript”
        src=http://google.com/maps/api/js?sensor=opt>
</script>
```

A URL passada aponta para o local de um arquivo JavaScript que carrega todos os símbolos e as definições necessárias ao uso da API do Google *Maps* V3 e a definição do parâmetro `”sensor”` serve para indicar se o aplicativo usa um sensor para determinar a localização do usuário, no caso `”opt”` deve ser substituído por um de seus possíveis valores que é `”true”` ou `”false”`. Também é necessário reservar um espaço na página para que o mapa seja exibido. Normalmente isso é feito através da tag `<div>` e obtendo uma referência para ele no DOM (modelo de objeto do documento) do navegador.

2.3.2.1 O objeto elementar google.maps.Map

A classe JavaScript que representa um mapa é a classe Map. Os objetos dessa classe definem um único mapa em uma página. Pode-se criar mais de uma instância dessa classe. Cada objeto definirá um mapa diferente na página. Cria-se uma nova instância dessa classe usando o operador JavaScript *new*. Ao criar uma nova instância de mapa, deve-se especificar um elemento HTML `<div>` na página como um recipiente para o mapa. A função `Map()` é o construtor e sua definição está a seguir:

Construtor	**Descrição**
Map(IdDiv, opts)	Cria um novo mapa dentro do recipiente de HTML especificado, geralmente um elemento DIV

Quadro 1 – Descrição da função Map.

Para inicializar um mapa, primeiro tem que ser criado um objeto MapOptions para conter as variáveis de inicialização do mapa. Os atributos obrigatórios de um objeto MapOptions são:

Atributo	**Tipo**	**Descrição**
Center	LatLng	Centro do mapa.
mapTypeId	MapTypeId	O tipo inicial do mapa.
zoom	Inteiro	O nível de zoom inicial.

Quadro 2 – Descrição do objeto MapOptions.

O atributo center é do tipo LatLng. O construtor da classe LatLng possui dois parâmetros do tipo inteiro que são a latitude e longitude, respectivamente.

O tipo de mapa é definida pelo atributo mapTypeId e são do tipo MapTypeId que possui os seguintes identificadores:

Constante	Descrição
HYBRID	Exibe uma mistura entre a visualização de satélite e a
ROADMAP	padrão.
SATELLITE	Exibe uma visualização de mapa padrão.
TERRAIN	Exibe uma visualização de imagens de satélite.
	Exibe um mapa físico com base nas informações do terreno.

Quadro 3 – Descrição dos identificadores do tipo MapTypeId.

A figura 2.10 é um exemplo de mapa gerado a partir do código abaixo.

```
<html>
<head>
    <meta name="viewport" content="initial-scale=1.0, user-
                scalable=no" />
    <script type="text/javascript"
                src="http://maps.google.com/maps/api/js?
                sensor=true">
    </script>
    <script type="text/javascript">
        function initialize() {
        var latlng = new google.maps.LatLng(-29.65, -53.75);
        var myOptions = {
           zoom: 10,
           center: latlng,
           mapTypeId: google.maps.MapTypeId.ROADMAP
           };
        var map = new google.maps.Map(
                        document.getElementById("map_canvas"),
                        myOptions);
        }
    </script>
</head>
<body onload="initialize()">
    <div id="map_canvas" style="width:100%; height:100%"></div>
</body>
</html>
```

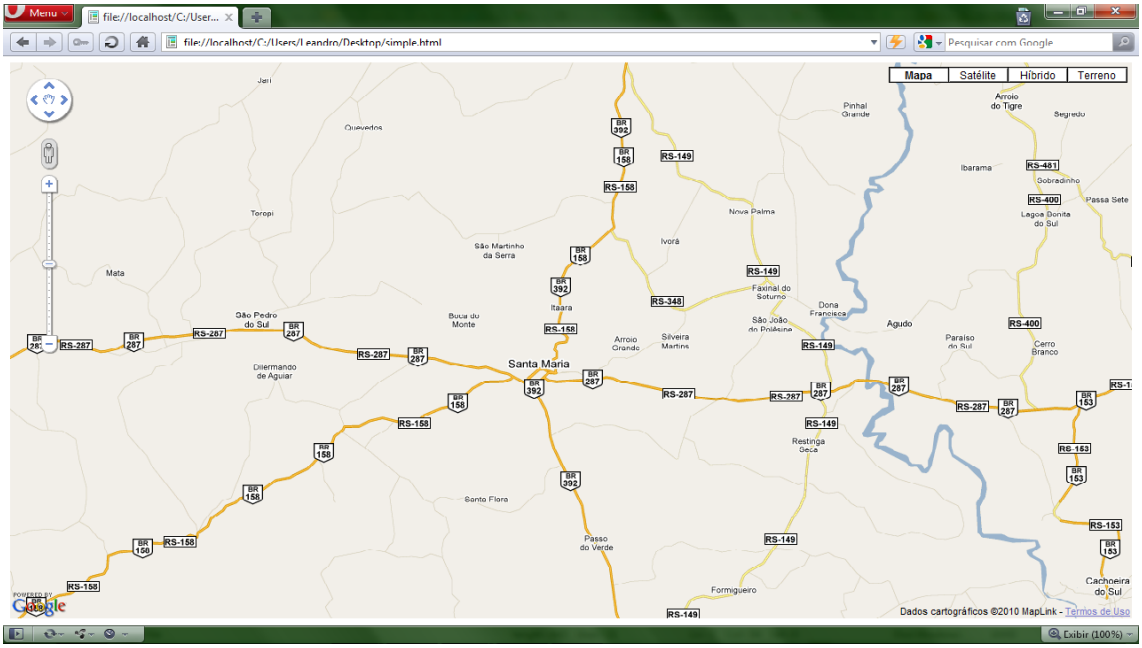

Figura 2.10 – Exemplo de mapa.

2.4 Considerações Finais

Inicialmente, neste capítulo, foi apresentado um estudo sobre a depuração, o processo de depuração, suas dificuldades, as abordagens e as técnicas. Também apresentou o depurador, uma ferramenta auxiliar no processo de depuração.

Em seguida, foi abordado o SADA. Mostrou-se as funcionalidades de cadastros, consultas e mensagens de alertas oferecidas. Algumas consultas resultam em gráficos e mapas, os quais são gerados por APIs disponíveis na *web*.

Na última seção foi feita uma breve descrição das APIs envolvidas na construção de gráficos e mapas, Google *Chart* e Google *Maps* JavaScript API respectivamente, utilizadas no SADA.

No próximo capítulo será abordada a observação de falhas e o modo que elas serão corrigidas.

3 MATERIAL E MÉTODOS

3.1 Observação das falhas

O ambiente de desenvolvimento Eclipse é utilizado, por ser um IDE usado para desenvolver aplicativos em Java e também suporta o desenvolvimento de sistemas *web*, como é o caso do SADA. Ele possui um navegador interno, mas é necessário integrar a ele um servidor caso queira executar serviços *web*. Para isso, o servidor *web* Java Tomcat foi integrado ao Eclipse.

A partir disso, foi executado o SADA e se testou cada uma de suas funcionalidades, com a finalidade de detectar falhas.

Tendo sido observadas falhas na execução do sistema, é, então, iniciado o processo de depuração.

3.2 Falhas observadas

Foram observadas as seguintes falhas:

- Mensagem de alerta com relação à chave utilizada na API Google *Maps*,
- Botão da tela de entrada do SADA,
- Erro no texto do botão em consultar histórico de mensagens e consultar histórico de operação de um talhão,
- Erro em “Consultar Posição de uma Máquina Agrícola”,
- Cálculo do consumo de combustível,
- Escala do gráfico de consumo de combustível e a unidade do gráfico,
- Escala do gráfico de velocidade,
- Consultar eficiência operacional,
- Campos obrigatórios,
- Consultar gráfico do nível de patinamento,
- Posição de uma máquina agrícola.

3.3 Abordagens utilizadas

Primeiramente, foi utilizada a abordagem de análise de programa. O SADA é composto por diversos arquivos javascript e arquivos de classes Java. O intuito dessa abordagem é entender o código escrito (através da leitura do código-fonte dos arquivos que fazem parte do SADA), a interação entre os arquivos (examinando as chamadas de funções e invocação de métodos) e verificar se está de acordo com as especificações do projeto.

As falhas geralmente corrigidas com essa abordagem estão relacionadas a erros de inicialização de classes ou variáveis e valores de variáveis.

Outra abordagem utilizada foi o rastreamento interno do programa. Essa abordagem foi usada à medida que as chamadas de funções ou as invocações de métodos entre os arquivos foram aumentando, tornado-se muito complexo a apenas a análise do código.

3.4 Técnicas utilizadas

Basicamente a única técnica utilizada foi o *tracing*, ou rastreamento. Essa técnica foi usada principalmente para encontrar os erros que provocavam a interrupção na execução do SADA.

O Eclipse oferece um depurador integrado no seu ambiente de desenvolvimento. A saída do depurador mostra o número da linha e o nome do arquivo na qual a execução de um programa foi interrompida, ou seja, faz o rastreamento do programa.

Com base nessa informação, é possível verificar o que está causando a falha. Conseqüentemente, projetar uma solução para a causa do problema e fazer as modificações necessárias.

Por fim, deve-se executar novamente o sistema para verificar se o defeito foi, de fato, solucionado.

4 RESULTADOS E DISCUSSÕES

Neste capítulo serão apresentados os erros corrigidos das falhas encontradas durante a execução do SADA.

4.1 Mensagem de erro da API Google ao mudar de uma tela para outra

Esse erro está relacionado a uma chave utilizada no Google Maps API. A figura 4.1 mostra essa situação.

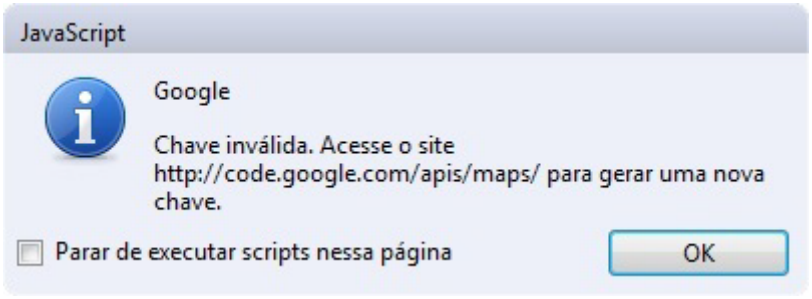

Figura 4.1 – Erro na chave da API Google *Maps*.

Quando o SADA foi desenvolvido a versão do Google Maps API era a versão 2. Esta exigia uma chave (*key*) na sintaxe de sua inicialização, como mostra o trecho de código da figura 4.2, que está no arquivo que serve de cabeçalho aos outros arquivos que geram as páginas *web* do SADA.

