Universidade de Évora

Escola de Ciências e Tecnologia

Mestrado em Engenharia Informática

Interface de instrumentos para veículo eléctrico Gecko Merula – Sistema de navegação por Satélite

Miguel Ângelo da Silva Gualdino

Orientador
Prof. Luís Miguel de Mendonça Rato

"Esta dissertação não inclui as críticas e sugestões feitas pelo Júri"

2011

Mestrado em Engenharia Informática

Interface de instrumentos para veículo eléctrico Gecko Merula – Sistema de navegação por Satélite

Miguel Ângelo da Silva Gualdino

Orientador
Prof. Luís Miguel de Mendonça Rato

"Esta dissertação não inclui as críticas
e sugestões feitas pelo Júri"

Aos meus pais
Ao meu irmão
Aos meus avós
Aos meus amigos

Sumário

O objetivo deste trabalho é o desenvolvimento de um painel de instrumentos para o veículo elétrico *Gecko Merula*. Para o seu desenvolvimento foi feito um levantamento de princípios para definição do painel de instrumentos. O painel de instrumentos foi desenvolvido na plataforma *Java* para definição dos instrumentos presentes no painel de instrumentos. Foi desenvolvido um ambiente de desenvolvimento para criação de interfaces gráficas definidas no documento *XML*. De forma a complementar o painel de instrumentos foi desenvolvida um sistema de navegação por satélite que faz uso da localização do veículo para cálculo de itinerários para um determinado destino com base em mapas *online*. As interfaces propostas foram avaliadas pelos princípios de apresentação de informação em veículos e pelo tempo de desenho, memória consumida da interface do painel de instrumentos e do sistema de navegação. Com base nos resultados obtidos conclui-se que o painel de instrumentos não deverá conter demasiados componentes pois prejudicará a atenção do condutor e eventualmente a velocidade de atualização dos valores apresentados no painel. O painel de instrumentos proposto permite definir e alterar os vários componentes gráficos no documento *XML* de forma flexível.

Gecko Merula electric vehicle dashboard - Satellite navigation system

Abstract

The objective of this work is the development of an instrument panel for the electric vehicle *Gecko Merula*. A survey of principles for setting the instrument panel was done for its development. The instrument panel was developed on the *Java* platform. It was implemented a development environment to create a graphical user interfaces specified in the *XML* document. In order to complement the instrument panel a satellite navigation system has been developed. This system makes use of the vehicle's location to calculate itineraries to a particular destination based on online maps. The proposed interface was evaluated by: principles of presentation of information in vehicles; and drawing time; memory consumption of the interface of the instrument panel and navigation system. Based on the achieved results it was concluded that the instrument panel should not include too many components because it may prejudice the driver's attention and the accuracy of the values presented on the panel. This proposed instrument panel allows the designers to set and change the various graphical components on the *XML* document in a flexible way.

Agradecimentos

Agradeço aos pais e irmão por ter chegado a esta etapa sem eles isto tudo não teria sido possível.

O acompanhamento e aconselhamento da dissertação por parte do orientador e professor Luís Rato foi determinante para conclusão da dissertação, para ele o meu muito obrigado pela ajuda e paciência prestada.

E um especial agradecimento ao meu colega e amigo Paulo Amaral pela entreajuda que prestamos para conclusão das nossas dissertações.

E todos os restantes amigos diretamente ou indiretamente me ajudam à conclusão da dissertação.

Acrónimos

PDI Painel de instrumentos

HUD *Head up display*

CPC Carro para carro

SNS Sistema de navegação por satélite

SMS *short message service*

SWT *Standard Widget Toolkit*

XML *Extensible Markup Language*

RACS *Road/Automobile Communication System*

QNX *Realtime Operating System*

API *Application programming interface*

GPS *Global Positioning System*

TCP *Transmission Control Protocol*

IP *Internet Protocol address*

RPM Rotações por minuto

Sapo Mapas Sapo Mapas

Conteúdo

Sumário **i**

Abstract **iii**

Agradecimentos **v**

Acrónimos **vii**

1 Introdução **1**

1.1 Introdução 1

1.2 Objetivos 2

1.3 Principais Contribuições 2

1.4 Estrutura da dissertação 3

2 Estado da Arte **5**

2.1 Painel de instrumentos 5

2.1.1 Estrutura comum do painel de instrumentos 6

2.2 Desenho de um painel de instrumentos 7

2.2.1 Desvantagens e vantagens em PDI's digitais 8

2.2.2 Princípios gerais para criação de um **PDI**'s 8

2.2.3 Exemplos de painéis 11

2.3 Sistema de navegação 16

2.3.1 Princípios 17

2.3.2 Exemplos 17

3 Proposta para um painel de Instrumentos **21**

3.1 Visão geral 21
3.2 Arquitetura 22
3.2.1 Arquitetura global do sistema 22
3.2.2 Arquitetura do módulo da interface gráfica 23
3.2.3 Constituição do documento *XML* do PDI 24
3.3 Implementação 27
3.3.1 Tecnologias usadas 27
3.3.2 Processamento do documento *XML* 28
3.3.3 Separação dos componentes *refreshable* dos estáticos 29
3.3.4 Algoritmo de desenho dos componentes *refreshables* e estáticos 30
3.3.5 Descrição do algoritmo de desenho de cada componente do PDI 31
3.4 Ambiente de desenvolvimento 36
3.4.1 Exemplo de criação de um componente para um PDI 37

4 Proposta para um sistema de navegação 41
4.1 Visão geral do sistema 41
4.2 Arquitetura da aplicação 42
4.2.1 Arquitetura global do sistema 42
4.2.2 Arquitetura da interface gráfica do SNS 43
4.3 Implementação 43
4.3.1 Tecnologias usadas 43
4.3.2 Embeber *SWT Browser Widget* numa aplicação *Java Swing* . 45
4.3.3 Inicialização do sapo mapas com a localização atual 46
4.3.4 Descrição dos painéis da interface SNS 47
4.4 Tutorial para criação de um *browser widget* numa aplicação *Java Desktop* em *Linux* 49

5 Casos de estudo 53
5.1 Constituição do painel de instrumentos 53
5.2 A interface do painel de instrumentos 54
5.3 Interface do sistema de navegação 56
5.4 Apreciação das informações apresentadas 56

6 Conclusões 61
6.1 Conclusões 61
6.2 Principais contributos 63

6.3 Trabalho futuro 63

Bibliografia **69**

A Documento XML da interface do painel **73**

Lista de Figuras

2.1 *Head-up display* presente no avião de guerra. 11
2.2 *Head-up display* no carro. 12
2.3 *BMW Head-up display*. 12
2.4 *Ford PDI*. 13
2.5 *Lamborghini PDI*. 13
2.6 Arquitectura do Software *QNX*. 14
2.7 Exemplo de um painel desenvolvido com *QNX*. 15
2.8 Vista por defeito do PDI do veículo **VEIL**. 16
2.9 Arquitectura do sistema **RACS**. 18
2.10 Painel de navegação **RACS**. 18
2.11 Interface do sistema de navegação por satélite *Roadnav*. 19
2.12 Interface do sistema de navegação por satélite *Gpsdrive*. 19
2.13 Interface do sistema de navegação por satélite *Navit*. 20

3.1 Arquitetura global do sistema 22
3.2 Arquitetura do módulo da interface gráfica 23
3.3 Indicador de um veículo que utiliza um medidor analógico com ponteiro. 25
3.4 Componente de um PDI com vários patamares no indicador de carga da bateria. 26
3.5 Indicador de porta aberta do veículo no estado ligado. 26
3.6 Indicador de porta aberta do veículo no estado desligado. 26
3.7 Exemplo de aplicação da técnica *double buffering* 31
3.8 distâncias $\boldsymbol{a}$ e $\boldsymbol{b}$ da origem aos pontos *p1* e *p2*. 33
3.9 Constituição do componente *levelGauge*. 34

3.10 Esquema de atualização do componente *label*. 36
3.11 Indicador de porta aberta do veículo no estado desligado. 37
3.12 Indicador de porta aberta do veículo no estado ligado. 38

4.1 Arquitetura global do sistema de navegação por satélite 42
4.2 Esquema da interface do sistema de navegação 43
4.3 Sapo mapas na coordenada **39.068461, -8.466054** 46

5.1 Painel de instrumentos para veículo elétrico *Gecko Merula* 54
5.2 Tabelas de dados obtidos do desempenho da interface. 55
5.3 Gráfico de desempenho da interface. 55
5.4 Gráfico de memória consumida. 56
5.5 Interface do sistema de navegação por satélite. 57

Capítulo 1

Introdução

1.1 Introdução

Nos dias hoje a condução de veículos faz parte do nosso quotidiano o "manter os olhos na estrada"é uma das prioridades dos instrutores de condução na hora de passar o testemunho aos alunos. No entanto, frequentemente isso não acontece, desviamos muitas vezes os olhos da estrada colocando em risco a nossa vida e a dos outros, com por exemplo a escrita de um **SMS**[1] ou uma mudança de estação de rádio. Atualmente no interior do veículo temos diversos painéis que nos ajudam na condução, sendo um dos mais importantes o painel instrumentos [Edmonds, 2009] e o uso recente de sistemas de navegação por satélite.

A função principal do painel de instrumentos é fornecer informações sobre veículo ao condutor com o mínimo de interferências com a condução do veículo. Os componentes presentes no painel de instrumentos não devem ser causa de comportamentos de risco rodoviária. Assim a definição da interface para o painel de instrumentos deve obedecer a uma série de princípios para o seu desenvolvimento definidos por:

- O condutor de um veículo deve ser capaz de interpretar as informações provenientes de forma precisa, correta e atempada. Por isso o painel de instrumentos deve transmitir confiança ao condutor durante a sua consulta.
- O painel de instrumentos do veículo elétrico *Gecko Merula* deve cativar o interesse das pessoas. Esse interesse provém da polivalência e conjunto de princípios que regeram a criação do painel de instrumentos e a inclusão de um

[1] *Short Message Service*

sistema de navegação por satélite.

1.2 Objetivos

O objetivo principal desta dissertação é a criação de um sistema para definição de painéis de instrumentos para o veículo elétrico *Gecko Merula*. Para a definição de um painel seguro e de confiança, os instrumentos informativos serão regidos por um conjunto de princípios para a sua apresentação, na qual a interface será avaliada.

O painel de instrumentos deverá ser um sistema fácil e ágil a alterações que possam ser efetuadas. Essa agilidade deve incidir no funcionamento do painel, mas não na disposição dos componentes do painel. Para essa separação de contextos deve existir uma abstração da parte gráfica da interface do painel e uma abstração da definição e programação do funcionamento dos componentes do painel. As duas abstrações serão geridas por duas equipas: uma responsável pelo desenho gráfico (*designers*) de componentes gráficos e painel, e uma equipa de programadores responsáveis pelo funcionamento dos mesmos elementos gráficos do painel de instrumentos. Resumindo será o objetivo desta dissertação e a criação de um ambiente de desenvolvimento de painéis de instrumentos que permita personalização do tema gráfico do painel pelos *designers* sem interferir com trabalho funcional dos programadores nas ações dos componentes visíveis no painel.

Será também um outro objetivo desta dissertação o estudo e proposta de um sistema de navegação por satélite para o painel de instrumentos para o *Gecko Merula*.

A definição e proposta do painel de instrumentos do *Gecko Merula* e o ambiente de desenvolvimento foi elaborada em conjunto com o meu colega e amigo **Paulo Amaral** e com o engenheiro do veículo elétrico *Gecko Merula*, **Ricardo Melro**.

1.3 Principais Contribuições

As principais contribuições desta dissertação são:

- Definição de um ambiente de desenvolvimento para interfaces de painéis de instrumentos de veículos.
- Integração de um sistema navegação por satélite para o veículo elétrico *Gecko Merula*.
- Avaliação de interfaces à luz dos princípios de apresentação de informação.

1.4 Estrutura da dissertação

Esta dissertação está dividida em 6 capítulos.

O primeiro capítulo é a –**Introdução**– tem o enquadramento e contextualização da dissertação e descrição dos objetivos propostos.

O segundo capítulo é –**Estado da Arte**–, formado pelo levantamento bibliográfico de conceitos e estruturas dos painel de instrumentos e sistemas de navegação por satélite, princípios para criação de um **PDI**[2] e **SNS**[3] e exemplos de interfaces presentes no mercado.

O terceiro e quarto capítulos –**Propostas para Painel de Instrumentos** e **Sistema de Navegação por Satélite**– apresentam a visão geral de cada interface, a arquitetura, a implementação e exemplos para o ambiente de desenvolvimento das interfaces.

O quinto capítulo –**Casos de estudo**– apresenta a avaliação de desempenho e princípios para apresentação de informação das interfaces do PDI e SNS, e apreciação dos resultados obtidos.

O sexto e último capítulo –**Conclusão**– tem as conclusões da dissertação, as principais contribuições e o trabalho futuro.

[2]painel de instrumentos
[3]sistema de navegação por satélite

Capítulo 2

Estado da Arte

Este capítulo está divido em três secções. A primeira secção retrata os conceitos do mundo dos painéis de instrumentos. O segundo, os princípios para desenvolver um painel de instrumentos. A terceira descreve as interfaces de sistemas de navegação por satélite existentes no mercado.

2.1 Painel de instrumentos

PDI[1] é um painel situado no interior do veículo que contém instrumentos e indicadores do veículo [Merriam-Webster, 2011]. Este painel permite avisar o condutor quando o carro necessita de mais combustível, ou a temperatura do motor, entre outros. Nos anos 30 surgiram os primeiros **PDI** em veículos que não passavam de simples luzes que indicavam um problema com o veículo. Nos dias de hoje qualquer **PDI** está construído para minimizar distrações por parte do condutor [Edmonds, 2009]. No caso dos veículos rodoviários um **PDI** convencional tem no mínimo um velocímetro, um indicador de combustível, sendo usual, um tacómetro, um indicador carga da bateria, um manómetro da pressão do óleo e um da temperatura do motor [Ofria, 2007] [CVEL, 2011].

O primeiro **PDI** eletrónico surgiu em 1976 no carro *Aston Martin Lagonda* e mais tarde em 1978 nos Estados Unidos no *Cadillac Seville* [Wikipedia, 2011b]. No início um **PDI**'s era componente opcional nos veículos motor e uma parte integral nos carros de luxo do seu tempo. Com avanço do tempo e da tecnologia os painéis deixaram de trazer apenas um velocímetro, incorporando mais informações indispensáveis para

[1] Painel de instrumentos

condutor, tais como, temperatura exterior do veículo, nível de consumo instantâneo, distância possível de percorrer com o actual nível de combustível, entre outros. Na Europa surge em 1983 o primeiro carro com **PDI** digital pela *Renault 11 Electronic*.

2.1.1 Estrutura comum do painel de instrumentos

Os componentes mais usais em painéis de instrumentos em carros são velocímetro, indicador de combustível ou/e de carga de bateria restante, tacómetro, indicador da pressão do óleo e o indicador temperatura do motor, entre outros [Edmonds, 2009].

Velocímetro

O velocímetro é um dos indicadores mais importantes nos veículos pois indica a velocidade instantânea do veículo em condução. Antigamente a velocidade do velocímetro era obtida através cabos ligados à transmissão (velocímetro mecânico) atualmente o sistema é baseado em sensores elétricos, que avalia a velocidade de rotação das rodas do veículo (velocímetro eletrónico) [Edmonds, 2009].

Esta invenção surge na viragem do século 20 em que os carros estavam a tornar-se populares e deslocar-se cada vez mais rápido sem terem no seu interior qualquer indicação da velocidade instantânea a que seguiam, com o consequente aumento de acidentes nas estradas. A invenção de *Otto Schulze* permitiu aos utilizadores dos carros saberem a velocidade circulavam e assim ajustar a mesma consoante os perigos rodoviários. Na mesma altura em vários países começam a ser implementadas regras de trânsito com especial incidência na velocidade de circulação nos vários tipos de vias e zonas urbanas ou não. O velocímetro eletrónico é uma invenção relativamente recente, surgiu em 1993 enquanto a versão mecânica patenteada surgiu pelo inventor francês de Estrasburgo *Otto Schulze* em 1902 [Harris, 2007].

Indicador de combustível

Indica a quantidade de combustível. No entanto apresenta um nível mais baixo do que na realidade presente no depósito [Edmonds, 2009]. Este desvio é deliberado para os condutores pensarem que estão a fazer uma boa média de consumo e evitar que o condutor fique sem combustível, ou seja o indicador fornece a informação que temos menos combustível do que realmente apresenta no depósito [Ofria, 2007].

Tacómetro

O tacómetro fornece ao condutor a informação do número de rotações do motor por minuto (RPM) está a girar, permite assim ao condutor mudar de mudança nas

rotações certas para maior eficiência e economia na condução [Ofria, 2007].