```
…
<script src="http://maps.google.com/maps?file=api&amp;v=2&amp;
            key = ABQIAAAApyNt0B0tOT3e4fNx1ogeART5GHwjocXozvB
                  Cb3ySKlpVvyBPkRSx6sInvM07Kis1Ffx2JxdEhMFGUg"
        type="text/javascript">
</script>
…
```

Figura 4.2 – Trecho de código contendo erro a chave API Google *Maps*.

Esse erro é corrigido ao se migrar para a versão 3, a qual não é mais exigida. Contribui para a migração da API o fato de que a Google Maps API versão 2 está

com previsão de remoção, conforme informação no site do Google Maps. O trecho anterior foi, então, alterado como mostra a figura 4.3:

```
…
<script src=" http://maps.google.com/maps/api/js?sensor=true"
        type="text/javascript">
</script>
…
```

Figura 4.3 – Trecho de código para versão 3 da API Google *Maps*.

4.2 Valor do botão na tela de entrada

O valor do botão, que é o texto contido nele, da tela de entrada do SADA estava como “Login”, conforme mostra a figura 4.4.

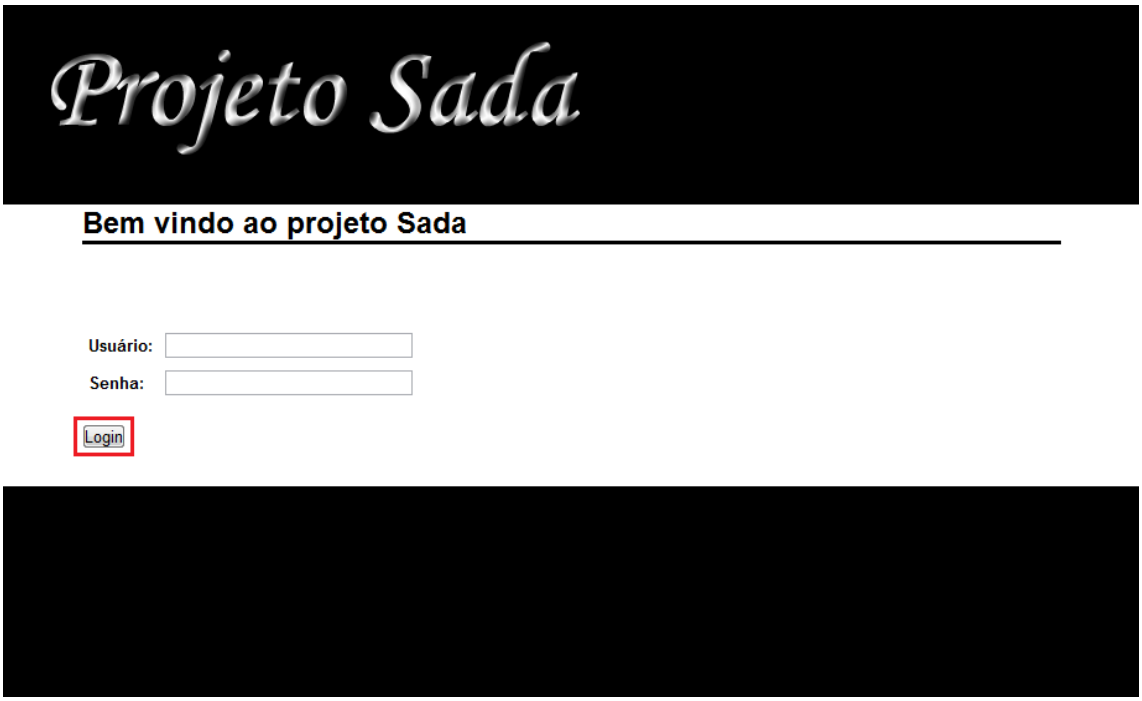

Figura 4.4 – Tela de entrada do com o valor do botão como Login.

O valor do botão foi alterado para Entrar, a figura 4.5 é a nova tela de entrada do SADA.

Figura 4.5 – Tela de entrada com o valor do botão alterado.

O valor do botão no código-fonte do arquivo referente à geração da tela de entrada estava conforme mostra a figura 4.6:

```
…
<button type=”submit”> Login </button>
…
```

Figura 4.6 – Trecho de código do antigo valor do botão da tela de entrada.

O trecho foi modificado para o mostrado na figura 4.7:

```
…
<button type=”submit”> Entrar </button>
…
```

Figura 4.7 – Trecho de código do antigo valor do botão da tela de entrada.

Assim, corrigiu-se o erro.

4.3 Em consultas, o botão está com o valor errado

O valor do botão se refere ao texto contido nele. Ao consultar o histórico de operações de um talhão ou o histórico de mensagens existe um botão que faz a submissão da consulta com os parâmetros desejados. O valor estava como “Salvar”, conforme figura 4.8.

Figura 4.8 – Erro no botão de submissão de consultas.

Porém para se realizar uma consulta o certo é “Enviar”, se deseja enviar os dados para uma consulta. Esse erro é de nomenclatura e não de funcionalidade. A correção foi feita, conforme a figura 4.9.

Figura 4.9 – Erro no botão de submissão de consultas corrigido.

Para isso, foram localizados os arquivos responsáveis a geração dessas telas de consultas e o trecho, figura 4.10, referente aos valores dos botões.

```
…
<button type=”submit”> Salvar </button>
…
```

Figura 4.10 – Trecho de código com o valor do botão como “Salvar”.

A correção do erro foi feita alterando o valor dos botões para Enviar, como mostrado a seguir pela figura 4.11.

```
…
<button type=”submit”> Enviar </button>
…
```

Figura 4.11 – Trecho de código com o valor do botão como “Salvar”.

4.4 Erro em “Consultar Posição de uma Máquina Agrícola”

Erro simples de ortografia referente à palavra agrícola que estava grafada como “agrícula” com u. A figura 4.12 mostra a tela dessa consulta.

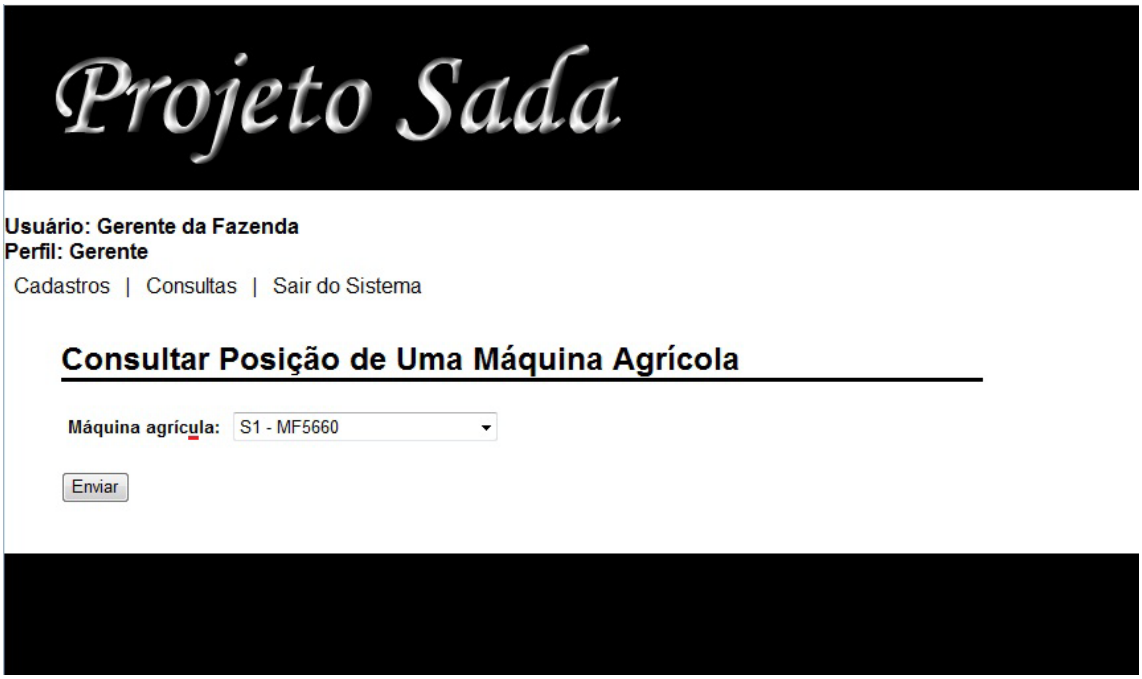

Figura 4.12 – Erro de ortografia

Localizado o arquivo que gera essa tela, encontrou-se o erro no código-fonte. Conforme mostrado na figura 4.13.

```
…
<table id=”editar-table>
     <tr>
        <td>
           <lable>M&aacute;quina agr&iacute;cula:</lable>
        </td>
        …
   </tr>
</table>
…
```

Figura 4.13 – Trecho de código contendo o erro de grafia.

Corrigiu-se a grafia da palavra, ou seja, foi trocado o “u” pelo “o”. O novo trecho de código é mostrado na figura 4.14.

```
…
<table id=”editar-table>
     <tr>
        <td>
           <lable>M&aacute;quina agr&iacute;cola:</lable>
        </td>
        …
   </tr>
</table>
…
```

Figura 4.14 – Trecho de código contendo o erro de grafia corrigido.

4.5 Cálculo do consumo de combustível

O consumo de combustível é calculado a partir da seguinte equação:

$$Cmc = \left(\frac{3{,}6}{t * n}\right) * \sum p$$

Onde:

C_{mc} = consumo médio de combustível no intervalos de tempo (l/h).

p = pulsos gerados pelo fluxômetro.

t = intervalo de tempo entre uma coleta de pulso e a próxima.

n = quantidade de leituras de pulsos dentro do intervalo de tempo sugerido.

Após encontrar o arquivo que realizava o cálculo, foi encontrado o erro no código, a seguir é mostrado na figura 4.15 o método chamado para realizar o cálculo.