Indicador da bateria

O indicador de bateria ou luz de aviso informam da carga atual da bateria se é suficiente para a ignição do motor ou se necessita de ser substituída [Edmonds, 2009].

Temperatura do motor

O indicador de temperatura do motor, é um ponteiro móvel com níveis quantitativos da temperatura do motor. Quando o a temperatura atinge temperaturas críticas acende uma luz indicativa [Edmonds, 2009].

Indicador da pressão do óleo

Indicador da pressão do óleo fornece a informação se óleo se encontra dentro dos limites aceitáveis para o funcionamento do motor, esta informação tanto pode ser visualizada por uma luz com um indicador ou ambos [Edmonds, 2009].

2.2 Desenho de um painel de instrumentos

As primeiras impressões que temos quando observamos um carro é o seu interior. No seu interior o painel de instrumentos é o elemento mais relevante que frequentemente pode ser um motivo de compra ou não [Edmonds, 2009]. Há condutores que preferem o antigo ponteiro analógico no velocímetro enquanto os mais jovens e mulheres têm uma maior afinidade pelo modelo digital [Edmonds, 2009]. Alguns condutores desejam ter à sua mão o mais variado tipo de informações acerca do carro por exemplo, alguns condutores que procuram saber o consumo instantâneo fazendo desta informação um jogo de condução eficiente. No caso da geração do *Baby boomer*[2] os condutores preferem informações básicas de fácil leitura.

O uso eficiente de interfaces *web* na *Internet* tornou-se uma prática comum que influenciou o *design* automóvel. As pessoas cada vez mais querem um ambiente envolvente como nas perspetivas usadas nos jogos [Mark, 2009]. A *Ford* e *Smart Design* reconheceram esse facto e começaram a meter mãos à obra com um protótipo apresentando informações simples de forma a não distrair o condutor. O modo monótono como os atuais painéis fornecem as informações devem passar da simples informação sobre a temperatura do motor, combustível, nível da bateria, para algo mais inovador.

[2]"Um *baby boomer* é uma pessoa nascida entre 1945 e 1964 na Grã Bretanha, Estados Unidos, Canadá ou Austrália."[Wikipédia, 2011]

2.2.1 Desvantagens e vantagens em PDI's digitais

Vantagens

Os atuais painéis de instrumentos analógicos têm sido sido gradualmente substituídos por congéneres digitais principalmente pelos fabricantes de carros. Esta mudança tem crescido devido às seguintes razões [Gryc, 2009a]:

- **Reutilizável** os fabricantes de carros podem equipar diferentes carros com o mesmo *hardware* através de uma simples troca de aparência gráfica dos painéis.
- **Dinâmico** painel adaptativo em função do tipo de informação mostrado.
- **Simples** disponibilização da informação essencial para o condutor evitar distrações.
- **Configurável** os fabricantes podem adicionar ou remover funcionalidades alterando o *software* do painel.

Desvantagens

- A espaço do painel limita o número de dígitos permitidos no Velocímetro digital em contraste com tradicional que permite uma larga margem de números e formatos dos dígitos [Wikipedia, 2011b].
- Dificuldade de leitura com incidência de luz no painel [Wikipedia, 2011b].
- Reparações caras [Wikipedia, 2011b].

2.2.2 Princípios gerais para criação de um PDI's

Processo de desenvolvimento de uma *interface* gráfica

O processo de criação de uma interface segundo a [Patrícia Goncalves, 2008] envolve essencialmente os seguintes passos:

- Levantamento de requisitos. Estudo comportamental dos utilizadores finais e modo como executam as suas tarefas de forma a moldar a interface.
- Esquematizar uma interface que responda as necessidades e requisitos estudados.
- Desenvolver várias interfaces de modo a que possa ser testado a sua eficiência diante dos futuros utilizadores.
- Avaliação contínua da interface ao longo do seu desenvolvimento.

Princípios específicos no contexto rodoviário

Comportamentos perigosos

A interface deve minimizar o risco de comportamentos perigosos com o seu uso quer para si quer para os outros condutores [Commission, 2006].

Atenção

A atenção necessária para uso da interface deve ser compatível com segurança de circulação rodoviária [Commission, 2006].

Distração

O condutor deve distrair-se o menos possível com a utilização do sistema de informação durante a condução de tal forma que controlo total do veículo não sejam comprometido [Commission, 2006].

Apresentação de informação

A interface deve fornecer as informações ao condutor de forma a não criar comportamentos de risco para segurança rodoviária e para os outros condutores [Commission, 2006].

Condução

O sistema deve permitir a utilização com coerência e harmonia com o condutor enquanto o veículo estiver em movimento [Commission, 2006].

Princípios para a apresentação de informação

As informações apresentadas pelo sistema deve ser visualizado de tal forma que o condutor é capaz de assimilar as informações relevantes com alguns relances de modo a não prejudicar a sua condução.

A capacidade visual do condutor está constantemente focada no trânsito é fundamental para a condução do veículo. Portanto, a capacidade para detetar e captar informações relevantes do sistema de informação é, e deve ser, limitada ao mínimo. Quantas mais vezes tiver que prender o olhar na obtenção de informação do sistema mais aumenta assim a probabilidade de causar um acidente. A proporção de informação mostrada deve restringir-se ao mínimo essencial de modo a reduzir a frequência que o condutor tem de prestar atenção. Um bom exemplo deste princípio é uma leitura de um item com informação ou/e ícone descritivo, com apenas um único olhar de um segundo. Em contraponto é por exemplo um sistema de navegação que apenas apresenta um ecrã rico em pormenores, que exige demasiada e prolongada atenção do condutor para visualizar a informação pretendida [Commission, 2006].

Um outro princípio é uso de normas nacionais e internacionais de legibilidade, de audibilidade, nos ícones, nos símbolos, nas palavras, nas siglas e nas abreviaturas na representação da informação. Todas estas diretivas aumentam consideravelmente a probabilidade de compreensão das representações por parte dos condutores no ambiente rodoviário [Commission, 2006].

As informações de relevância devem surgir ao condutor em momento oportuno durante condução de modo a corrigir ou alterar a sua condução durante a viagem. A condução exige ao condutor monitorização constante do ambiente para interpretar estímulos relevantes e a assim adaptar a sua condução [Commission, 2006].

Os avisos mais importantes para segurança devem ter prioridade máxima. O condutor deve saber das informações prioritárias de modo a agir em curto espaço de tempo, portanto, as informações de prioridade máxima devem surgir primeiro que as informações de rotina [Commission, 2006].

As informações sonoras que não pode ser controlado pelo condutor, não devem confundir-se com sons de outros painéis e indicadores do interior do carro e sons do exterior do carro [Commission, 2006].

Interface, ecrã e controlos

O condutor deve ser capaz de manter pelo menos uma mão no volante enquanto a outra interage com a interface. Deste modo por esta razão os dispositivos portáteis não são os indicados. A interface para estar de estar de acordo com este princípio, o sistema deve ser desenhado de tal forma que o condutor consiga interagir com sistema apenas com uma mão evitando uma situação de risco com uso das duas mãos [Commission, 2006].

A utilização da interface tem de ser orientada de modo a não requerer uma longa e contínua sequência de passos. Caso contrário se for uma pequena sequência de passos convém que seja rápida e contínua. Mesmo que exista uma sequência longa de passos a interface deve permitir que caso o condutor interrompa a sequência de passos, permita continuar mais tarde, assim que for possível. Os passos que o utilizador introduza deve ser dado ao seu ritmo e não ritmo imposto pela interface [Commission, 2006].

Comportamento do sistema

Durante a condução, as informações visuais que não estejam relacionadas com a mesma devem ser omitidas automaticamente de maneira que o condutor não se distraia. O comportamento do sistema não deve interferir com os controlos necessários para a condução e segurança rodoviária. Este princípio visa garantir que o condutor

tenha sempre o controlo total do veículo, em um nenhuma maneira seja influenciada pela interface. Isto significa que o sistema não deve sobrepor informações relevantes da condução automóvel. Novas informações com impacto na segurança rodoviária devem ser apresentadas sobre a atual informação da interface de forma a salvaguardar o seu visionamento ao condutor [Commission, 2006].

2.2.3 Exemplos de painéis

Delphy Head-up display

Head-up display[3] foi introduzido pela companhia *Delphi Corporation* no carro protótipo constituído por diversos instrumentos digitais e com particularidade da visualização de diversas informações no vidro para-brisas do veículo. Esta tecnologia não é uma ilusão ótica mas sim uma rentabilização de informações visuais usada em aviões militares na qual a informação é projetada no para-brisas para sua visualização, exemplo figura 2.1 [CVEL, 2011] [CVEL, 2010].

Figura 2.1: *Head-up display* presente no avião de guerra.

Os painéis de instrumentos introduzidos pela *Delphi* incidem principalmente na informação do velocímetro e a configuração do sistema de entretenimento sem tirar os olhos da estrada no seu manuseamento [Wood, 2008], figura 2.2 [CVEL, 2011].

BMW's head-up display

A condução na auto-estrada exige uma regular verificação da velocidade no velocímetro, essa verificação constitui um desvio do olhar da estrada (relance), por exemplo, uma viagem a 100 km por hora compreende um deslocamento médio de 28

[3] ecrã transparente que permite apresentar os dados sem que os utilizadores necessitam de desviar os olhos da estrada [Wikipedia, 2011f].

Figura 2.2: *Head-up display* no carro.

metros no sem ter olhado para a estrada nesse instante [BMW, 2010]. O **HUD**[4] da BMW à semelhança dos pilotos de jato tem as informações projetadas no para-brisas a cores e de fácil leitura da velocidade, navegação, alerta de colisão, velocidade limite permitida e outras informações importantes à condução, ver figura 2.3 [BMW, 2010]. As informações exibidas são de fácil leitura nas diversas situações de luminosidade e pode ser personalizado ao gosto do condutor [CVEL, 2010].

Figura 2.3: *BMW Head-up display.*

Ford dasboard

Desenvolvido em conjunto com *Smart Design* e a *Ford* introduz uma riqueza de informação atraente sem distrair o condutor. Os habituais indicadores de combustíveis foram substituídos por informações mais iconográficas tal como por exemplo um ícone do depósito de combustível com líquido amarelo traduz o combustível presente na viatura, figura 2.4 [Moere, 2009]. Uma outra característica deste painel é o nascimento ou regressão de uma planta verde no lado direito do painel consoante a eficiência da condução do condutor. Toda esta inovação deste painel pode ser trocada pelo painel tradicional de instrumentos.

[4] *Head up display*

Figura 2.4: *Ford PDI.*

Lamborghini Reventón PDI

Este é um caso de um **PDI** pouco apelativo e demasiado futurista, figura 2.5 [Eisenberg, 2008], que levou cerca de 20 donos do *Lamborghini Revenón* a trocarem este painel por um mais tradicional.

Figura 2.5: *Lamborghini PDI.*

Painel de instrumentos digital com sistema operativo *QNX*

Com o aparecimento de PDI digitais, empresas que disponibilizaram ambientes de desenvolvimento como *QNX*, começaram a virar-se para o mercado potencial da criação de PDI digitais. O painéis analógicos são constituídos por pequenos sistemas eletrónicos sem sistema operativo. As informações visualizadas no painel são programados por pequenos circuitos que processam toda a atividade presente

no painel. Ao contrário os atuais painéis digitais possuem na sua constituição um sistema operativo. O sistema operativo de tempo rela *QNX* permite implementar painéis que renderizam os componentes gráficos através do sistema operativo de tempo real estendendo assim as funcionalidades dos tradicionais sistemas eletrónicos que anteriormente estavam mais limitados [Gryc, 2009a].

O painel digital apresentado pela empresa *QNX* é composto por duas camadas de software, o primeiro é o fundo do painel implementado com tecnologia da *Adobe Flash Player* em duas dimensões, a segunda parte é aplicação dos ponteiros exibidos no painel usando a tecnologia *OpenGL ES library* em três dimensões. Ambas camadas comunicam com a camada *Digital acquisition manager* abaixo que fornece as informações digitais do carro para aplicação gráfica ver figura 2.6 [Gryc, 2009a].

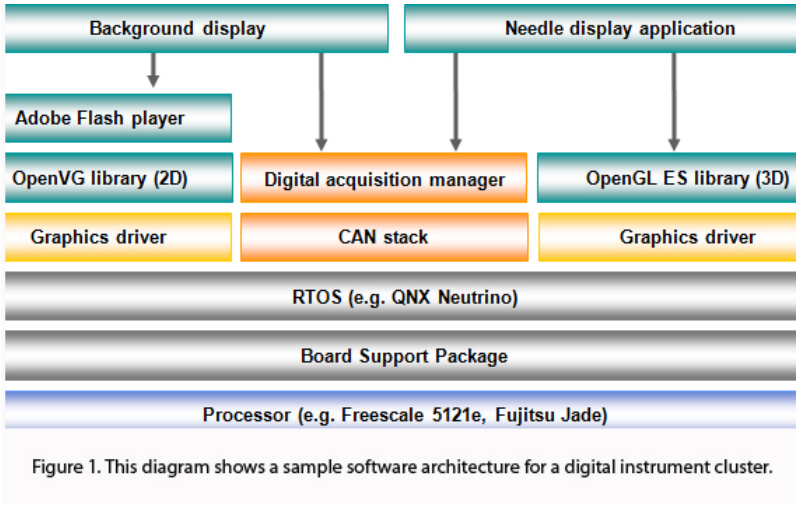


Figura 2.6: Arquitectura do Software *QNX*.

O uso de uma *framework* padrão com *OpenGL ES* e *OpenVG* na estrutura do sistema do painel permite uma vasta escolha de *ToolKits* gráficos para escolhas *design*. *OpenGL ES* é uma extensão do *OpenGL*, um dos mais utilizado pela *API 3D* na indústria de computadores. Como resultado, os fabricantes de automóveis escolheram *OpenGL ES* pela vasto leque de funções gráficas ao seu dispor bem como código fonte aberto e respectiva documentação. As principais funções gráficas disponíveis pelo *OpenGL ES* são *alpha blending*[5], sombreamento *Gouraud*[6] e mapeamento e modelagem de textura, transformações, iluminação e entre outras técnicas. A reutilização do código usado pelo *OpenGL ES* permite que possa ser usada em novos projetos ou outros da mesma família, caso a aplicação mantenha o esqueleto do *OpenGL ES* fica fica com possibilidade de execução em diferentes chips e sistemas

[5]processo de combinar uma cor transparente em primeiro plano com uma cor de fundo, produzindo assim uma nova cor através da sua fusão `http://www.fastgraph.com/help/alpha_blending.html`

[6]técnica de computação gráfica de renderização de sombras em objectos 3D `http://www.howstuffworks.com/question484.htm`

operativos [Gryc, 2009b].

Para o desenvolvimento gráfico de painéis de instrumentos com *QNX* o programador pode usar ferramentas da *Adobe* tais como *Flash Player* no fundo do painel através da livraria *OpenVG 2D* presente no sistema operativo. Possibilitando assim uma grande variedade de ferramentas e possibilidades de *design* gráficas, para os ponteiros o programador/*designer* usa a livraria *OpenGL ES* para desenhar os ponteiros que depois sobrepõe sobre a camada do fundo do ecrã. Veja-se o exemplo da figura 2.7 [Gryc, 2009b] de um **PDI** desenvolvida em *QNX*.

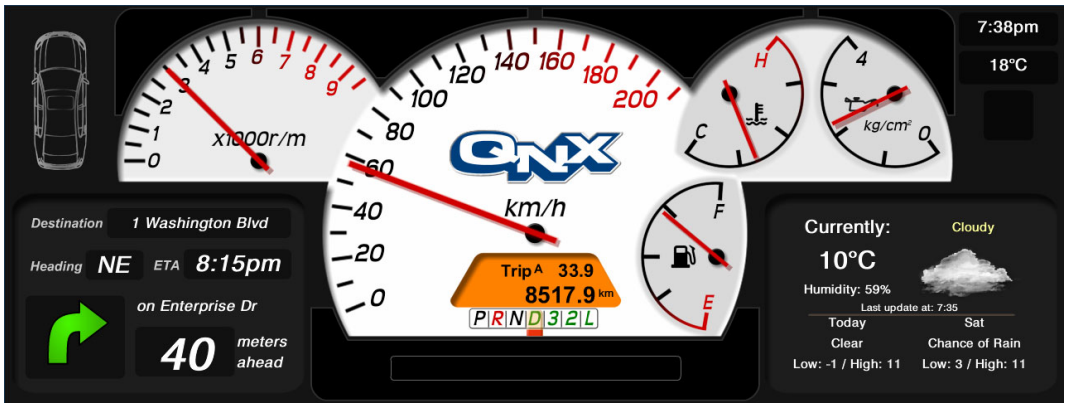


Figura 2.7: Exemplo de um painel desenvolvido com *QNX*.

Outras possibilidades e ideias usadas em painéis como demonstrado na figura 2.7 é usando o conceito *Cloud* com [Gryc, 2009b]:

- Apresentação dos limites de velocidade, condição da via.
- Personalização do aspeto gráfico do painel de instrumentos.
- Mudança automática do modo noite/dia e configuração do painel para sistema métrico/inglês.
- Redução no destaque dos indicadores menos importantes tais como pressão do óleo, voltagem da bateria quando situados em níveis normais de modo a que o condutor evite distrações desnecessárias para estes indicadores.
- Alteração do modo de apresentação consoante o estilo de condução dando ênfase aos indicadores correspondentes.
- Incorporar funcionalidade de navegação, bem como visionamento de câmara no auxílio ao estacionamento e consulta de informações sobre o tempo, bomba de abastecimento mais barata, etc através da *Internet*.

Painel de instrumentos para pequeno veículo elétrico

O **VEIL** - Veículo Elétrico Isento de Licença de Condução - é um projeto em desenvolvimento no ISEC, (Instituto Superior de Engenharia de Coimbra), para a construção de um pequeno veículo elétrico constituído por diferentes de fontes de energia e com uma arquitetura reconfigurável baseada em *X-by-wire*[7] [Marques et al., 2011].

O painel apresenta na figura 2.8 [Marques et al., 2011] as informações necessárias para condução , a sua disposição pode ser personalizada pelo condutor, apenas quando o veículo encontra-se parado.

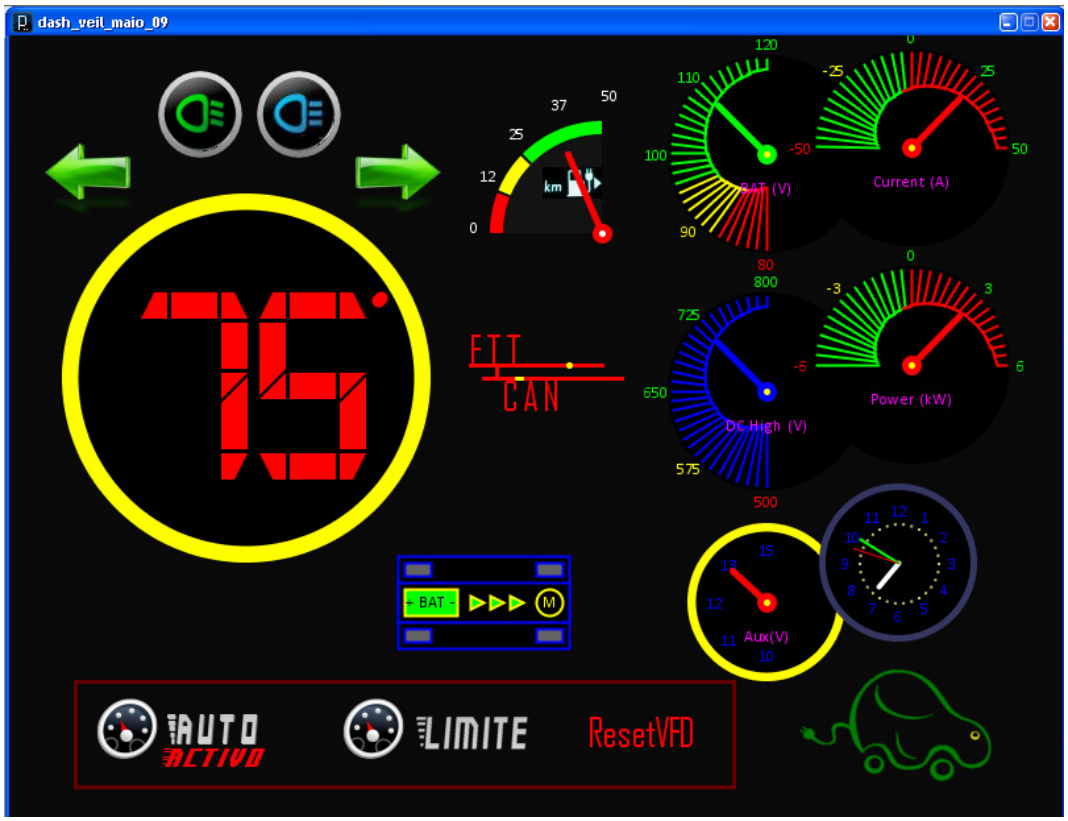


Figura 2.8: Vista por defeito do PDI do veículo **VEIL**.

2.3 Sistema de navegação

O sistema de navegação atuais como *Google Maps Navigation* usa GPS e localização por *IP*[8] através das redes móveis terrestres, tornando-se num sistema com melhor

[7] *X-by-Wire* é um termo genérico que se refere à substituição de sistemas mecânicos ou hidráulicos, tais como a travagem ou direção, por outros eletrónicos. fonte:`http://hal.inria.fr/inria-00000562/en/`

[8] Endereço *IP* de *internet*

precisão geográfica. O sistema faz a triangulação da sua posição pelos os satélites e com seu *IP*.

A condução é uma atividade de exigência física e mental que leva a diversas condicionantes, o condutor além de consultar as direções a tomar no uso do **SNS** (sistema de navegação por satélite) tem de estar atento à estrada, outros veículos, peões, entre outros. Deste modo, é exigido a implementação de um **SNS** que evite que ao mínimo as distrações do condutor, e evitando direções rodoviárias demasiado complexas para a condução do veículo mesmo que tenha que ser uma caminho ligeiramente mais longo[Baus et al., 2002].

2.3.1 Princípios

Um dos critérios para escolha e desenvolvimento de um sistema de navegação é o tamanho do ecrã que deve permitir uma boa leitura de informações mesmo com luz solar a incidir no ecrã [Newcomb, 2011].

Ecrã de toque devem ser fáceis de manusear e ao alcance do condutor e introdução de destinos por voz [Newcomb, 2011].

Mapas alojados no disco do GPS padecem de desatualização ao contrário dos alojados online [Newcomb, 2011].

2.3.2 Exemplos

Comunicação carro-carro

A *General Motors* anunciou o desenvolvimento de um sistema de alertas para os condutores sobre os outros veículos em circulação num raio de 400 metros. Esta comunicação carro-para-carro (CPC) será baseada na comunicação rádio e os recetores GPS[9] presentes nos carros com o sistema CPC [Wood, 2008].

Sistema de navegação por Radiofrequência

A *Toyota* desenvolveu um protótipo de sistema de navegação que comunica por radiofrequência com torres presentes na estrada [Kanemitsu et al., 1991]. O sistema **RACS** (*Road/Automobile Communication System*) gera as comunicações e permite fornecer informações de trânsito ao condutor e enviar informações sobre a estrada e condições atmosférica pelo condutor a outros automobilistas e às autoridades competentes.

[9] *Global Positioning System*

O **RACS** a assenta num núcleo central de controlo das comunicações, no transmissores colocados ao longa das estradas e nos sistemas de navegação como visível na figura 2.9 [Kanemitsu et al., 1991].

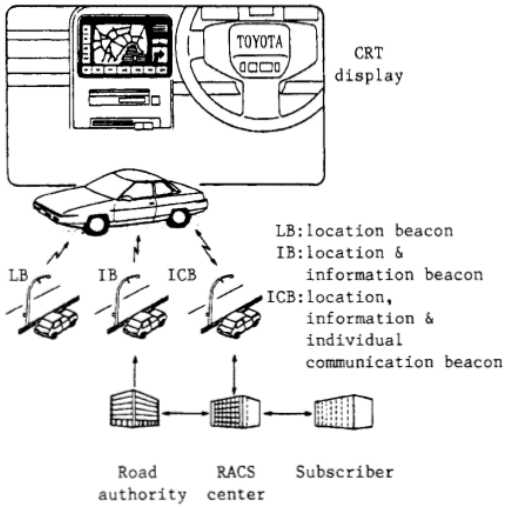


Figura 2.9: Arquitectura do sistema **RACS**.

Figura 2.10: Painel de navegação **RACS**.

A navegação fornecida por este sistema assenta pela comunicação com as torres na via, quando não existem, o sistema usa método de localização estimada através da velocidade do veículo, diferenças de tempo entre dois pontos entregando estas informações com o mapa de caminhos, estimando assim a possível localização do veículo. Com constantes mudanças de métodos de localização o condutor fica sujeito a erros na sua localização atual e só corrigida à passagem pela antena de comunicação [Kanemitsu et al., 1991].

A nova informação proveniente do sistema de navegação é emitida pelo sintetizador de voz [Kanemitsu et al., 1991]. O mapa de estradas está contido na unidade **ROM**[10] e contem 350 quilómetros quadrados das cidades de *Tokyo*, *Yokohma* e

[10]memória apenas de leitura

Kamasaki em quatro tipos de escalas, figura 2.10 [Kanemitsu et al., 1991].

Sistemas de navegação de código aberto

Roadnav é uma solução navegação aberta multiplataforma que permite através da sua posição dada pelo **GPS** traçar percursos no mapa conforme o destino pretendido exemplo figura 2.11 [Navit, 2011a]. Este sistema permite usar sintetizadores de voz da *Microsoft SAPI 5.1*[11] , *Festival*[12], *flite*[13], e *Mac OSX*[14] para direções de navegação.

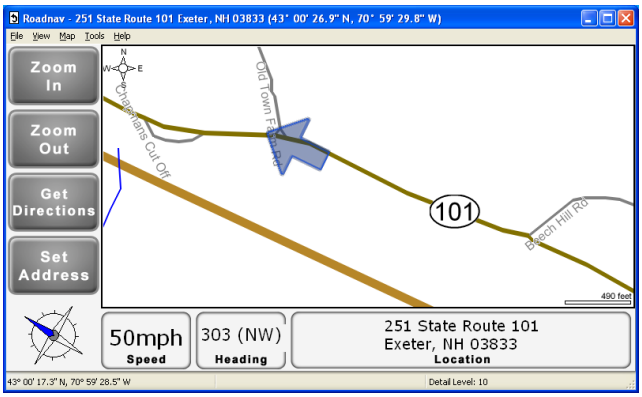


Figura 2.11: Interface do sistema de navegação por satélite *Roadnav*.

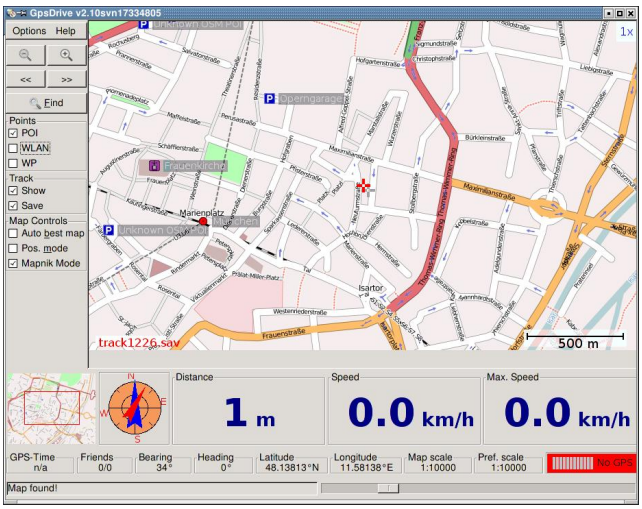


Figura 2.12: Interface do sistema de navegação por satélite *Gpsdrive*.

[11] http://www.microsoft.com/download/en/details.aspx?id=10121
[12] http://www.cstr.ed.ac.uk/projects/festival/
[13] http://www.speech.cs.cmu.edu/flite/
[14] http://www.apple.com/macosx/

Gpsdrive é um **SNS** para bicicleta, navio, avião, entre outras aplicações. Os mapas ajustam-se automaticamente conforme a sua posição e podem ser atualizados pela *Internet* através do site `http://www.openstreetmap.org/` figura 2.12 [Gpsdrive, 2011]. O *Gpsdrive* como tantos outros do mesmo segmento permite emitir direções em voz. O sistema está inscrito em *C* e usa *GTK+* para a interface e pode ser usado nos sistemas operativos *Linux*[15], *Mac OSX* e *FreeBSD*[16].

Figura 2.13: Interface do sistema de navegação por satélite *Navit*.

Navit é um **SNS** para *Linux*, *Windows* e dispositivos móveis. A sua interface apresenta uma vista em duas e três dimensões ver figura 2.13 [Navit, 2011b], visualização de pontos de interesse e os mapas são gerados dinamicamente em tempo real a partir de várias fontes de dados vetoriais como está na figura.

[15] `https://www.linux.com/`
[16] `http://www.freebsd.org/`

Capítulo 3

Proposta para um painel de Instrumentos

Este capítulo está dividido em quatro secções. A primeira faz o enquadramento da proposta da interface. A segunda descreve estrutura da interface proposta. A terceira apresenta a implementação da interface. A quarta última apresenta um exemplo de desenvolvimento de um componente do PDI.

3.1 Visão geral

O objetivo do trabalho aqui descrito é construção de uma interface para um painel de instrumentos para um veículo elétrico. A interface é constituída pelos indicadores presentes nos carros usais, tais como, velocímetro, tacómetro, indicador de bateria, indicador de bateria, entre outras luzes. No mundo automóvel existem milhões e milhões de carros e cada um tem PDI próprio e diferentes dos restantes, do ponto de vista informático os PDI não têm uma arquitetura padrão e aberta para terceiros para desenvolver e inovar os PDI no modo de transmitir as informações que chegam ao utilizador. No caso da parte gráfica existe o mesmo problema para desenvolver temas adicionais para o PDI os *designers*[1] deparam-se com o problema de os fabricantes não forneceram ferramentas e/ou documentação para desenvolvimentos de temas por terceiros.

A interface proposta pretende eliminar os problemas anteriormente referidos através

[1] "Pessoa que trabalha em design." `http://www.priberam.pt/dlpo/default.aspx?pal=designer`

de um painel de instrumentos aberto a terceiros para desenvolvimento de aplicações e temas. A arquitetura do sistema que suporta a interface está dividida em vários módulos de comunicação que será referida em pormenor na secção 3.2. Cada módulo tem *API*[2] para outras pessoas poderem desenvolver aplicações. No caso dos *designers*, essa parte é assegura através de *XML*[3] onde se encontra todas especificações e restrições para criação de temas independente do tipo ferramentas de *design* [Sol, 2011]. O painel, além dos indicadores, conta com um sistema de navegação por satélite, apresentado no capítulo 4.1, que permite navegar para o destino pretendido, assim como apresentar localizações de pontos de interesse.

3.2 Arquitetura

A arquitetura do sistema do **PDI** está divida em duas partes. Arquitetura global do sistema e a arquitetura da interface gráfica. A arquitetura global é constituída por um módulo de informações do veículo e um outro módulo responsável pela interface gráfica do **PDI** caso da figura 3.1.

3.2.1 Arquitetura global do sistema

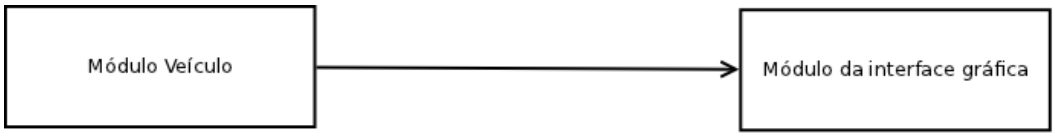


Figura 3.1: Arquitetura global do sistema

A arquitetura global do sistema é formada por um **módulo veículo** que processa os sinais digitais provenientes dos componentes eletromecânicos do veículo, tais como, a velocidade instantânea, a rotações do motor, o nível de energia, o nível do combustível, o ponteiro da temperatura do motor e outras luzes de aviso. Estes sinais são processados por um microcontrolador *Arduino*[4]. O módulo do veículo recolhe os sinais processados e são disponibilizados por uma *API* através da linguagem de programação *Java*[5] para acesso dos restantes módulos da arquitetura ou para desenvolvimento de terceiros. O **módulo da interface** recebe os valores pela *API* do módulo veículo e representa-os nos respetivos componentes do painel.

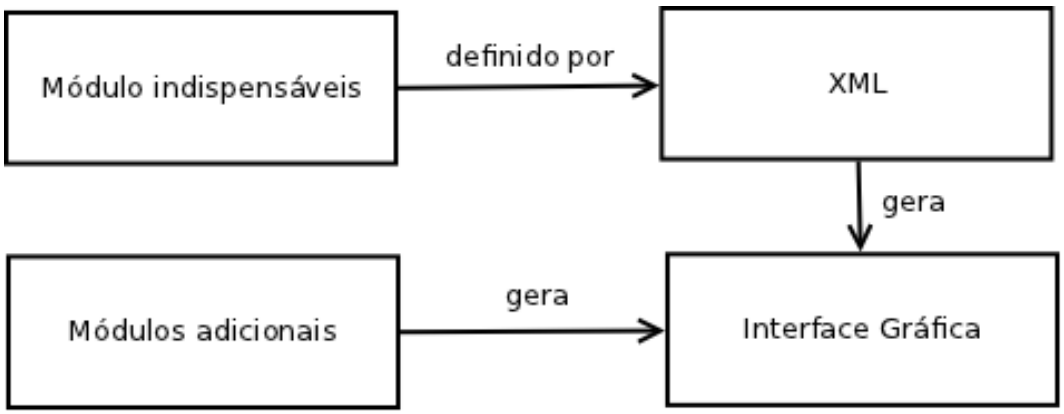


Figura 3.2: Arquitetura do módulo da interface gráfica

3.2.2 Arquitetura do módulo da interface gráfica

A arquitetura do módulo da interface gráfica é composta por dois módulos de componentes, um **indispensáveis** e outro **módulo adicionais** e por dois componentes, um de ***XML*** e outro da **interface gráfica**, como se pode visualizar na figura 3.2.