```
…
public Float calcular(List<Integer> pulsos) {
    int soma = 0;
    for(Integer pulso : pulsos) {
         soma = soma + pulso;
    }
    return soma * ( (3.6f / tempoDeColeta) * pulsos.size());
}
…
```

Figura 4.15 – Trecho de código com erro em calcular consumo de combustível.

O cálculo do código acima pode ser representado pela equação a seguir.

$$Cmc = \sum p * \left(\frac{3{,}6}{t}\right) * n$$

Nota-se que *n*, o número de leitura, no código é o método pulsos.size(), está multiplicando ao invés de dividir. O código do cálculo foi alterado, corrigindo o erro. A figura 4.16 mostra como ficou o código.

```
…
public Float calcular(List<Integer> pulsos) {
    int soma = 0;
    for(Integer pulso : pulsos) {
         soma = soma + pulso;
    }
    return soma * (3.6f / (tempoDeColeta * pulsos.size()));
}
…
```

Figura 4.16 – Trecho de código do cálculo de consumo de combustível corrigido.

Os parênteses foram rearranjados entre a multiplicação da variável tempoDeColeta e o método pulsos.size(), dando prioridade a essa multiplicação, o resultado, então, divide 3,6. Finalmente, o resultado da divisão é multiplicado pela variável soma, que representa o somatório dos pulsos.

4.6 Escala do gráfico de consumo de combustível e sua unidade

A figura 4.17 mostra uma tela para a consulta do consumo de combustível.

Figura 4.17 – Visualizar o gráfico do consumo de combustível

Após selecionados os filtros, submete-se a consulta. A figura 4.18 representa o resultado que era gerado na consulta.

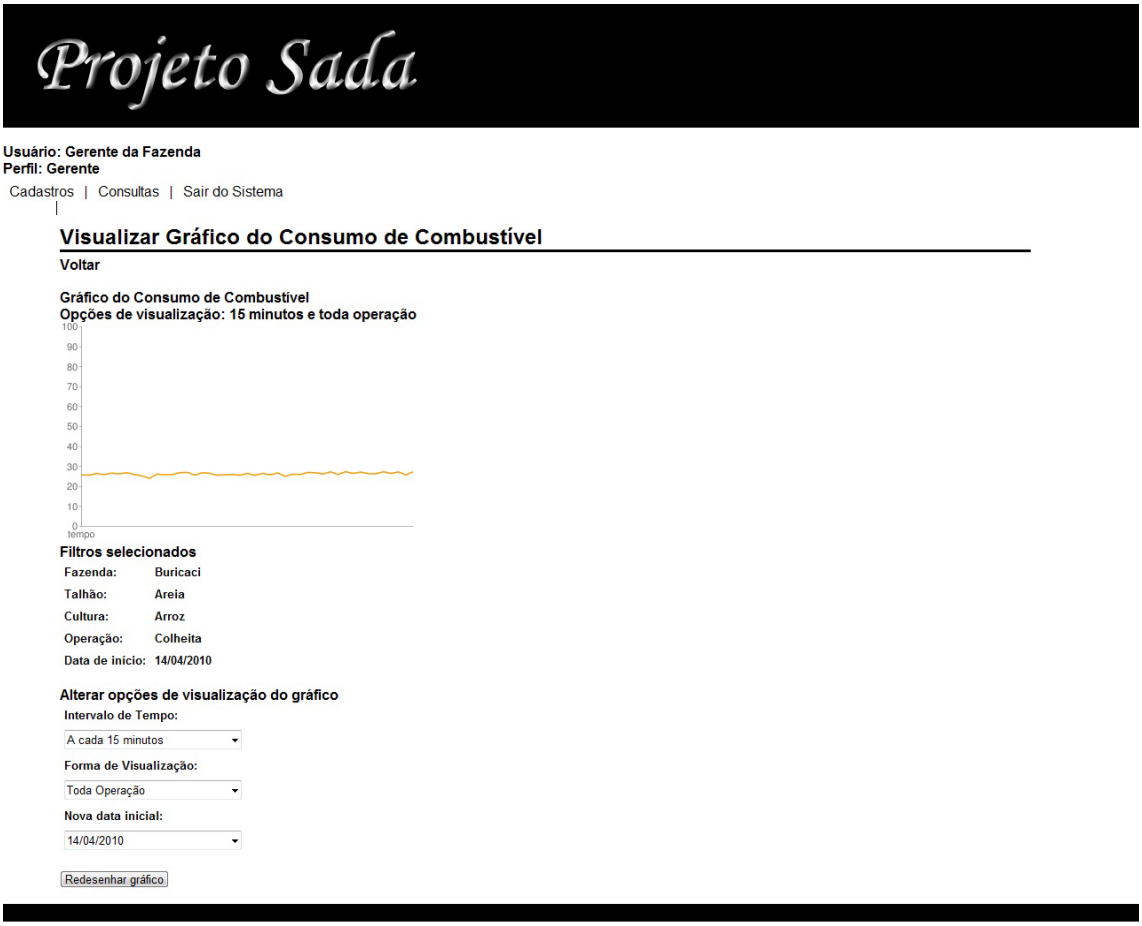

Figura 4.18 – Gráfico do consumo de combustível com erros.

Para a geração de gráficos no SADA é utilizada o Google *Chart* API. Por padrão os valores no eixo y variam de zero a cem. Contudo, o consumo de combustível de uma máquina agrícola pode chegar a 70 litros por hora. Nesse caso se quer mudar a escala do gráfico e também corrigir a unidade. Realizadas as alterações necessárias, a figura 4.19 representa a nova tela para o resultado das consultas ao gráfico do consumo de combustível.

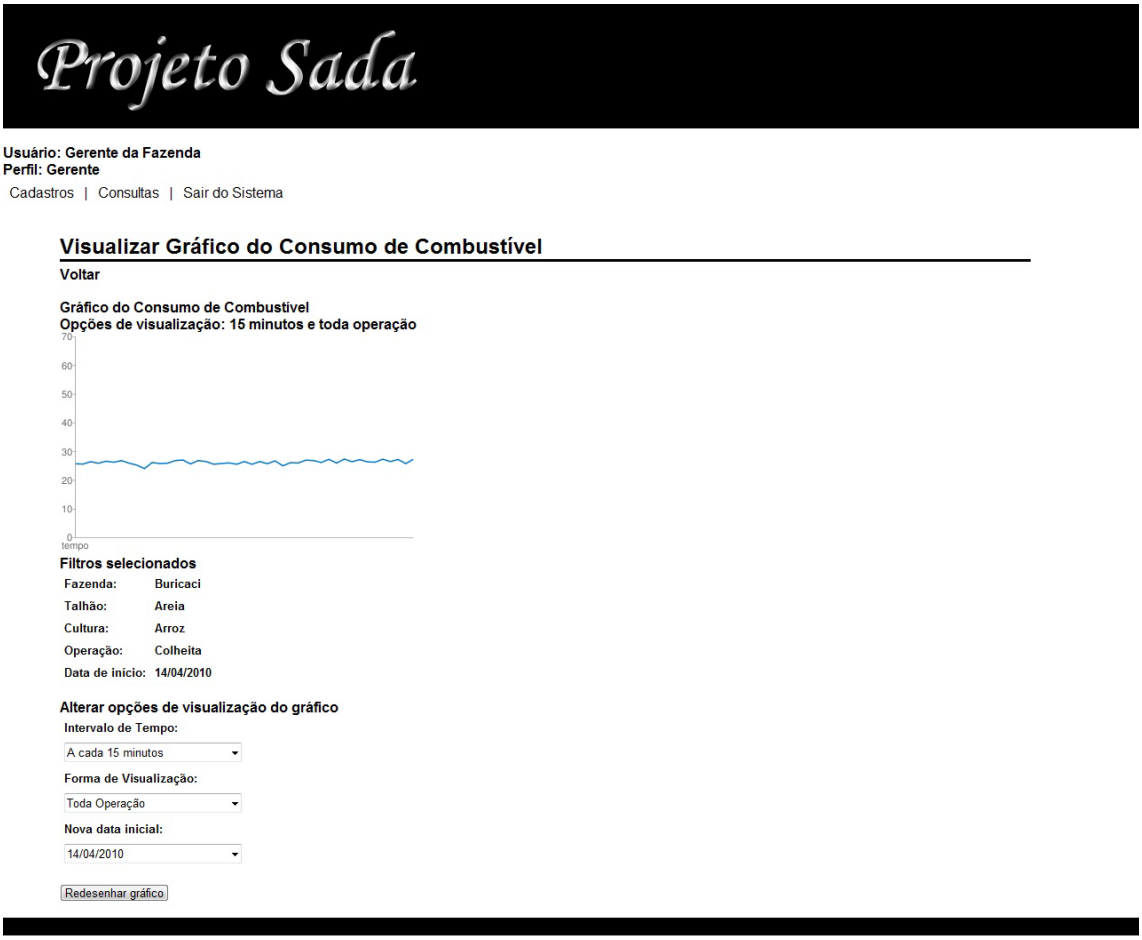

Figura 4.19 – Gráfico do consumo de combustível corrigido.

Para realizar as alterações, foi necessário identificar o arquivo que gera o resultado da consulta e identificar no código-fonte o trecho que gera o gráfico. A figura 4.20 mostra o trecho de código.