Os módulos adicionais e indispensáveis fazem a separação dos componentes gráficos essenciais para segurança rodoviária dos adicionais para PDI. O módulo indispensáveis funcionam de forma independente e autónoma do módulos dos componentes adicionais do painel, assegurando assim, que em caso de falha no funcionamento de algum elemento adicional do painel não se prejudique a condução do veículo. O terceiro elemento fundamental na implementação da interface é o componente de *XML* que permite especificar as características dos elementos a serem desenhos no PDI. A especificação dessas características em *XML* permite que os dados acerca dos elementos do painel possam usados por outras linguagens de programações e outras ferramentas de desenvolvimento. O *XML* é uma linguagem de marcação padrão mundialmente aceite [Bray and Yergeau, 2008] [Sol, 2011] [Rúben Fonseca, 2007] que separa o conteúdo da formatação, de fácil legibilidade para pessoas e computadores. Um componente usado no *XML* é por exemplo do indicador do óleo do motor, como se pode observar na amostra de código na figura 3.2.2.

[2]Interface para programação de aplicações.
[3]Linguagem de marcação criada para descrever e caracterizar dados.[Bray and Yergeau, 2008]
[4]`http://www.arduino.cc/`
[5]`http://www.java.com/pt_BR/download/index.jsp`

```
<indicator>
                    <imageActive>oleo.png</imageActive>
                    <imageInactive>oleo2.png</imageInactive>
                    <px>447</px>
                    <py>567</py>
</indicator>
```

Figura 3.2.2 Código *XML* com as definições de desenho do indicador do óleo.

O indicador definido no *XML* é constituído pela etiqueta raiz **indicador** que especifica o tipo componente presente no PDI. As etiquetas ***imageActive*** e ***imageInactive*** contêm o caminho das imagens necessárias para o indicador. Essas imagens permitem com que o indicador ter dois estados, um estado desligado e uma ligado, ou seja, o indicador ou está aceso ou está apagado. As restantes duas etiquetas ***px*** e ***py*** fornecem as coordenadas do ponto para desenho do indicador no painel.

O componente da interface e a componente do *XML* encontram-se separados para permitir uma distinção entre a representação gráfica do painel da configuração interna dos componentes do painel. Este distinção permite modificação e personalização dos elementos gráficos do painel pela simples modificação do ficheiro *XML*. Os componentes do módulo indispensáveis são os únicos especificados no ficheiro *XML* porque são os indicadores fundamentais para funcionamento do painel e segurança rodoviária. No módulo dos adicionais os componentes não são englobados no ficheiro *XML* pela sua menor importância no painel de instrumentos, assim, a componente da interface gráfica é pelos dados provenientes do *XML* e os componentes do módulo adicionais.

3.2.3 Constituição do documento *XML* do PDI

Os componentes indispensáveis estão definidos no documento *XML* como foi referido na secção anterior e a suas propriedades estão agrupadas no *XML* por etiquetas da seguinte forma:

painel

Propriedades: não aplicável.

Etiqueta raiz que contém os restantes elementos presente no painel.

nome

Propriedades: não aplicável.

Etiqueta com um campo onde está definido o nome do painel.

screen

Propriedades: *width* e *heigth*.

As *tags*[6] *width* e *heigth* contêm largura e altura respetivamente do PDI.

background

Propriedades: *file* e *refreshable*.

O elemento *background* é definido por uma imagem que ocupa o ecrã na totalidade. A propriedade *file* especifica a localização da imagem a ser desenhada no ecrã. No caso do *refreshable* possuí um valor booleano *true* ou *false* que indica se o elemento pode ser redesenhado.

pointerGauge* e *pointer

Propriedades *pointerGauge*: *id*, *file*, x e y.

Propriedades *pointer*: *id*, *color*, *p0x*, *p0y*, *p1x*, *p2y*, *p2x*, *p2y*, *vmin* e *vmax*.

O elemento *pointerGauge* define um componente analógico, como por exemplo na figura 3.3 um velocímetro de um veículo. O desenho do *pointerGauge* é formado pela localização da imagem e as coordenadas para desenho no PDI junto das *tags file*, x, y. Além das *tags* atrás referidas está definida a *tag pointer* que define como é desenhado o ponteiro no PDI. O ponteiro tem como propriedades: a cor, *p0x*, a *p0y* e *p1x*, *p1y* para posições inicial e final onde o ponteiro é desenhado. O ponto final do ponteiro está apontado para o valor mínimo da imagem do *pointerGuage*. A posição *p2x* e *p2y* é o ponto na qual o ponteiro pode atingir o valor máximo na imagem do *pointerGauge*. As restantes propriedades *vmin* e *vmax* indicam os intervalos mínimo e máximo, na qual, o ponteiro pode variar. O elemento *pointerGauge* permite que seja desenhado qualquer tipo de indicador analógico assente nas características atrás enunciadas, exemplo um velocímetro e/ou tacómetro[7].

Figura 3.3: Indicador de um veículo que utiliza um medidor analógico com ponteiro.

levelGauge* e *levelGaugeElement

Propriedades *levelGauge*: *id*, *file*, x e y.

Propriedades *levelGaugeElement*: *id*, *elements*, *file*, x, y, *length* e *heigth*

[6] etiqueta

[7] "Instrumento com que se determina a velocidade dos movimentos de uma máquina." `http://www.priberam.pt/dlpo/default.aspx?pal=tacometro`

O elemento *levelGauge* especifica que um componente do PDI é constituído por vários estados como por exemplo de indicador de carga de bateria da figura 3.4. A propriedade *file* do *levelGauge* e *levelGaugeElement* especificam a localização da imagem a ser desenhada no ecrã. No caso das *tags* x e y representa a posição das imagens no ecrã do PDI. No *levelGaugeElement* a propriedade *elements* define o valor máximo de níveis que a imagem do indicador do tipo *levelGauge* suporta nas suas dimensões. Nas restantes propriedades *length* e *heigth* são as dimensões numéricas da imagem presente no PDI. Com estas propriedades e definição pode-se construir o mais variado tipo indicadores de estados, por exemplo, o nível da temperatura do motor e/ou o nível de combustível disponível entre outros.

Figura 3.4: Componente de um PDI com vários patamares no indicador de carga da bateria.

indicator

Propriedades: *id*, *imageActive*, *imageInactive*, x e y.

O elemento *indicator* define um indicador informativo sobre veículo, caso das figuras 3.5 e 3.6 que mostra o aviso de porta aberta do veículo. As propriedades *imageActive* e *imageInactive* fornecem as localizações das imagens que permitem o indicador do PDI poder aceder e apagar, pela simples transição destas duas imagens. As restantes propriedades x e y definem a localização no PDI onde o indicador vai ser desenhado.

Figura 3.5: Indicador de porta aberta do veículo no estado ligado.

Figura 3.6: Indicador de porta aberta do veículo no estado desligado.

staticImage

Propriedades: *id*, *file*, x e y.

As propriedades *file*, x e y, representavam a localização e coordenadas de desenho no PDI respetivamente. Ao contrário do elemento *indicador* definido anteriormente este indicador não é dinâmico, apenas tem uma imagem estática desenhada no PDI.

label

Propriedades: *id*, x, y, *color*, *fontType*, *fontSize* e *refreshable*.

A *tag label* defini como uma caixa de texto é apresentado no PDI. A propriedade x, y determina a posição onde o texto vai ser desenho no ecrã do PDI com cor da propriedade *color*. As propriedades *fontType* e *fontSize* definem o tipo de letra a usar e seu tamanho respetivamente. O valor *true* e *false* booleano do campo *refreshable* permite redesenho da caixa de texto no PDI.

Para visualização de um exemplo da representação de componentes num documento *XML*, poder ser consultado no anexo A o documento que foi utilizado na definição do painel de instrumentos elaborado nesta dissertação.

3.3 Implementação

3.3.1 Tecnologias usadas

Java

Na implementação do painel de instrumentos foi usada a tecnologia *Java* para definições e especificações dos elementos constituintes do painel de instrumentos. O *Java* é uma linguagem de programação de alto nível, de sintaxe similar a C[8] e C++[9]. Apresenta uma sintaxe simples, orientada a objetos, compilada, independentemente de arquitetura, *multithreaded*[10], com *garbage collection*[11], robusta, segura, extensível e organização estruturada [Ferreira Filho, 2000] [Moura, 2010]. Os programas *Java* correm sobre a *Java Virtual Machine*, que por sua vez, interpreta os *bytes-codes*[12] gerados pela compilação do código fonte das aplicações *Java*. Os *bytes-codes* podem ser executados em qualquer sistema operativo que tenha a *JVM*[13] instalada [Ferreira Filho, 2000] [Moura, 2010]. Ou seja, a aplicação pode ser executada em qualquer sistema operativo sem necessidade de alterar e recompilar o código para a máquina em questão. A escolha recaiu no *Java* devido às razões acima descritas e da incerteza de qual seria o sistema operativo a usar no PDI. Assim, apostou-se na portabilidade do código *Java* para a eventualidade de mudança de plataforma.

[8] `http://www.open-std.org/jtc1/sc22/wg14/`
[9] `http://www.open-std.org/jtc1/sc22/wg21/`
[10] multitarefa
[11] coletor de lixo
[12] código binário
[13] *Java Virtual Machine*

A tecnologia *Java* possui a ferramenta *Java Swing* para desenho de interfaces gráficas para utilizadores [Newmarch, 1999]. A ferramenta *Java Swing* oferece um enorme conjunto de componentes de interfaces gráficas e uma extensão de funcionalidades da antiga ferramenta *Java AWT*[14] [Oracle, 2011j]. A escolha para desenvolver o PDI foi *Java Swing* pois permite a utilização dos componentes *JFrame* e *JPanel*.

- *JFrame* é o componente que permite desenhar as janelas das aplicações com uma determinada altura e largura, definidas pelo programador [Oracle, 2011e], a escolha justa para definição do ecrã do painel de instrumentos. Para permitir o desenho na *JFrame* é necessário o componente *JPanel* esteja inserida na *JFrame*.

- *JPanel* serve de contentor para o outros componentes gráficos [Oracle, 2011h]. O *JPanel* permite desenhar outros componentes personalizados por terceiros [Oracle, 2011c]. A classe *Java JPanel* possui, ainda, o método *paintComponent()* que contém todo o código de desenho.

- A chamada do *paintComponent* é invocado pelo método *paint* da mesma classe. Este passo faz com que todos componentes que estejam contidos no *JPanel* sejam redesenhados desnecessariamente provocando perda de performance do PDI, dado que existem componentes que não necessitam de ser redesenhados [Oracle, 2011k] [Oracle, 2011b]. A solução deste problema passou pelo *override* do método *paint* de forma a que apenas alguns componentes sejam atualizados constantemente.

3.3.2 Processamento do documento *XML*

A configuração dos componentes do PDI é efetuada no documento *XML* como anteriormente referido na secção 3.2. Para a implementação do sistema é necessário obter as definições dos componentes a serem desenhados, esse método é obtido pelo processo de *parsing*[15] do documento *XML* [Wikipedia, 2011d]. O *parsing* do documento é efetuado através de um *parser*[16] que retira os valores contidos no agrupamento estruturado das *tags* do documento *XML*. Em *Java* existe a ferramenta *SAX*[17] para o *parsing* de documentos em *XML*. Uma das principais vantagens do uso do *SAX* é eficiência no *parsing* do documento em relação a ferramentas semelhantes. O *SAX* faz a leitura dos dados de forma sequencial do documento sem ser preciso ler a sua totalidade. O *Parsing* necessita de um *handler* para definir as ações de leitura do documento quando encontra as *tags* correspondentes, ou seja, define quais são os

[14] *Abstract Window Toolkit* ferramenta inicial de desenvolvimento de interfaces gráficas [Wikipedia, 2011a].
[15] análise sintática
[16] ferramenta de análise sintática de sintaxe e retorno de dados.
[17] *Simple API for XML* `http://www.saxproject.org/`

procedimentos a realizar quando é invocado um evento na leitura do *XML* guardando as informações respetivas.

O resultado do *parsing* do documento *XML* é uma lista de dados acerca dos componentes que vão ser desenhados. A lista é uma hierarquia de classes, na qual cada classe tem os atributos dos componentes processados pelo *parsing* do documento do *XML*. As classes geradas do processamento do *XML* são estendidas pela classe abstrata *ObjetoInterface* que especifica e adiciona as classes a uma lista de componentes. As classes criadas e especificadas pelos diferentes tipos são as seguintes:

Screen - classe que define o componente *screen*.

Background - classe que define o fundo do componente *background*.

PointerGauge - classe que define o componente *pointerGauge*.

Pointer - classe que define o ponteiro do componente *PointerGauge*.

LevelGauge - classe que define o componente do *levelGauge*.

LevelElement - classe que define escala do componente *levelGauge*.

Notifier - classe que define o componente *indicator*.

StaticImage - classe que define o componente *StaticImage*

Label - classe que define o componente*Label*.

Exemplo do *parsing* do componente *screen* do documento *XML*

No processo de *parsing* do documento *XML* quando é encontrada a *tag screen* (< *screen* >), o *parser* associa as próximas *tags* como propriedades do componente *screen* até encontrar a *tag* (< /*screen* >) de fecho das propriedades. Até ao fecho da *tag* as propriedades vão sendo adicionadas à instância do objeto *Screen*, e o mesmo objeto é adicionado à lista de componentes após a inserção da última propriedade. O restante *parsing* em falta decorre até ao fim do documento para os restantes componentes descritos no documento *XML*.

3.3.3 Separação dos componentes *refreshable* dos estáticos

Como anteriormente referido na secção 3.3.1 existe no PDI componentes que não são necessários serem redesenhados. Para maior eficiência da aplicação *Java* do PDI procede-se a essa separação dos componentes dos *refreshable* e dos estáticos (não *refreshable*) em duas listas de objetos do tipo *ObjetoInterface*. A divisão é efetuada pela variável booleana *refreshable* presente alguns dos componentes. Os

componentes que não possuem a propriedade *refreshable* são à partida *staticImage* e *pointerGauge* e são colocados diretamente na lista dos componentes estáticos durante processo de *parsing* do documento *XML*. Os componentes atrás referidos são marcados como estáticos porque no PDI não sofrem qualquer atualização da sua estrutura, ou seja, não têm valores que necessitam de constantes atualizações. Toda esta divisão fornece um aumento de eficiência à aplicação como foi referenciado na secção 3.3.1.

3.3.4 Algoritmo de desenho dos componentes *refreshables* e estáticos

Uma vez concluída a lista dos componentes estáticos e *refreshable* inicia-se o desenho dos componentes no PDI. Para esse processo é necessário fazer *override* do método *paint()* para evitar o redesenho de todos os componentes, como anteriormente referido em 3.3.1. Assim o método *paint()* foi reconstituído da seguinte forma:

```
a cada iteração de paint(){

    atualiza_componentes_refreshable();

    se primeira iteração:

        pinta_componentes_não_refreshable();

    pinta_componentes_refreshable();

}
```

O método *paint()* inicia a comunicação com o **Módulo do Veículo** (secção 3.2.1) para o *download* dados relevantes dos componentes *refreshable*, por exemplo velocidade instantânea necessária para o velocímetro. Seguindo o algoritmo descrito em cima, no primeiro instante que o método *paint()* é executado desenha os componentes estáticos pois, estes não contêm dados que necessitam de constante atualização. E por fim os componentes *refreshable* são desenhados nas segundas e próximas chamadas do método *paint()*.

Um dos problemas do desenho dos componente foi o constante *flickering*[18] das imagens desenhadas no PDI. Para solucionar essa problema usou-se o técnica *double-buffering*[19] que consiste no desenho dos elementos numa nova imagem temporária,

[18] imagens desenhadas no ecrã exibem oscilações na sua posição dando a impressão de "tremor". `http://www.thefreedictionary.com/flickering`

[19] armazenamento duplo em memória

após a junção dos elementos na nova imagem temporária, esta é adicionada à imagem de fundo do PDI, como demonstrado na figura 3.7 [Oracle, 2011d]. Com este algoritmo evita-se assim o demorado e pesado processo de desenho de imagem, uma por uma e reduz-se assim o problema do tremor das imagens.