```
…
<img src="http://chart.apis.google.com/chart?
    chs=300x150&
    <c:choose>
         <c:when test="${operadorPontos.pontos[1] == null}">
              cht=bvs&
         </c:when>
         <c:otherwise>
              cht=lc&
         </c:otherwise>
    </c:choose>
    chd=t:
    <c:forEach items="${operadorPontos.pontos}" var="ponto">
         ${ponto}
    </c:forEach>&
    chxt=x,y
    chxl=0:|tempo"
/>
…
```

Figura 4.20 – Trecho de código com erro do gráfico de consumo de combustível.

A escala do mapa foi feita pela inclusão do parâmetro chds=0,70, onde 0 é o valor mínimo e 70 o valor máximo. Também foram alterados os valores do eixo y através do parâmetro chxr=1,0,70,10, onde o primeiro atributo de chxr é o eixo, x corresponde a 0 e y a 1, que se quer modificar, o segundo é o menor valor, o terceiro é o maior valor e o último é o intervalo que será mostrado os valores no eixo. Ainda foi editado o valor do parâmetro chxl=0:|tempo para chxl=0:|litros/hora, o primeiro atributo de chxl é referente ao eixo e o segundo o rótulo do eixo.

A figura 4.21 mostra o novo trecho que gera o gráfico.

```
…
<img src="http://chart.apis.google.com/chart?
    chs=300x150&
    <c:choose>
        <c:when test="${operadorPontos.pontos[1] == null}">
            cht=bvs&
        </c:when>
        <c:otherwise>
            cht=lc&
        </c:otherwise>
    </c:choose>
    chd=t:
    <c:forEach items="${operadorPontos.pontos}" var="ponto">
        ${ponto}
    </c:forEach>&
    chxt=x,y&
    chxl=0:|litros/hora&
    chco=0077CC&
    chds=0,70&
    chxr=1,0,70,10”
/>
…
```

Figura 4.21 – Trecho de código do gráfico de consumo de combustível corrigido.

4.7 Escala do gráfico de velocidade

A funcionalidade de consulta sobre a velocidade de uma máquina agrícola é mostrada na figura 4.22.

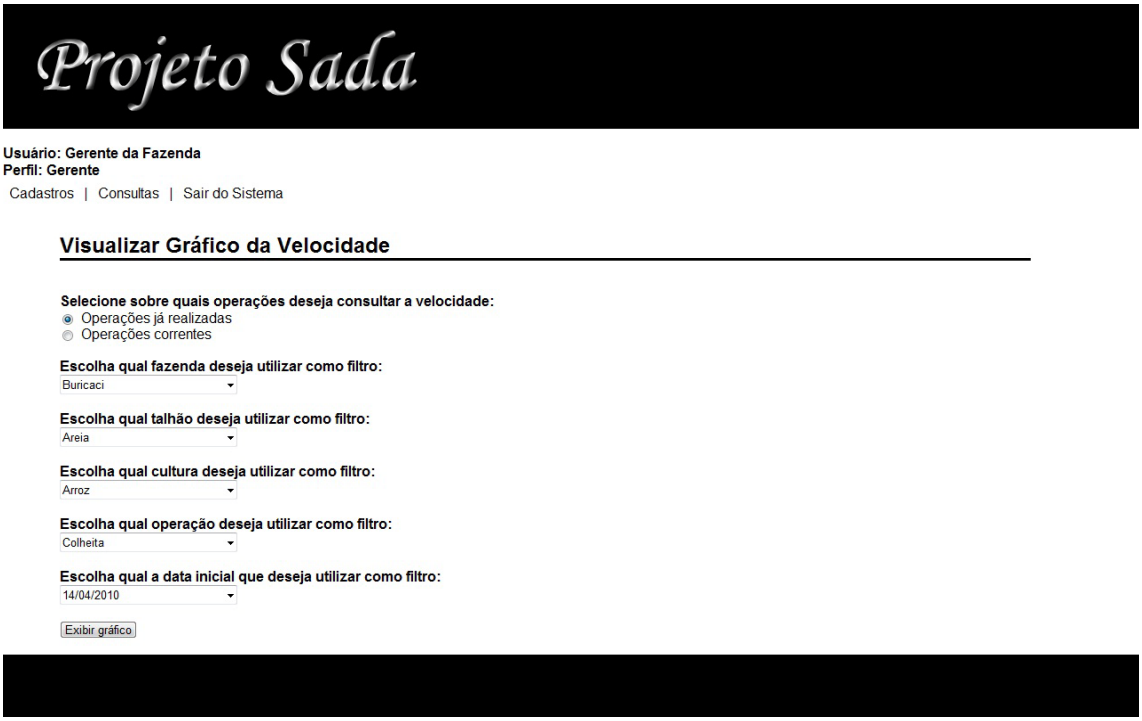

Figura 4.22 – Visualizar gráfico da velocidade.

Depois de selecionar os filtros e submeter a consulta, a figura 4.23 representa o resultado que estava sendo gerado.

Figura 4.23 – Gráfico da velocidade sem escala.

Também é utilizada a API Google *Chart* para a geração do gráfico. Como visto na subseção 4.6, a escala no eixo y varia de zero a cem. Porém, uma máquina agrícola dificilmente chega a uma velocidade 20 km/h. Nesse caso, é ainda mais importante que a escala do mapa seja ajustada, para se ter uma melhor visualização. Depois de ter feito as alterações necessárias, a figura 4.24 é o novo resultado para a consulta da visualização do gráfico da velocidade.

Figura 4.24 – Gráfico da velocidade com escala.

Para realizar as alterações, primeiro teve que localizar o arquivo que gera a tela do resultado da consulta, após identificar o trecho no código-fonte do arquivo que gera o gráfico. O trecho é mostrado na figura 4.25.