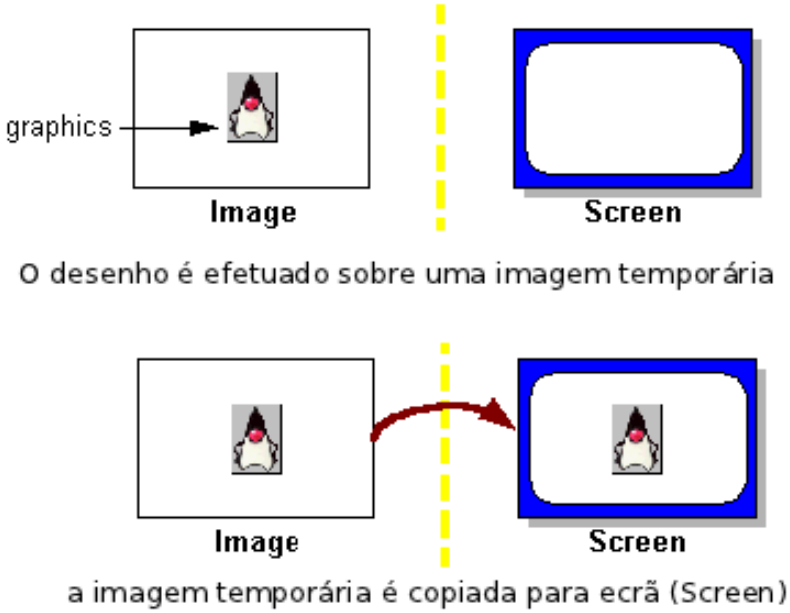


Figura 3.7: Exemplo de aplicação da técnica *double buffering*

3.3.5 Descrição do algoritmo de desenho de cada componente do PDI

Com processamento do documento *XML* e obtenção dos respetivos valores das classes dos componentes. Proceda-se ao desenho dos componentes de acordo com seu comportamento singular, pelo método *paint()*. Os componentes constituídos por imagens são desenhados pelos método *drawImage()* da *API* do *Java* que permite desenhar a imagem numa determinada posição do ecrã. Deste modo esta secção apresenta a implementação singular de cada componente apresentado no ecrã do painel de instrumentos do veículo.

background

A classe *Background* é constituído por apenas um atributo da localização da imagem de fundo do ecrã do PDI. Para o desenho dessa imagem é instanciado o objeto *File*(*Java API*) com a localização da imagem. Assim que a imagem é carregada é a imagem de fundo desenhada pelo método *drawImage()*[20] na posição (0,0) de modo

[20] `http://docs.oracle.com/javase/tutorial/2d/images/drawimage.html`

a ocupar todo o fundo do ecrã do PDI.

pointerGauge

O componente *pointerGauge* encontra-se divido em duas classes *PointerGauge* (exemplo: imagem de fundo do velocímetro) e *Pointer* (exemplo: estrutura do ponteiro do velocímetro). A classe *PointerGauge* contém as localização da imagem e posição onde qual vai ser desenhada. A segunda classe *Pointer* é composta por:

- posição da origem do ponteiro.
- posição final do valor mínimo do componente.
- posição final do valor máximo do componente.
- cor do ponteiro.
- valor mínimo e máximo da escala do componente onde o ponteiro vai ser desenhado.

O ponteiro é constituído por uma reta da *drawLine()*[21] da *API* do *Java* com origem no ponteiro de origem referido anteriormente e com ponto final no intervalo definido pela escala do componente *pointer*. Assim que ocorre uma atualização do valor do velocímetro, por exemplo no documento *XML*, é necessário recalcular a posição final do ponteiro para corresponder ao valor descrito no *XML*. Esse cálculo está na implementação no método *rotateAngle()* que dado os valores mínimos e máximos da velocidade ou rotações do motor calcula a nova posição final do ponteiro dos componentes do tipo *GaugePointer*.

O cálculo matemático do ângulo entre os dois valores mínimo e máximo do *GaugePointer* começa pela translação do ponto inicial (pI) para a origem e dos pontos finais ($p1$ *e* $p2$) dos valores mínimo e máximo da imagem do componente *GaugePointer*. Sendo *xP0 e yP0* ponto da origem do ponteiro, posição mínima *xP1, yP1* e máxima *xP2, yP2* para os valores da imagem do ponteiro respetivamente, os pontos resultantes da translação são dados por *px1, py1, px2 e py2* pela fórmula:

```
px1 = xP0 - xP1;
py1 = yP0 - yP1;
px2 = xP0 - xP2;
py2 = yP0 - yP2;
```

Após a translação procede-se ao cálculo da distância do ponto de origem (pI) ao ponto *p1* e *p2*. Assumindo ***a*** *e* ***b*** como distâncias de *p1 e p2* à origem calculadas pela

[21] `http://docs.oracle.com/javase/6/docs/api/java/awt/Graphics.html`

norma do vetor [Wikipedia, 2011c] entre *pI* e os pontos *p1* e *p2*, veja-se o exemplo na figura 3.8. E a função *squareRoot()* como raiz quadrada, as distâncias $\boldsymbol{a}$ e $\boldsymbol{b}$ normas dos vetores a e b são calculadas pelas fórmulas:

```
a = squareRoot(px1^2 + py1^2);
b = squareRoot(px2^2 + py2^2);
```

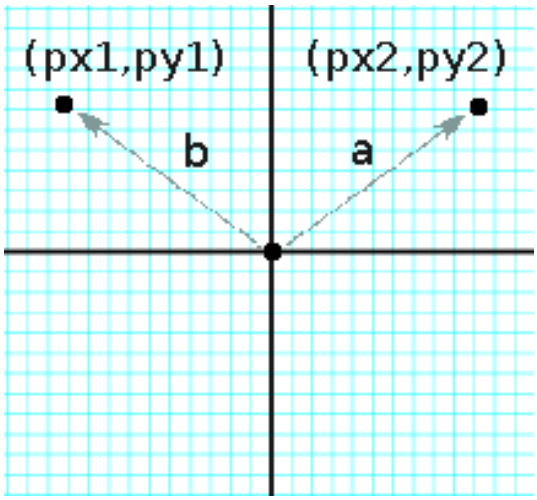


Figura 3.8: distâncias $\boldsymbol{a}$ e $\boldsymbol{b}$ da origem aos pontos *p1* e *p2*.

Concluído o cálculo dos valores dos vetores dos pontos *a e b*, segue-se a obtenção do intervalo de rotação do ponteiro entre os pontos *p1 e p2*. Esse cálculo é processado usando a fórmula do produto interno entre dois vetores para obtenção do ângulo α entre os vetores $\|\boldsymbol{a}\|$ *e* $\|\boldsymbol{b}\|$:

$$< a, b >= \|a\|.\|b\|.\cos(\alpha) \tag{3.1}$$

$$\cos(\alpha) = \frac{< a, b >}{\|a\|.\|b\|} \tag{3.2}$$

Aplicando a função inversa de cos (arccos):

$$\alpha = \arccos(\frac{< a, b >}{\|a\|.\|b\|}) \tag{3.3}$$

Produto interno entre *a e b*:

```
prod = px1*px2 + py1*py2
```

Cálculo do ângulo:

```
ang = acos(prod/(a*b))
```

Cálculo do ângulo, *ang2* entre *pI* (posição inicial) e posição corrente de $\boldsymbol{v}$:

```
ang2 = (ang/(vmax-vmin))*v;
```

Passando por estas etapas para o cálculo do ângulo de v segue-se o cálculo da posição final *(x,y)* do ponteiro em função do valor de $\boldsymbol{v}$. Para esse processo é necessário aplicar uma matriz rotação [Wikipedia, 2011e]. Os pontos (xP0,yP0) e (xP1,yP1) sofrem uma translação da origem para a sua posição original e vice-versa, resumindo v é calculado pela fórmula:

$$\begin{vmatrix} x & -xP0 \\ y & -yP0 \end{vmatrix} = \begin{vmatrix} \cos(ang2) & -\sin(ang2) \\ \sin(ang2) & \cos(ang2) \end{vmatrix} \begin{vmatrix} xP1 & -xP0 \\ yP1 & -yP0 \end{vmatrix}$$

Cálculo da posição *(x,y)* em função de v:

```
x = (xP0 + (xP1 - xP0) * cos(ang2) - (yP1 - yP0) * sin(ang2));
y = (yP0 + (xP1 - xP0) * sin(ang2) + (yP1 - yP0) * cos(ang2));
```

levelGauge

O componente *levelGauge* é formada por uma imagem e por outras imagens dentro da mesma anterior de modo a definir os vários níveis do componente. Estes níveis permitem a representação, por exemplo, da quantidade e energia existente na bateria do veículo (figura 3.9).

Figura 3.9: Constituição do componente *levelGauge*.

Este componente é formado por duas classe: *LevelGauge e ElementGauge* que definem a imagem do indicador de bateria e imagem do nível do indicador respetivamente. A classe *LevelGauge* é apenas formado pela localização da imagem a desenhada e pela posição onde a imagem vai ser representada no PDI. A *LevelElement* é constituída pela localização e posição (x,y) de desenho da imagem do primeiro nível do componente, bem como a sua largura, comprimento e número limite de *levelElements* que podem ser desenhados no interior do *LevelGauge*. As definição da quantidade de níveis (*levelEment*) foram calculadas no método *paint()*. Após o seu cálculo passa-se para o seu desenho no PDI de indicador vertical, como demonstrado na figura 3.9, para esse processo, sendo *initX* e *initY* a posição do primeiro nível do componente para os restantes são desenhados segundo as seguintes instruções:

```
numberOfLevels = (battery or fuel Percentage * maxLevels) / 100;

for(i = 0; i<numberOfLevels; i++){
desenhar levelImage na posição ( initX, initY - (alturaNivel * i) )}
```

maxLevels - número máximo de níveis do componente.
numberOfLevels - quantidade de nível do componente nesse instante.
levelImage - imagem do nível.
alturaNivel - altura da imagem do nível do componente.
initX e initY - posição x e y da imagem do nível do componente.

indicator

Componente de apenas dois estados: ativo e desligado ou por outras palavras uma luz acesa ou apagada. Para cada estado tem uma imagem associada, ou seja, conforme a indicação de mudança de estado procede-se à troca da atual imagem por uma nova imagem. A classe *indicador* é formada por: duas imagens (estado ativo ou inativo), posição de desenho e por um valor booleano que indica qual é o estado inicial do componente tipo *indicador* e assim desenhar a imagem correspondente.

staticImage

O componente *staticImage* determina o desenho de imagem desprovida de qualquer ação especifica . A classe respetiva é formada pela localização da imagem e respetiva posição de desenho para o ecrã do PDI.

label

A componente *Label* permite o desenho de porções de texto no PDI. A classe associada *Label* é formada pelo texto a ser desenhado bem como, o estilo, tamanho e cor da fonte da letra do texto associado. Além dos atributos atrás referidos contém um outro atributo de nome *refreshable* que permite especificar se é um texto estático a ser desenhado ou dinâmico. Se o tipo atributo *refreshable* tiver valor *false* não necessita de ser atualizado a cada iteração do método *paint()*, bastando apenas chamar um função *drawString()*[22] do *Java API* para colocar a *label* na posição desejada. Se valor de *refreshable* for *true* a cada iteração do *paint()* a *label* é atualizada. A atualização foi otimizada de forma a evitar o redesenho de todo ecrã. Assim, são utilizados o método *getSubimage()* da classe *Graphics*[23] e os métodos *getHeight()* e *getWidth()* da classe *FontMetrics*[24] da *Java API*. Com métodos *getHeigth()* e *getWidth()* obtém-se a altura e largura, respetivamente, da imagem ocupada pela *label* no ecrã do PDI. De Seguida executa-se o método *getSubimage()* para obter uma sub-imagem formada pelos componentes que estão dentro do espaço ocupado pelo componente *label* no ecrã. A sub-imagem obtida é desenhada na posição de desenho da *label* retirando assim a antiga *label* anteriormente desenha na mesma posição e por fim colocando a nova *label* pelo método drawString() na posição pretendida, demonstrado pela figura 3.10.

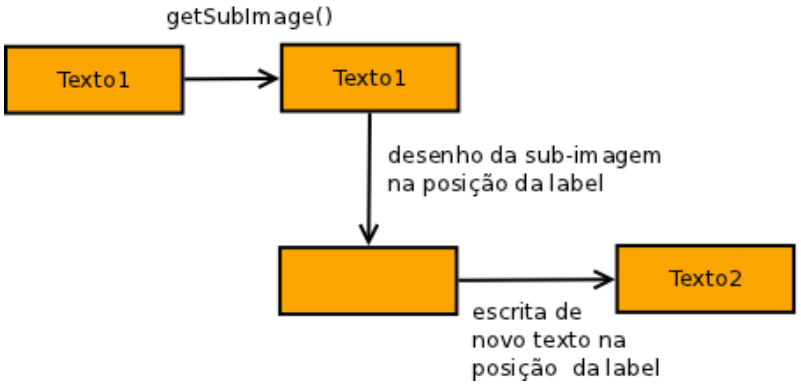


Figura 3.10: Esquema de atualização do componente *label*.

3.4 Ambiente de desenvolvimento

Um objetivos mais importantes enunciados para construção de um painel de instrumentos, referido na secção 3.1 é a construção de uma relação entre as aéreas diferentes como programação e *design*. Estas duas áreas vão assegurar a definição

[22] http://docs.oracle.com/javase/1.4.2/docs/api/java/awt/Graphics2D.html
[23] http://docs.oracle.com/javase/6/docs/api/java/awt/Graphics.html
[24] http://docs.oracle.com/javase/1.4.2/docs/api/java/awt/FontMetrics.html

do PDI pelos mãos dos respetivos especialistas de cada área. A definição do ambiente para desenvolvimento assenta a separação da estrutura gráfica da parte da programação de comportamento dos componentes alojados no PDI.

A separação desta duas áreas de desenvolvimento é feita por dois tipos de grupos, grupo de programadores e o grupo de *designers*. Os ***designers*** responsáveis pelo aspeto gráfico do PDI e os **programadores** pela programações funcionamento dos componentes gráficos do PDI. Quando um dos grupos pretender desenvolver algum componente ou funcionalidade para o PDI através de algum dos ambientes de desenvolvidos nesta dissertação, está assegurado que, quer um grupo, quer outro, podem desenvolver separadamente os componentes sem depender um do outro grupo.

A utilização do meio de desenvolvimento para *designers* consiste apenas na preocupação de criar componentes para o PDI. A disposição e configuração desses mesmos componentes é assegurada pelo documento *XML* definido pelo *designer* segundo as normas de escrita do documento *XML* para os dois tipos de grupos de desenvolvimento. No processo de adição de componentes ao PDI pelo *design* tem de inserir no documento *XML* a *tags* identificadora do componente em questão e finalizar a definição do mesmo componente com o resto das *tags* das propriedades associadas.

Após a definição do documento *XML* entra em jogo o papel do programador que apenas tem como diretivas: a leitura do documento *XML* de definições dos componentes gráficos, representação dos componentes pré-definidos no *XML* e programação das ações dos componentes gráficos que são executados durante o funcionamento do PDI.

3.4.1 Exemplo de criação de um componente para um PDI

Uma equipa de trabalho constituída por um programador e *designer* tem como objetivo criar um componente para um PDI. Em que esse componente poderia ser por exemplo um indicador de porta aberta do veículo. Sempre que existe uma porta estiver aberta, uma luz no PDI acende a indicar o problema ao condutor. Para o desenvolvimento o ***designer*** começa por:

1. O indicador é para ser desenhado na posição (50,100) do ecrã do PDI.
2. Quando não existem problemas relacionados com uma ou mais portas abertas deve surgir a imagem *portaOff.png*, figura 3.11.

Figura 3.11: Indicador de porta aberta do veículo no estado desligado.

3. Caso surja algum problema deve "aceder"a imagem *portaOn.png*, figura 3.12.

Figura 3.12: Indicador de porta aberta do veículo no estado ligado.