```
…
<img src="http://chart.apis.google.com/chart?
    chs=400x250&
    <c:choose>
        <c:when test="${operadorPontos.pontos[1] == null}">
            cht=bvs&
        </c:when>
        <c:otherwise>
            cht=lc&
        </c:otherwise>
    </c:choose>
    chd=t:
    <c:forEach items="${operadorPontos.pontos}" var="ponto">
        ${ponto}
    </c:forEach>&
    chxt=x,y&
    chxl=0:|Tempo&
    chco=0077CC"
/>
…
```

Figura 4.25 – Trecho de código com erro no gráfico da velocidade.

A escala do gráfico foi alterada para variar de 0 a 20, incluindo o parâmetro chds=0,20; esse parâmetro não altera os valores do eixo y, para isso foi incluído o atributo chxr=1,0,20,5. O primeiro atributo de chxr é o eixo, x corresponde a 0 e y a 1, que se quer modificar, o segundo é o menor valor, o terceiro é o maior valor e o último é o intervalo que será mostrado os valores no eixo. Como mostra o trecho da figura 4.26.

```
…
<img src="http://chart.apis.google.com/chart?
    chs=400x250&
    <c:choose>
        <c:when test="${operadorPontos.pontos[1] == null}">
            cht=bvs&
        </c:when>
        <c:otherwise>
            cht=lc&
        </c:otherwise>
    </c:choose>
    chd=t:
    <c:forEach items="${operadorPontos.pontos}" var="ponto">
        ${ponto}
    </c:forEach>&
    chxt=x,y&
    chxl=0:|Tempo&
    chco=0077CC&
    chds=0,20&
    chxr=1,0,20,5”
/>
…
```

Figura 4.26 – Trecho de código do gráfico da velocidade corrigido.

4.8 Consultar eficiência operacional

O retorno da consulta sobre a eficiência operacional de uma operação em uma cultura é um valor percentual. Ao invés disso, estava sendo gerado um gráfico como retorno dessa consulta, como mostra a figura 4.27.

Figura 4.27 – Antiga tela com o resultado da consulta a eficiência operacional

A correção foi a remoção do trecho de código responsável pela geração do gráfico e também se removeu a parte em que podem ser feitas alterações no gráfico. Na nova tela do resultado da consulta, é mostrado primeiramente os filtros selecionados e após e valor da eficiência operacional (figura 4.28).

Figura 4.28 – Nova tela do resultado da consulta da eficiência operacional.

O novo código é mostrado pela figura 4.29.

```
<label>Filtros selecionados</label>
  <table id="editar-table">
     <tr>
       <td><label>Tipo de opera&ccedil;&otilde;es:</label></td>
          <td><label>${tipoOperacao}</label></td>
     </tr>
     <tr>
        <td><label>Fazenda:</label></td>
        <td><label>${fazenda.nome}</label></td>
     </tr>
     <tr>
        <td><label>Talh&atilde;o:</label></td>
        <td><label>${talhao.nome}</label></td>
     </tr>
     <tr>
        <td><label>Cultura:</label></td>
        <td><label>${cultura.nome}</label></td>
     </tr>
     <tr>
        <td><label>Opera&ccedil;&atilde;o:</label></td>
        <td><label>${operacao.nome}</label></td>
     </tr>
     <tr>
        <td><label>Data de in&iacute;cio:</label></td>
        <td><label>${data}</label></td>
     </tr>
  </table>
  <br>
  <label>Valor da efici&ecirc;ncia operacional: ${ponto}%</label>
```

Figura 4.29 – Novo trecho de código de eficiência operacional.

4.9 Campos obrigatórios

Ao se realizar a inclusão de fazendas, talhões, culturas, etc, na opção de cadastros, todos os campos são obrigatórios. Esta verificação não estava sendo realizada.

A correção desse erro foi a adição, nos arquivos responsáveis por essas funções de inclusão, um trecho de código que verifica a presença da informação, caso não esteja presente é emitido um alerta informando que o campo é obrigatório.

4.10 Consultar gráfico do nível de patinamento

Estava dando erro, pois os valores são nulos. O erro foi corrigido, inserindo um trecho de código que faz a verificação se os valores são nulos, e um alerta está sendo mostrado informando que os campos são nulos (figura 4.30). Após isso, pode-se voltar para realizar nova consulta selecionando outros filtros.

Figura 4.30 – Mensagem de valores nulos para gráfico do nível de patinamento.

Para isso, depois de localizado o arquivo que processa a consulta e identificado o trecho em que ocorria o erro em função dos valores serem nulos, acrescentou-se um trecho de código que faz um desvio no fluxo de execução através de uma verificação dos valores dos pontos. Uma variável de verificação foi criada, sendo atribuído o valor um a ela. Se os pontos forem nulos é atribuído o valor zero a variável de verificação. A variável é, então, testada e se seu valor for zero

uma mensagem de alerta é enviada, se não, é gerado o gráfico. O trecho é mostrado na figura 4.31.