A parte desenvolvida pelo *designer* quando definiu o componente fez a especificação dos conteúdos: imagens e posição de desenho no documento *XML* com a seguinte sintaxe:

$$
\begin{gathered}
< indicator > \\
< imageActive > portaOn.png < /imageActive > \\
< imageInactive > portaOff.png < /imageInactive > \\
< px > 50 < /px > \\
< py > 100 < /py > \\
< /indicator >
\end{gathered}
$$

Para o **programador** cabe-lhe fazer o seguinte:

1. Leitura do documento *XML* para saber que imagens precisa e que posição vai ser desenhada.
2. Ler a informação do sensor do motor.
3. Conforme o valor do sensor, o programador programa a troca da imagem *portaOff.png* ou para *portaOn.png*, exemplo:

```
.........

// se componente for a imagem da porta
// troca a imagem atual pela outra imagem,
if (comp instanceof Notifier) {
        Notifier not = (Notifier) comp;
        BufferedImage image = null;

                try {
                String imageName = "";

                if (not.getActive().equals("portaOn.png")) {
                        if (portaActive)
                                imageName = "portaOn.png";
```

```
                                    else
                                     imageName = "portaOff.png";
.....

// Desenhar a imagem na posicao (x,y)
int x = not.getX();
int y = not.getY();
gTemp.drawImage(image, x, y, null);
```

Capítulo 4

Proposta para um sistema de navegação

Este capítulo está dividido em quatro secções. A primeira faz o enquadramento da proposta da interface. A segunda descreve estrutura da interface proposta. A terceira apresenta a implementação da interface. A quarta última apresenta um tutorial de como embeber um *browser* numa aplicação *Java*.

4.1 Visão geral do sistema

Atualmente todas atividades rodoviárias têm como utensílios diários sistemas de navegação por satélite. Nestas atividades as pessoas cada vez mais não passam sem usar os SNS [Mascarenhas, 2010]. Estes sistemas (GPS) no início eram de uso exclusivo militar norte americano passando mais tarde para uso [Baroni, 2009] civil. A qualidade da ligação aos satélites praticamente ininterrupta em qualquer tipo de condições atmosféricas e interferências, tornou-se numa ferramenta indispensável na condução rodoviária.

De forma a complementar a PDI proposta criou-se um sistema de navegação por satélite de auxílio na navegação rodoviária ao condutores do veículo elétrico. O sistema de navegação por satélite (SNS) permite ao utilizador navegar da sua localização atual para o destino pretendido, modo de viagem mais rápida (duração mais curta) ou viagem mais curta (duração maior), com direções passo a passo, tempo estimado para chegada ao destino e velocidade instantânea. A aplicação tem como função de navegação: a apresentação de direções para o destino inserido no *popup*

da aplicação.

4.2 Arquitetura da aplicação

A arquitetura da aplicação está divido em duas arquitetura: a primeira aborda o funcionamento global da aplicação e a segunda parte retrata a arquitetura da interface gráfica da aplicação.

4.2.1 Arquitetura global do sistema

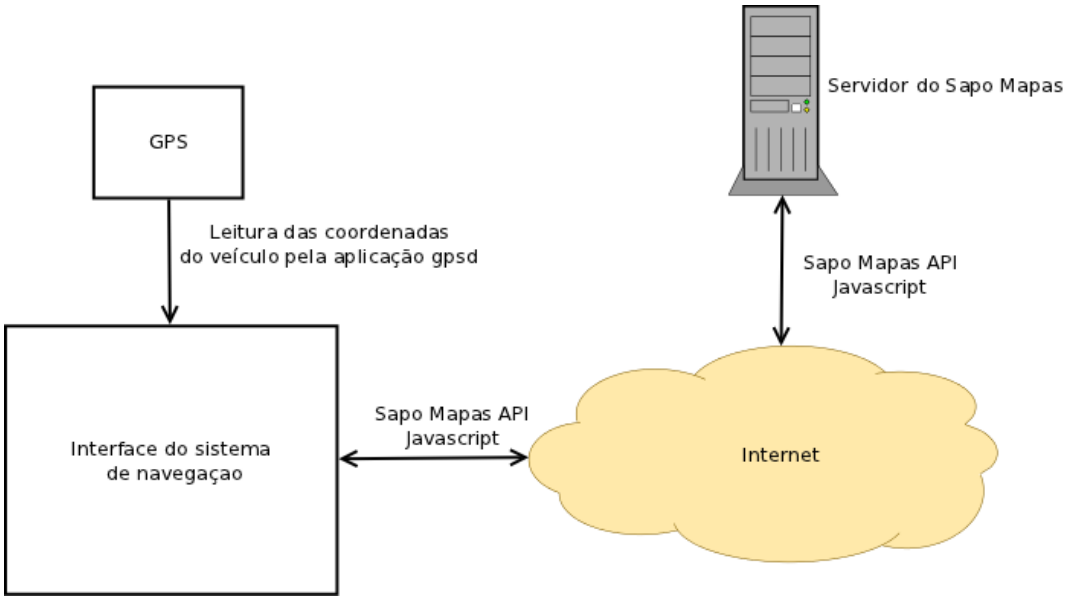


Figura 4.1: Arquitetura global do sistema de navegação por satélite

O sistema de navegação por satélite (SNS) comunica com servidor do *sapo* mapas[1] para obtenção dos mapas pela Sapo Mapas *API*[2] através da linguagem de *script Javascript*[3], como demonstrado na figura 4.1. Para encontrar a encontrar a localização do veículo durante a condução a aplicação do *SNS* comunica com serviço *gpsd daemon*[4] que estabelece a ligação *TCP*[5] à porta *USB serial* onde está ligado um equipamento *GPS*. A aplicação SNS após obter os valores do *GPS*, começa por obter o mapas através da *internet* pela *API* do sapo mostrando nesse mapa a atual localização do utilizador.

[1] `http://mapas.sapo.pt/`
[2] `http://api.mapas.sapo.pt/`
[3] `https://developer.mozilla.org/en/JavaScript`
[4] um *daemon* é um programa executado em segundo plano que fornece um serviço. `http://linux.about.com/cs/linux101/g/daemonlparservi.htm`
[5] protocolo de comunicação *internet* [Wikipedia, 2011g].

4.2.2 Arquitetura da interface gráfica do SNS

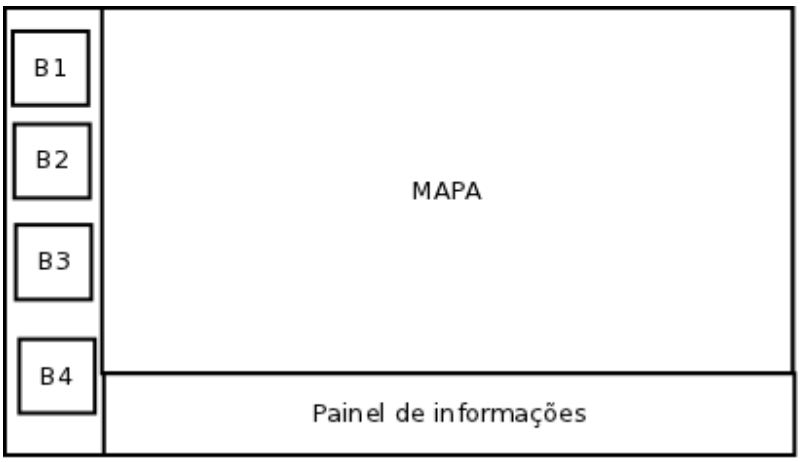


Figura 4.2: Esquema da interface do sistema de navegação

A interface do SNS está divida em três grupos de informações distintos (ver figura 4.2): o painel de visualização do mapa, painel dos botões de ações e um painel de informações descritivas. O painel onde é desenhado o mapa é um ***browser***[6] onde são executados os *scripts* da *API* do sapo mapas para manipulação dos mapas. No painel dos botões estão desenhados quatro botões cada uma com a uma ação, dois botões para fazer *zoom*, um para selecionar o destino pretendido e um último para definições da aplicação. Quanto ao painel de informações podemos visualizar a velocidade instantânea fornecida pelo *GPS*, a distância e tempo que falta para o destino e a próxima instrução de mudança de direção.

4.3 Implementação

Para a construção da aplicação foram usadas as seguintes tecnologias: *Java*, *SWT*, Sapo Mapas e *Javascript*.

4.3.1 Tecnologias usadas

Java

A implementação do SNS foi em *Java* para comunicação com o equipamento com o *GPS* e com sapo mapas pelo *widget* do eclipse que permite embeber um *browser* dentro de aplicações nativas do *Java* e para a criação da interface gráfica do sistema. As características do *Java* permitem que esta aplicação possa ser executada

[6]"Programa que permite aceder a páginas e a sítios da Internet e aos recursos neles disponibilizados". fonte:`http://www.priberam.pt/dlpo/dlpo.aspx?pal=browser`

em qualquer sistema operativo com *Java virtual machine* é uma das razões fundamentais para o êxito do *Java* e uma mais valia para a aplicação. Além desta valia anteriormente enunciada é de salientar *multi-threaded* para aplicações e o *garbage collection* que liberta sempre que possível memória não utilizada pelo programa, já anteriormente enunciado no capítulo 3.3.1. Para a interface gráfica foi usado o *Java Swing* pela conjunto de ferramentas oferecidas tais como o *JFrame*, *JPanel*, *JLabel*, *JButton* e *canvas* (providenciado pela classe *Graphics*). *JFrame* permite definir o tamanho da janela onde vão ser adicionados os outros componentes gráficos do *Java Swing* [Oracle, 2011e]. *O JPanel* funciona como contentor dos componentes gráficos do *Java* ou de componentes desenhos através do *canvas* [Oracle, 2011h]. A classe *JLabel* permite desenhar texto e imagens estáticos (sem animação) [Oracle, 2011g]. Quanto ao *canvas* a sua função principal é desenhar uma área branca no ecrã onde se pode desenhar componentes ou intercetar eventos do utilizador [Oracle, 2011a]. A classe *JButton* é componente formado por um botão e tem como objetivo ser premido para desencadear uma ação programada [Oracle, 2011f]. A cada botão desenhado na interface vai ter uma ação programada, essa ação quando o botão é premido é captada e executa a respetiva ação pela interface do *ActionListener* [Oracle, 2011i].

SWT (Standard Widget Toolkit) - Browser widget

SWT[7] é uma biblioteca gráfica *Java* criada pela *IBM*[8] e mantida atualmente pela *Eclipse Foundation*[9] que contém o componente *browser widget*[10] que permite integrar um *browser* dentro de uma aplicação gráfica nativa de *Java* [Alves, 2009]. O *widget* permite a execução de *Javascript* pelos métodos da classe *Browser* da *API SWT*. As *widgets SWT* são compatíveis para todos sistemas operativos que tenham *Java instalado* porque esta biblioteca acede a componentes nativos do *Java* de forma a manter a portabilidade do código, *SWT* apresenta-se como alternativa ao *Java Swing* contando com uma grande variedade quantidade de *widdgets*[11].

O serviço *gpsd*[12] é uma aplicação executada em segundo plano e tem como função de receber e monitoriza os dados provenientes dos recetores *GPS*. Apresentando-se como uma interface de comunicação para variados tipos de *GPS's* eliminando assim a heterogeneidade de cada equipamento. A interface *gpsd* estabelece uma ligação *TCP/IP* pela porta 2947 à porta *USB* na qual o *GPS* está ligado. Os pedidos de informações geográficos são efetuados pelo protocolo *TCP* e as respostas vêm na sintaxe *JSON*[13] que depois são processadas pela aplicação SNS.

[7] componente da interface gráfica responsável pela interação com o utilizador. [Steve Northover, 2004]

[8] `http://www.ibm.com/`

[9] `http://www.eclipse.org/swt/`

[10] `http://help.eclipse.org/indigo/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/swt/browser/Browser.html`

[11] `http://www.eclipse.org/swt/widgets/`

[12] `http://www.catb.org/gpsd/`

[13] `http://www.json.org/`

Sapo Mapas

O Sapo Mapas é uma ferramenta que permite visualizar os mapas de Portugal gratuitamente. Esta aplicação *web* tem refrescamento assíncrono pelas chamadas das funções *Javascript* da *API* do Sapo mapas evitando o recarregar de toda a página *html*. A *API* permite usar e programar as opções de visualização do mapa, eventos, interações do utilizador com mapa e adição e remoção de conteúdo do mapa pelas funções *Javascript* da *API*. Com a programação dos mapas do Sapo *API* o programador não precisa de se preocupar com os fatores de transferência de dados e nem com a gestão de pedidos de informação e tratamento de respostas [Silva, 2010]. Em termos de detalhes geográficos e rigor das estradas o sapo mapas no contexto português está à frente do concorrente *google maps*[14], como descrito por este autor [Fernandes, 2008] e pela comparação de imagens retiradas da mesma localidade por serviços de mapas diferentes, *Google Maps* e Sapo Mapas, figura 4.3. Como pode-se constatar pelas imagens a escolha no serviço mapas recaiu na solução nacional que tem maior cobertura de redes de estradas.

Javascript

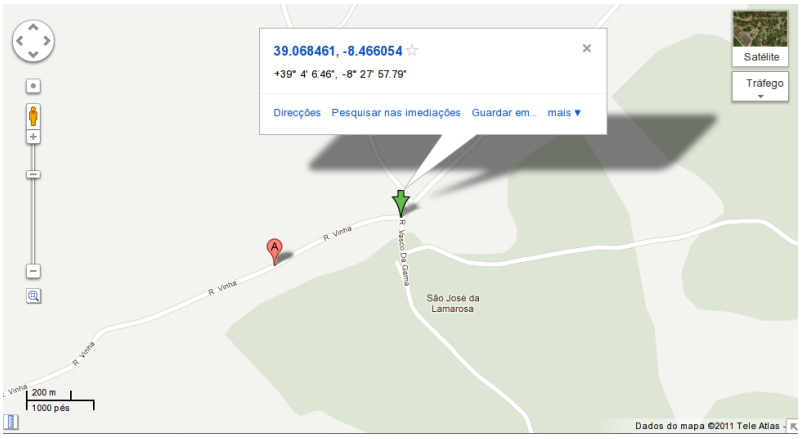
A linguagem de *script* de nome *JavaScript* é dinâmica, de tipagem fraca, imperativa e funcional. *JavaScript* enquanto linguagem foi influenciada pela linguagem *ECMAScript*[15] padrão e é utilizado principalmente em aplicações do lado cliente, implementado como parte integrante do navegador Web, interfaces *web* dinâmicas para utilizadores e sites mais dinâmicos. Na implementação do sistema SNS esta linguagem é usada na chamada de funções da *API* do sapo mapas.

4.3.2 Embeber *SWT Browser Widget* numa aplicação *Java Swing*

O *Java Swing* não tem um *widget* para visualizar páginas *html* para tal recorreu-se ao *SWT Browser Widget* que possui essa *widget* essencial para apresentar os mapas das estradas. Como *Java Swing* e *SWT* são duas bibliotecas gráficas independentes entre si e apenas têm como camada de baixo nível comum *Abstract Window Toolkit (AWT)* (API gráfica original do Java) a sua retro compatibilidade é assegurada pela *AWT*. Para embeber o *browser SWT* escolheu a classe *Canvas AWT* que permite o desenho do ecrã tipo *SWT* no interior da aplicação *Java Swing*

A classe *CanvasBrowser* do SNS estende a classe *Canvas AWT* de modo a definir o *Browser* como componente *Canvas* para poder ser adicionado à Janela *JFrame do Java Swing* [Feigenbaum, 2006] [Foundation, 2011] [Tapken, 2011]. A classe *CanvasBrowser* é constituído pelas classes *Display* e *Shell*. A classe *Display* permite

[14] `http://maps.google.pt/`
[15] `http://www.ecmascript.org/`



Google mapas na coordenada **39.068461, -8.466054**.

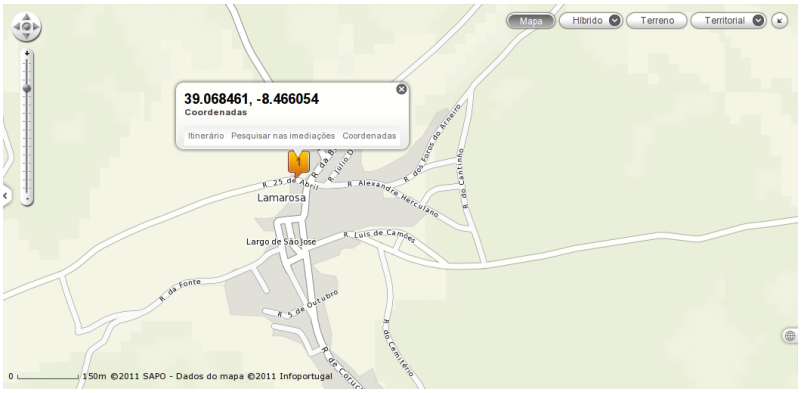


Figura 4.3: Sapo mapas na coordenada **39.068461, -8.466054**

criar a janela da aplicação *SWT* à semelhança do *JFrame* e assegura a comunicação entre a *API SWT* e o sistema operativo subjacente. A sua função principal é gerir e criar eventos associadas à aplicação e recorrer aos recursos da *API* do sistema operativo. A classe *Shell* representa as janelas instanciadas pela classe *Display* e faz a sua gestão de eventos e ciclo de vida [Steve Northover, 2004].

```
Display display = new Display();
Shell shell = SWT_AWT.new_Shell(display, canvas);
```

Definição da classe *CanvasBrowser* do *Browser Widget* como componente *Canvas*. A integração do *browser* finaliza-se com adição do *browser canvas* ao JPanel da aplicação SNS.

4.3.3 Inicialização do sapo mapas com a localização atual

A inicialização do sapo mapas é feita pela introdução do código *html* (***String homepage***) e ***JavaScrit*** (função *init()* presente na *String homepage*) da API *Sapo Ma-*

pas ao *browser SWT*. A inicialização é efetuada no construtor da classe *GpsGui* e é definido a uma *String* de código *html* para a página principal do **browser**.

```
String homepage =

+ "function init(){"
+ "map = new SAPO.Maps.Map('mapDiv');""
+ "map.zoomTo(" + Integer.valueOf(zoom) + ");"
+ "var home= new OpenLayers.LonLat(" + getLog() + ","
+ getLat() + ");"
+ "map.setMapCenter(home);"
```

A função *JavaScript init()* define uma nova instância do mapa da sapo com um determinado *zoom* e centrado pelas coordenadas da posição veículo dadas pelas variáveis: *getLog()* valor da longitude e *getLat()* valor da latitude.

```
String homepage;
canvasBrowser.getBrowser().setText(homepage);
```

Após a definição da *homepage* executa a função *setText() da API SWT* que dado a *string homepage* renderiza o mapa no componente *Browser*.

4.3.4 Descrição dos painéis da interface SNS

Como descrito no secção 4.2.2 esta interface está agrupada em três painéis de componentes: um painel do elemento do mapa descrito na secção anterior, um painel de botões com respetivas funções e um painel de informações informativas.

As funções *Javascript* executadas após carregamento da página *html* são chamadas pela função *execute(script)* do *browser SWT* que recebe como argumento o código *script* a ser executado.

Painel de botões

Zoom out ao ser premido este botão desencadeia a função *script map.zoomOut()*[16] da *API* do Sapo Mapas que diminui o nível de *zoom* que está compreendido numa escala numérica de 4 a 19.

Zoom in executa a função *Javascript map.zoomIn()* que aumenta o nível de *zoom* do mapa.

Destino reproduz a função *Javascript getDestination()* que apresenta uma pequena janela para inserir o destino. Essa janela é uma *popup*[17] do *JavaScript* que per-

[16] `http://api.mapas.sapo.pt/ClasseSapo.Map.php`
[17] `http://www.w3schools.com/js/js_popup.asp`

mite a introdução do destino pretendido. De seguida pela função *getSearch()* da *API* do sapo devolve o ponto de coordenadas do destino.

```
canvas1.getBrowser().evaluate("return destination();").
toString())
```

Após a determinação do destino procede a uma apresentação no mapa e cálculo do percurso entre os pontos de origem e destino, passando depois para um *Zoom* próximo da sua atual localização.

A cálculo do percurso é implementado pela função da *getItinerary() API* do *sapo mapas* descrito no código abaixo.

```
canvas1.getBrowser().
        execute("itinerary.getItinerary('" + from + "','"
    + getDestination()
    + "', {mode: '" + getModoViagem() + "'});"));
```

from - posição atual do utilizador fornecida pelo *GPS*.
getDestination() - função recebe a *String* do destino proveniente da *popup* anterior referida.
getModoViagem() - *String* do modo viagem (curto ou mais rápido) por defeito está definido o percurso mais rápido.

Preferências - botão para definir o modo percurso pretendido (rápido ou curto) e visualizações de informações a cerca do *GPS*.

Painel de informações

O painel informativo situado no rodapé da *interface* é constituído por quatro *JLabel* e que apresentação as seguintes informações: velocidade instantânea, distância em falta até à chegada ao destino, o tempo que falta para chegar e próxima direção rodoviária que condutor tem que tomar.

Velocidade é obtida pela informação proveniente do *GPS*.

Distância é processada pelo código:

```
Double dist = ((Double) canvas1.getBrowser().
        evaluate("return itinerary.getDirections().
        getRoute().getDistance();"));
```

Dist - variável que guarda em memória a distância do resultado do cálculo do percurso.
GetDistance() - obtém a distância do percurso.

Duração - devolve a duração do percurso com recurso à função do sapo: *return itinerary.getDirections().getRoute().getDistance()*.

Direção - lê o primeiro passo da lista de passos gerados durante o cálculo do percurso.

4.4 Tutorial para criação de um *browser widget* numa aplicação *Java Desktop* em *Linux*

Importação da biblioteca gráfica *SWT*

A criação da classe onde vamos correr o *browser* necessita de importar as classes da biblioteca *SWT*:

1. descarrega-se o *jar* de nome *swt.jar* do site `http://www.eclipse.org/swt/`

2. procede-se à definição do cabeçalho da nossa classe com *imports* das classes:

```
import org.eclipse.swt.SWT;
import org.eclipse.swt.SWTError;
import org.eclipse.swt.browser.Browser;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
```

   Browser - *widget* que fornece um *browser para aceder à internet.*
   Display - gere as ligações entre *SWT* e sistema operativo.
   Shell - representa as janelas criadas pelo sistema operativo.
   SWTError - providencia um grupo de classes para tratamento de erros e exceções.

3. na compilação executa-se o comando *javac -cp swt-3.7.1-gtk-linux-x86/swt.jar BrowserClass.java* que compila o código fonte segundo a biblioteca externa do ficheiro *swt.jar*.

Implementação

1. defini-se o método *main* da classe *BrowserClass*, na qual vai estar toda a implementação do *browser widget*.

2. instanciação das classes *Shell e Display* para criar a janela onde vai ser apresentado o *browser*:

```
Display display = new Display();
Shell shell = new Shell(display);
```

3. instanciação da classe *Browser* dentro do contentor classe *Shell*:

   ```
   Browser browser = new Browser(shell, SWT.NONE);
   ```

4. definição do tamanho do ecrã do *Browser*:

   ```
   browser.setSize(shell.getSize());
   ```

5. inserção do *url* para ser renderizado pelo *Browser*:

   ```
   browser.setUrl("google.com");
   ```

6. para executar o exemplo basta usar o comando *java -cp swt-3.7.1-gtk-linux-x86/swt.jar: BrowserClass*.

7. *browser* pronto a navegar para qualquer *site* pela função *browser.setUrl("**Website URL**")*.

Código-fonte do exemplo

Apresenta-se abaixo o código-fonte do exemplo descrito.

```
import org.eclipse.swt.SWT;
import org.eclipse.swt.SWTError;
import org.eclipse.swt.browser.Browser;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;

public class BrowserClass {

    public static void main(String[] args) {

        System.setProperty("sun.awt.xembedserver", "true");

        Display display = new Display();
        Shell shell = new Shell(display);

        try {
            Browser browser = new Browser(shell, SWT.NONE);
            browser.setSize(shell.getSize());
            browser.setUrl("google.com");
        } catch (SWTError e) {
```

```
            e.printStackTrace();
        }

        shell.open();

        while (!shell.isDisposed()) {
            if (!display.readAndDispatch()) {
                display.sleep();
            }
        }

        display.dispose();

    }
}
```

Capítulo 5

Casos de estudo

Este capítulo está dividido em quatro secções. A primeira descreve a constituição do painel de instrumentos proposto. A segunda apresenta a interface do PDI proposto e avaliação de desempenho. A terceira apresenta a interface do SNS. A quarta última é avaliação dos princípios para apresentação de informação nas interfaces do PDI e SNS.

5.1 Constituição do painel de instrumentos

Os componentes que estão incluídos no painel de instrumentos do veículo *Gecko Merula* são:

- Velocímetro.
- Tacómetro.
- Indicador do nível de bateria do veículo.
- Indicador de travão de mão.
- Indicador de aviso de abertura de portas.
- Indicador de luzes de presença (mínimos).
- Indicador de luzes de cruzamento (médios).
- Indicador de luzes de estrada (máximos).

- Indicador de luzes de nevoeiro.
- 2 indicadores de luzes indicadoras de direção.
- Indicador de bateria fraca.

5.2 A interface do painel de instrumentos

A interface do PDI veículo *Gecko Merula* foi desenhada numa resolução de 1280x800 pixeis. A interface foi desenhado numa aplicação *Java Desktop*. A configuração gráfica foi toda especificada no documento *XML* que pode ser consultado no anexo na página A. A interface proposta pode ser consultada na figura 5.1.

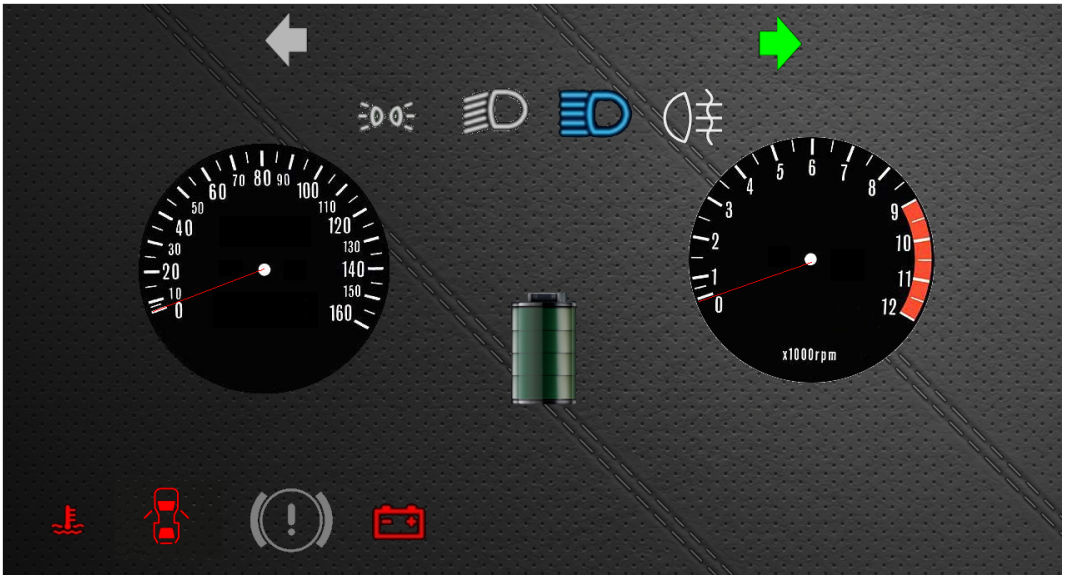


Figura 5.1: Painel de instrumentos para veículo elétrico *Gecko Merula*

Análise de desempenho da interface

Para análise de desempenho foi usado o portátil *Insys* de dois processadores *Intel(R) Core(TM)2 Duo CPU T6600* a 2.20GHz cada um, memória RAM de 4GB, placa gráfica da *NVDIA GeForce G 105M*, com sistema operativo Ubuntu[1] 11.04 e a versão 1.6.0.26 do *SUN JDK Java* instalado na máquina. São executados testes de desempenho sobre a quantidade de elementos desenhados e o tempo que demoram a ser apresentados no ecrã, bem como a memória necessária para a sua execução. Para os testes foram incluídos os componentes do PDI: *PointerGauge, levelGauge, indicator, staticImage, label*. No teste vão ser testadas cinco interfaces constituídos cada

[1] `http://www.ubuntu.com/`

uma com 1, 10, 20, 40 e 60 componentes do tipo anteriormente referido. A primeira interface é formada por 5 elementos de 1 tipo de cada componente. Enquanto a última é formada por 300 componentes no total repartidos pelos 5 componentes.

As medições temporais médias realizadas são expressas em milissegundos (ms) por cada iteração da função *paint()* no desenho de todos componentes do PDI. A média é efetuada ao fim de 300 iterações de desenho e o tempo gasto das mesmas. Os resultados dessas medições podem ser consultados na tabela 5.2 e no gráfico 5.3.

N.º de componentes De cada tipo	Numero total De elementos	Média de tempo De desenho (ms)	Média de memória Ram consumida (MB)
1	5	22	41
10	50	126	51
20	100	238	75
40	200	452	81
60	300	886	95

Figura 5.2: Tabelas de dados obtidos do desempenho da interface.

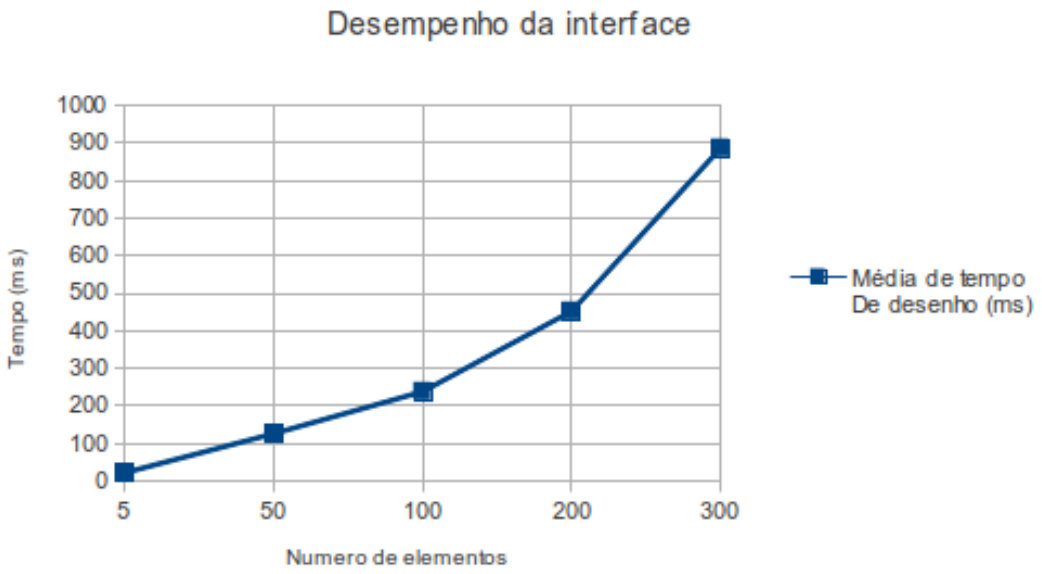


Figura 5.3: Gráfico de desempenho da interface.

Os resultados apresentados indicam que o tempo médio de desenho cresce de forma linear com número de componentes desenhados. Em relação à memória consumida apresentada na figura 5.4, o gasto de memória é proporcional ao número de elementos desenhados.

Os resultados indicam que um aumento do número de elementos no ecrã aumenta o tempo no desenho das mesmas. Em relação ao tempo de desenho é satisfatório no pior caso a interface é desenhada em menos de segundo neste ambiente de testes.

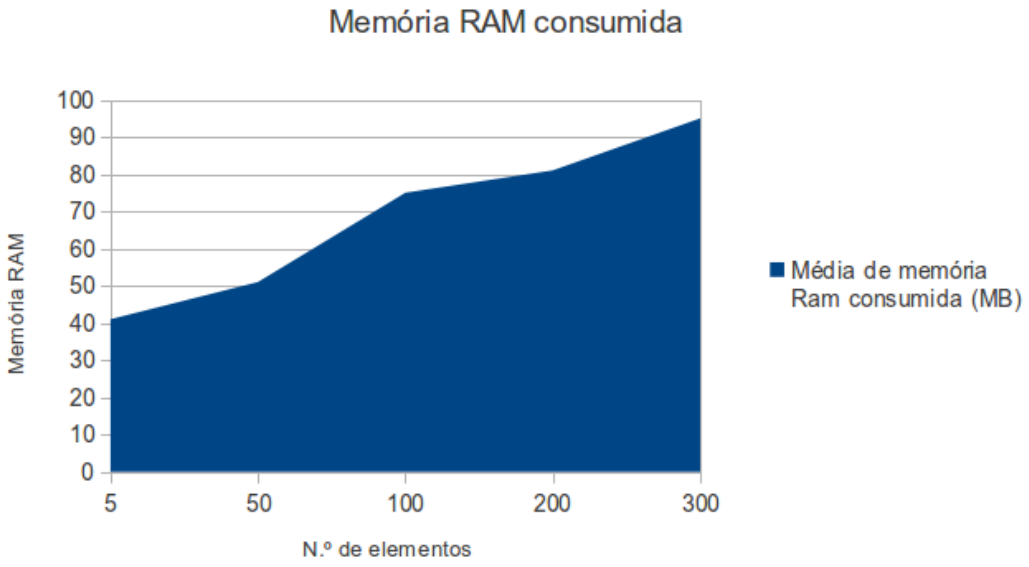


Figura 5.4: Gráfico de memória consumida.

5.3 Interface do sistema de navegação

A aplicação do sistema navegação ao iniciar mostra automaticamente no mapa a nosso posição atual pelo sinal de GPS. É constituído por botões de dimensões razoáveis para que utilizador não se distrai demasiado na interação com SNS, figura 5.5.