```
…
<c:set var="flag" value="1" />
<c:forEach items="${listaOperadorPontos}" var="operaorPonto">
    <c:if test="${operaorPonto == 'NaN' || ponto == 'NaN,'}">
         <c:set var="flag" value="0" />
    </c:if>
</c:forEach>
<c:choose>
    <c:when test="${flag == 0}">
         <script type="text/javascript">
              alert("Valores Nulos");
         </script>
    </c:when>
    <c:otherwise>
         … (trecho de geração do gráfico)
    </c:otherwise>
</c:choose>
…
```

Figura 4.31 – Trecho de código com o erro do nível patinamento corrigdo.

4.11 Posição de uma máquina agrícola

A figura 4.32 mostra a consulta da posição de uma máquina agrícola. Essa consulta permite visualizar um mapa indicando a posição de uma máquina agrícola. O filtro para essa consulta é apena a máquina que se quer saber a posição.

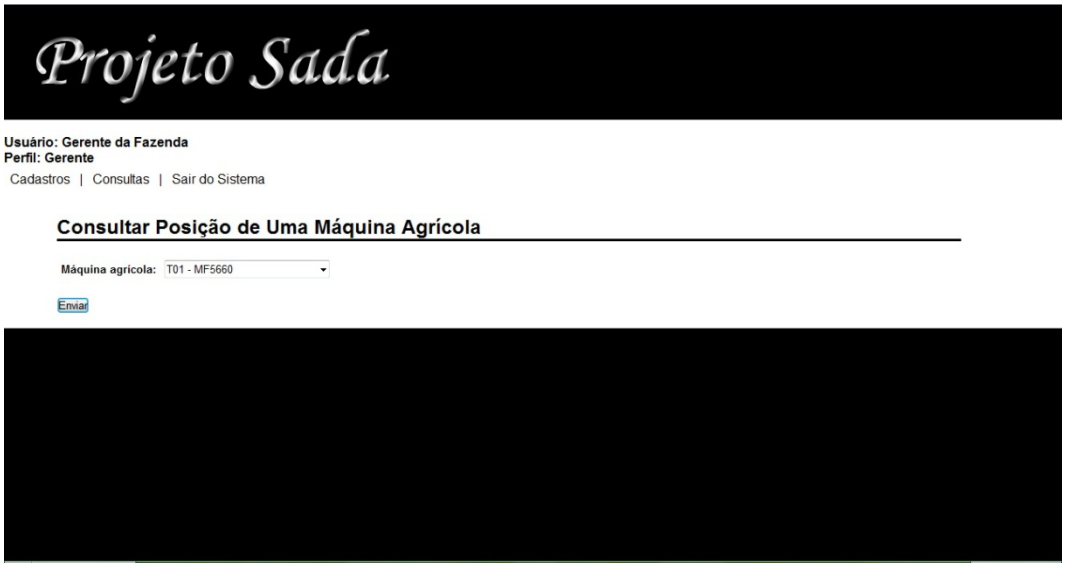

Figura 4.32 – Consultar posição de uma máquina agrícola

Quando submetida a consulta, acontecia uma falha no sistema. Foi localizado o arquivo que processava a consulta e nele o método responsável pela geração do mapa. O método (*loadGMaps()*) foi totalmente refeito. A seguir, figura 4.33, é mostrada a tela gerada a partir do novo método.

Figura 4.33 – Resultado da consulta da posição de uma máquina agrícola

O novo método é mostrado na figura 4.34.

```
<script type="text/javascript">
function loadGMaps(latitude, longitude) {
  var initialLocation = new google.maps.LatLng(latitude,
                                                longitude);
  var myOptions = {
     zoom: 16,
     mapTypeId: google.maps.MapTypeId.SATELLITE
  }
  var map = new google.maps.Map(document.getElementById("mapa-
                                 maquina"), myOptions);
  map.setCenter(initialLocation);
  var marker = new google.maps.Marker;
  marker.setMap(map);
  marker.setTitle("Máquina");
  marker.setPosition(initialLocation);
  var contentString = '<div id="content">'+
        '<div id="siteNotice">'+
        '</div>'+
        '<div id="bodyContent">'+
        '<p>Fazenda: ' +
        '{historico.culturasOperacoesTalhoes.talhao.fazenda.nome}
          </p>' +
       '<p>Talhão: '+'
        ${historico.culturasOperacoesTalhoes.talhao.nome} </p>'+
       '<p>Nome operador: '+
       '${historico.culturasOperacoesTalhoes.culturasOperacoes
        TalhoesPK.idTalhao}
        </p>'+
       '<p>Contato operador: '+'</p>'+
       '</div>';
  var infowindow = new google.maps.InfoWindow(
                              {content: contentString} );
  google.maps.event.addListener(marker, 'click',
                                 function() {
                                   infowindow.open(map,marker);
                                 });
}
</script>
```

Figura 4.34 – Novo código do método *loadGMaps()*.

4.12 Considerações Finais

Durante a depuração do SADA, foi constatada a complexidade de encontrar os locais defeituosos em um programa. A dificuldade na localização dos erros foi variável e, conseqüentemente, tiveram diferentes tempos para serem descobertos. Depois, projetar a solução e implementar foi mais simples.

As falhas apresentadas nas seções 4.2, 4.3 e 4.4 foram as que tiveram os seus locais defeituosos encontrados mais rapidamente, isso aconteceu enquanto o código-fonte dos arquivos era estudado, sendo corrigidos logo.

Também durante a análise do código-fonte, os locais que continham os erros das falhas mostradas nas seções 4.1, 4.6 e 4.7 foram encontrados, porém antes de projetar as correções foi necessário o estudo das APIs Google *Chart* e Google *Maps*.

As demais falhas, das seções 4.5, 4.8, 4.9, 4.10 e 4.11, tiveram os locais dos erros encontrados com o auxílio do depurador presente no Eclipse. A saída do depurador mostrava a o número da linha e o arquivo na qual estava a execução do SADA no momento da falha. Através dessa informação foi analisado o código anterior e o da linha, observado as variáveis e chamadas de funções ou invocação de métodos, assim descobrindo os locais dos erros. A correção desses erros envolveu conhecimento lógica de programação e das linguagens de programação Java e JavaScript, além das já citadas APIs Google.

Por ter sido um sistema implementado por outro programador, uma dificuldade presente foi a de entender o código-fonte dos arquivos do SADA e a interação entre as classes. A aplicação da abordagem da análise do programa ajudou a resolver esse problema.

Uma combinação entre abordagens e técnica de depuração foi utilizada na correção dos defeitos encontrados no SADA. As duas abordagens usadas, baseada na análise de programas e a baseada no rastreamento interno do programa, têm enfoques diferentes, a análise do sistema e a analise do código juntamente com exemplos de saída, respectivamente. Por fim, a utilização da técnica de *tracing* para auxiliar a segunda abordagem.

5 CONCLUSÃO

Este trabalho realizou um estudo sobre a depuração, apresentando a sua definição, abordagens, técnicas e ferramentas utilizadas durante o processo de depuração. Foram mostradas algumas dificuldades que podem ser encontradas durante a execução dessa tarefa, que variam dependendo do tipo de sistema em desenvolvimento, mas também por ser uma área pouco desenvolvida em engenharia de *software*.

Ainda neste trabalho, foi proposta a depuração do SADA, aplicativo concebido a partir de um projeto entre o curso de Ciência da Computação e o PPGEA. Antes disso, foi necessário um estudo acerca do sistema desenvolvido e das APIs utilizadas, como Google *Chart* e Google *Maps*. Após, então, foram observadas as falhas contidas no SADA para iniciar a depuração.

A importância deste trabalho está em garantir que as funções do SADA estejam de acordo com as especificações do projeto e, também, que elas funcionem corretamente, corrigindo os defeitos observados. Sem as correções efetuadas no sistema, o mesmo não tinha condições nem sequer de ser demonstrado para o usuário, pois continha erros já na sua tela de abertura.

Para realizar o processo de depuração do SADA foram utilizadas duas abordagens. A primeira foi a baseada na analise do programa, na qual o código-fonte do programa é lido, analisado e verificado se o mesmo era compatível com as especificações do projeto. A outra abordagem foi a baseada no rastreamento interno do programa, que é realizada através do código-fonte e exemplos de saída da execução. Também foi utilizada a técnica de rastreamento, ou *tracing*, essa técnica rastreia um programa durante a execução e a inspeção quando ocorre um evento, no caso uma falha. Essa técnica é auxiliada pelo uso de depuradores.

Observa-se que a forma que a atividade de depuração é executada depende do sistema que se está depurando e de quem a realiza. Para sistemas de grande porte pode não ser eficiente o uso de certas abordagens ou técnicas, como a técnica de *backtracking*. O programador deve decidir qual abordagem, técnica ou uma combinação entre elas, como foi feito neste trabalho, para utilizar no processo de depuração. Essa escolha pode ser influenciada pelo conhecimento e pelas aptidões do programador.

Ao final, pode-se concluir que foi atingido o objetivo da depuração do SADA para as falhas observadas. Contudo, é importante ressaltar que com a perspectiva de acréscimo de novas funcionalidades ao sistema, pode ser necessário realizar novamente a depuração do sistema. Assim, esse trabalho pode ser material de consulta a trabalhos futuros.

REFERÊNCIAS BIBLIOGRÁFICAS

AGRAWAL, H.; DEMILLO, R. A.; SPAFFORD, E. H. **An Execution-Backtracking Approach to Debugging.** IEEE Software. v8, p. 21-26, 1991.

ARAKI, K.; FURUKAWA, Z.; CHENG, J. **A general framework for debugging.** IEEE Software. p. 14-20. Maio, 1991.

BEIZER, B. **Software Testing Techniques.** Nova York: Van Nostand Reinhold, 1990.

CHAIM, M. L. **Depuração de Programa Baseado em Informação de Teste Estrutural.** 2001. 243 f. Tese (Doutourado em Engenharia Elétrica) – Universidade Estadual de Campinas, Campinas, 2001.

CHEUNG, W. H.; BLACK, J. P.; MANNING, E. **A Framework for Distribuited Debugging.** IEEE Software. p. 106-115. Janeiro, 1990.

CRUZES, D. S. **Visualização de Informações para Suporte ao Teste de Depuração de Programas.** 1999. 121 f. Dissertação (Mestrado em Engenharia Elétrica e Computação) – Universidade Estadual de Campinas, Campinas, 1999.

DELAMARO, E. D.; MALDONADO, J. C.; JINO, M. **Introdução ao Teste de Software.** Rio de Janeiro: Elsevier, 2007. 396 p.

ECLIPSE. **Standard JET2 Control Tags.** Disponível em: <http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.jet.doc/references/taglibs/controlTags/chooseTag.html>. Acesso em: 23 nov. 2010.

ECLIPSE**. The Eclipse Foundation open source community website.** Disponível em: <www.eclipse.org/>. Acesso em: 15 out. 2010.

GDB. **GDB: The GNU Project Debugger.** Disponível em: <http://www.gnu.org/software/gdb/gdb.html>. Acesso em: 18 nov. 2010.

GOOGLE CODE. **Getting Started With Charts.** Disponível em: <http://code.google.com/intl/pt-BR/apis/chart/docs/making_charts.html>. Acesso em: 05 out. 2010.

GOOGLE CODE. **Google Maps Javascript API V3 Reference.** <http://code.google.com/intl/pt-BR/apis/maps/documentation/javascript/reference.html>. Acesso em: 15 set. 2010.

GOOGLE CODE. **Google Maps JavaScript API V3.** Disponível em: <http://code.google.com/intl/pt-BR/apis/maps/documentation/javascript/>. Acesso em: 13 set. 2010.

IBM. **Rational Purify.** Disponível em: <http://www-01.ibm.com/software/awdtools/purify/>. Acesso em: 12 nov. 2010.

JSWAT. **jswat - JSwat Java Debugger.** Disponível em: <http://code.google.com/p/jswat/>. Acesso em: 18 nov. 2010.

LIAN, L. et al. **A new fault localizing method for the program debugging process.** Information and Software Technology, v.39, p 271-284. 1997.

MYERS, G. J. **Software Reliability: Principles and Practices.** Nova York: Willly-Interscience, 1976. 360 p.

ORACLE. **jdb - The Java Debugger.** Disponível em: <http://download.oracle.com/javase/1.3/docs/tooldocs/solaris/jdb.html>. Acesso em: 18 nov. 2010.

PRESSMAN, R. S. **Engenharia de Software.** São Paulo: Pearson Makron Books. 2006. 1056 p.

SANTOS, F. et al. **Sistema de Telemetria Redundante e Tolerante a Talhas Utilizando Tecnologia GSM/GPRS e ZigBee.** Conferência IADIS Ibero-Americana, Algarve, Portugual, 2010.

SANTOS, F. **T-SADA - Sistema de Telemetria Redundante e Tolerante a Falhas Utilizando Tecnologia GSM/GPRS e ZigBee.** 2010. Dissertação - (Mestrado em Informática) Universidade Federal de Santa Maria, Santa Maria, 2010.

SEVIORA, R. E. **Knowledge-Based Program Debugging System.** IEEE Software, v.4, p 20-32. Maio, 1987.

SICHONANY, O. **Uso de telemetria para transmissão de dados de desempenho de máquinas agrícolas visando o gerenciamento.** 2010. Exame

de Qualificação - (Doutourado em Engenharia Agrícola) Universidade Federal de Santa Maria, Santa Maria, 2010.

SOMMERVILLE, I. **Engenharia de Software.** São Paulo: Pearson Addison Wesley, 2005. 592 p.

STAA, A. V. **Programação Modular.** Rio de Janeiro: Editora Campus, 2000. 690 p.

THE APACHE SOFTWARE FOUNDATION. **Apache Tomcat.** Disponível em: <http://tomcat.apache.org/>. Acesso em: 18 nov. 2010.

VIRAVAN, C. **Enhancing Debugging Technology.** 1994. 192 f. Tese (Doutourado) Pardue University, West Lefayette, 1994.

WIKIPEDIA. **Debugger.** Disponível em: <http://en.wikipedia.org/wiki/Debugger>. Acesso em: 10 nov. 2010.