A interação com a interface é efetuada pelos botões à esquerda da interface que permite ao utilizador uma melhor proximidade com os botões. Os botões do *Zoom* permitem ampliar ou reduzir a visão do mapa. O botão direções permite inserir o destino para onde utilizador pretende ir. O último botão permite configurar o modo de viagem entre o trajeto mais rápido ou mais curto. Na parte inferior da interface estão situados dados relativos ao trajeto do mapa. Nomeadamente a que distância, tempo, próxima direção a tomar e velocidade instantânea a que circula.

5.4 Apreciação das informações apresentadas

Em seguida apresenta-se a avaliação da interface do painel de instrumentos e do sistema navegação por satélite segundo as normas de apresentação de informações enunciadas no capítulo 2.2:

1. **Interface deve ser desenhada de forma minimizar o número de vezes que utilizador tem de olhar para o ecrã da interface**

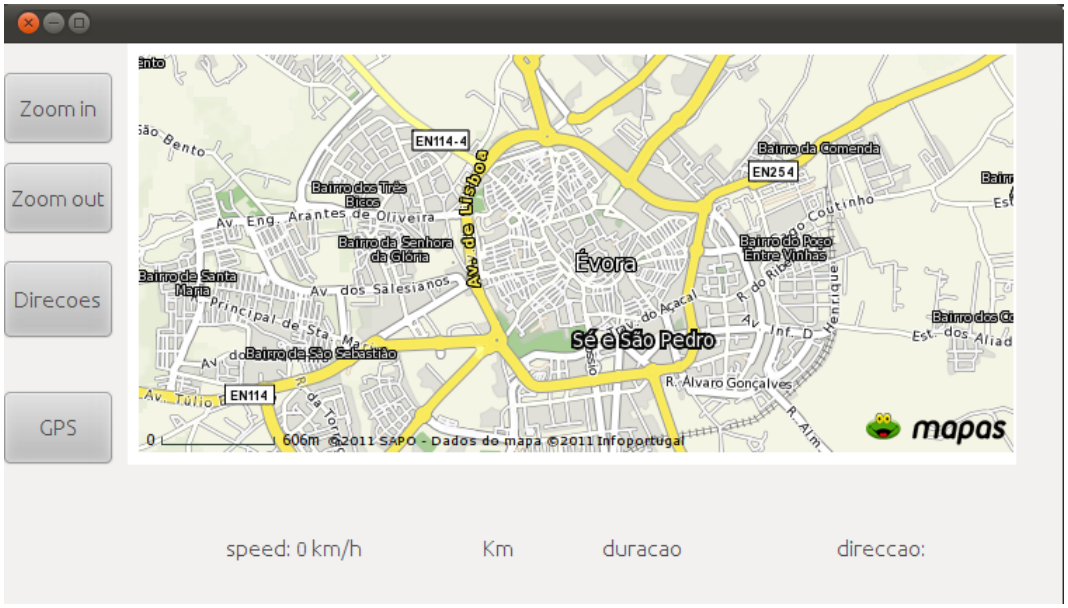


Figura 5.5: Interface do sistema de navegação por satélite.

A verificação desta norma **não é verificada** nas duas interfaces. A sua avaliação tem de ser efetuada em ambientes reais de condução ou simuladores de condução que permita a sua medição através *eye tracking*[2]. A utilização do *eye tracking* será fundamental na avaliação da qualidade da interface, já que, pode-se quantificar a quantidade de relances do utilizador necessita para executar um determinada sequência de passos para utilização da interface.

2. **Usa de normas de legibilidade em ícones, símbolos, palavras, acrónimos ou abreviações**

   O objetivo desta norma é verificar se as informações apresentadas estão de acordo com legibilidade. A informação apresentada na interface deve ser legível para que o utilizador tenha a ótima perceção de leitura das informações apresentadas. A interface do PDI utiliza o conjunto símbolos padrão segundo as normas dos símbolos rodoviários [UMTRI, 2002]. Com uso dos símbolos padrão a **norma verifica-se** para a interface PDI.

   No caso da interface do SNS os símbolos usados não estão presentes na norma sobre os símbolos. Porém considera-se que as informações apresentadas, símbolos usados não prejudiquem a sua legibilidade perante o utilizador. Os símbolos e informações apresentados na interface são de rápida e lógica compreensão e legibilidade por parte do utilizador e estão contextualizados no ambiente de navegação por satélite. Dentro do requisitos da norma apresentada considera-se esta como **verificada** nesta interface.

[2] `http://en.wikipedia.org/wiki/Eye_tracking`

3. **Apresentação precisa e antecipada de informação**

   As informações de influência para a condução devem chegar ao condutor de forma coerente e antecipada. De forma a permitir um mudança do estilo de condução conforme a informação apresentada. A quantidade de informações apresentadas deve ser adequada às normas anteriormente referidas. Para a interface do PDI a norma é **verificada** pela apresentação das informações indispensáveis para condução e de informações sobre o estado do veículo em quantidades adequadas.

   As informações apresentadas pelo sistema de navegação por satélite dizem respeito a informações geográficos para condução do veículo. As informações apresentadas na interface são relativas a direções a tomar pelo condutor para chegar a um determinado destino e são apenas visualizadas se a aplicação de navegação estiver a navegar para o destino inserido pelo condutor. Caso o sistema não esteja a navegar, as direções a tomar na condução não são apresentadas. As informações sobre as direções a tomar são apresentadas passo a passo para não sobrecarregar de informação o condutor. Princípio **verifica-se** nesta interface.

4. **O sistema deve evitar a interação com as duas mãos**

   Nenhuma das ações ou interações não deve levar o condutor a retirar as duas mãos do volante, de modo a evitar qualquer tipo de atitude de risco para a segurança rodoviária. A inserção ou consulta de dados na interface deve ser feita apenas com uma das mãos mantendo a outra segura ao volante.

   O PDI do veículo elétrico não tem comandos de inserção de informação na interface por isso a norma é **verificada**, não existe atitudes de risco por parte deste tipo de ação.

   Quanto ao SNS existem os seguintes modos de interação com sistema: premir os botões de ações *zoom* sobre o mapa e inserção de texto do nome da localidade de destino. Destas ações a inserção de texto requer maior exigência por parte do condutor. O número de toques no ecrã fique dependente da extensão de carateres do destino, mas quer para um nome grande ou pequeno a inserção pode ser suspensa ou retomada por parte do condutor sempre não houver risco para a segurança rodoviária. A norma é **verificada**.

5. **O sistema não deve impor uma sequência de passos num determinado período de tempo para interação com utilizador**

   O objetivo desta norma é evitar que o sistema provoque no condutor situações de risco na interação com o sistema. Ou por outras palavras o sistema deve conter um número pequenos de passos para as interações e permitir que as mesmas possam ser retomadas assim que o condutor desejar.

Como na norma anterior o PDI não tem qualquer interação manual com condutor, assim não viola os princípios da norma, **verificando a norma**.

No SNS a interação que exige a maior sequência de passos é a inserção do destino para onde o condutor pretende navegar como descrito na norma anterior. A inserção do nome do destino independentemente do número de carateres do mesmo, o utilizador pode retomar a escrita interrompida sempre que desejar sem qualquer exigência temporal da aplicação. Princípio foi **verificado**.

6. **A informação visual apresentada fora do contexto da condução do veículo pode criar situação de risco da segurança rodoviária**

   Norma que restringe as informações fora do contexto da condução do veículo. Ou seja as informações não relacionadas com a condução devem minimizadas para evitar situações de risco.

   As informações apresentadas no *PDI* restringem-se a todo o contexto da condução do veículo diz respeito, pelo que a norma se **verifica** para estas condições.

   No uso sistema SNS durante a condução as informações apresentadas são tipicamente relacionadas com deslocamento do veículo, não apresentando outro tipo de informações não relacionadas com a condução. Assim **verifica-se** os requisitos exigidos.

7. **Evitar o acesso a tarefas durante a condução ao condutor**

   Este princípio de segurança que evita o condutor execute tarefas irrelevantes para a condução do veículo, evitando assim qualquer tipo de comportamento de risco.

   Para as duas interfaces PDI e SNS todas as tarefas são relevantes para a condução do veículo, por isso este princípio encontra-se **verificado** para ambas interfaces.

8. **Informações sobre o estado do veículo de impacto na segurança devem ser apresentados**

   Esta norma a assegura que o condutor tem acesso a todas as informações acerca do estado do veículo, de forma a condutor agir em segurança pelas informações apresentadas. Sempre que existir um problema com veículo deve ser apresentado as indicações na interface.

   A interface PDI possui um conjunto de componentes indicativos do estado do veículo. Esses indicadores possuem dois estados (luz acesa ou luz apagada), quando apagados indicam ao condutor que não existe nenhuma situação anómala com veículo caso contrário, se surgir um problema, uma ou mais luzes

acedem a indicar o possível o problema ao condutor. Resumindo a norma é **verificada** para este caso.

A aplicação interface do SNS não foi integrada no PDI. Esta interface não apresenta informações acerca do estado do veículo. Devido ao contexto da situação anterior a interface **não verifica** esta norma.

Capítulo 6

Conclusões

6.1 Conclusões

Esta dissertação foi apresentado uma interface para um painel de instrumentos para UM veículo e um sistema de navegação por satélite.

A definição da interface foi feito pelo levantamento dos princípios necessários para a conceção de uma interface para um PDI. Esse conjunto de princípios destacou-se a importância da atenção perdida quando o condutor tem de olhar para o painel de instrumentos. Destacando desses mesmos princípios os mais relevantes:

- Os componentes desenhados no painel de instrumentos devem ter uma familiaridade com os habituais congéneres de forma a permitir uma rápida adaptação aos componentes do painel por parte dos condutores.
- As informações apresentadas no painel devem ser coerentes e relevantes a cada situação de condução do veículo, evitando outras informações irrelevantes para a condução, que ponha em risco a segurança rodoviária.

Com a criação da interface para o PDI definiu-se também um ambiente para desenvolvimento de componentes gráficos e sua respetiva programação. Efetua-se a separação dos componentes gráficos dos sensores que fornecem as informações para os componentes do PDI. No ambiente desenvolvido especificou-se a parte gráfica do painel da parte da programação dos componentes desenhados no painel. Esta separação permite assim uma interdisciplinaridade entre os responsáveis da constituição gráfica do painel (*designer*) e os responsáveis pela programação (programadores) das

ações dos componentes do painel, bem como obtenção dos valores dos sensores do veículo. O aspeto gráfico é definido num documento *XML* que contém todas as informações sobre os componentes que vão ser apresentados no painel.

A interface do PDI foi desenvolvida na linguagem de programação *Java* para criação da interface gráfica e das ações dos componentes do painel de instrumentos. A aplicação *Java* que define o PDI e começa por carregar as informações dos componentes presentes no documento *XML* onde estão definidos as propriedades: imagens associadas ao componente bem como a sua posição e outras propriedades descritas nesta dissertação.

Para complemento da painel foi proposta uma aplicação *Java* de navegação por satélite, equipamento indispensável nos veículos rodoviárias da atualidade. O sistema de navegação proposto permite ao utilizador navegar da sua localização atual para o destino pretendido, modo de viagem mais rápida (duração mais curta) ou viagem mais curta (duração maior), com direções passo a passo, tempo estimado para chegada ao destino e velocidade instantânea. O mapa usado para o SNS foi Sapo Mapas pela sua excelente cobertura a nível nacional, bem como a excelente *API* que permitiu o seu funcionamento no sistema de navegação.

Após a proposta destas duas interfaces procedeu-se à sua avaliação. A avaliação incidiu nos princípios de desenho, na apresentação de informação e no desempenho da interface. Da avaliação conclui-se que a interface para o PDI cumpriu todos os princípios impostos para o seu funcionamento, à exceção do princípio "Interface deve ser desenhada de forma minimizar o número de vezes que utilizador tem de olhar para o ecrã da interface"porque não foi possível testá-lo em ambiente real.

Quanto à interface da aplicação do SNS dos oito princípios testados houve dois que não foram verificados:

1. "Interface deve minimizar a quantidade relances"não foi verificada tal como no PDI, pois não foram efetuadas os testes necessários para a sua verificação.
2. "Informações sobre o estado do veículo de impacto na segurança devem ser apresentados"também não foi verificada pois o sistema de navegação não se insere do contexto da condução do veículo, ou seja o sistema não possui nenhuma informação acerca do veículo.

Além do testes de princípios foram efetuados testes de desempenho da aplicação PDI. Os testes foram realizados em *PC* na plataforma *Java*. Para efeito de testes foram medidos o tempo médio e memória *RAM* do desenho dos elementos no ecrã. Os elementos dos cinco tipos de componentes foram desenhados no ecrã. A partir dos resultados obtidos conclui-se que o aumento do tempo médio e memória *RAM* é aproximadamente linear perante o aumento de elementos a serem desenhados no ecrã do painel. Estes resultados indicam que um painel de instrumentos não deve

conter demasiados componentes porque põe em causa a atenção do condutor e o tempo de resposta e precisão dos valores apresentado no PDI.

6.2 Principais contributos

Principais contributos dadas por esta dissertação:

1. Levantamento e apresentação de um conjunto de princípios para desenvolvimento de painéis de instrumentos.
2. Criação de um ambiente de desenvolvimento de painéis de instrumentos para programadores e *designers*.
3. Aplicação de navegação por satélite através da *API JavaScript* do sapo mapas.
4. Tutorial de como usar um *browser* em aplicações Java.

6.3 Trabalho futuro

O trabalho futuro que pode ser realizado para aperfeiçoamento do PDI e SNS:

1. Testar e avaliar as duas interfaces ao nível de relances que podem provocar no condutor, através um ferramenta quer permita localizar o sítio para onde utilizador está a olhar, e quantificar os tempos associados.
2. Integrar as interfaces PDI e SNS.
3. Testar a interface PDI Gecko Merula num cenário de condução real no veículo Gecko Merula.
4. Portar ambas interfaces para o sistema operativo *Android*[1] com objetivo de ter um *tablet* no interior do veículo a fazer de PDI e de sistema de navegação por satélite.

Com estas ideias funcionalidades descritos acima pretende-se que desperte o interesse a outros eventuais colaboradores para desenvolvimento e aperfeiçoamento das interfaces propostas.

[1] `http://www.android.com/`

Bibliografia

[Alves, 2009] Alves, R. (2009). Declarative approach to data extraction of web pages. Master's thesis, Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia.

[Baroni, 2009] Baroni, L. (2009). Algoritmos de navegação em tempo real para um sistema gps de posicionamento relativo de precisão.

[Baus et al., 2002] Baus, J., Krüger, A., and Wahlster, W. (2002). A resource-adaptive mobile navigation system. In *Proceedings of the 7th international conference on Intelligent user interfaces*, IUI '02, pages 15–22, New York, NY, USA. ACM.

[BMW, 2010] BMW (2010). *BMW Head-Up Display*. `http://www.bmw.com/com/en/insights/technology/connecteddrive/2010/safety/vision_assistance/head_up_display_information.html#more`. [Online; acedido 16-Novembro-2011].

[Bray and Yergeau, 2008] Bray, T. and Yergeau, F. (2008). Extensible markup language. `http://www.w3.org/TR/REC-xml/#sec-origin-goals`. [Online; accessed 21-Dezembro-2011].

[Commission, 2006] Commission, E. (2006). On safe and efficient in-vehicle information and communication systems: update of the european statement of principles on human machine interface. *Official Journal of the European Union*, pages 200–241.

[CVEL, 2010] CVEL (2010). Head-up displays. `http://www.cvel.clemson.edu/auto/systems/head-up_displays.html`. [Online; accessed 6-Dezembro-2011].

[CVEL, 2011] CVEL (2011). Instrument clusters. `http://www.cvel.clemson.edu/auto/systems/instrument_cluster.html`. [Online; accessed 6-Dezembro-2011].

[Edmonds, 2009] Edmonds, M. (2009). *How Dashboard Displays Work*. http://electronics.howstuffworks.com/gadgets/automotive/dashboard-display.htm. [Online; acedido 6-Outubro-2011].

[Eisenberg, 2008] Eisenberg, T. R. (2008). *Reventón Dashboard*. http://www.swiss-miss.com/2008/03/reventn-dashboa.html. [Online; accessed 17-November-2011].

[Feigenbaum, 2006] Feigenbaum, B. (2006). Swt, swing or awt: Which is right for you? http://www.ibm.com/developerworks/grid/library/os-swingswt/. [Online; accessed 21-Dezembro-2011].

[Fernandes, 2008] Fernandes, G. (2008). Culture's for us: uma plataforma para o turismo móvel. Master's thesis, Universidade da Madeira.

[Ferreira Filho, 2000] Ferreira Filho, J. C. (2000). Implementação de objetos replicados usando java. Master's thesis, Universidade Federal do Rio Grande do Sul. Instituto de Informática.

[Foundation, 2011] Foundation, T. E. (2011). Swing/swt integration. http://www.eclipse.org/articles/article.php?file=Article-Swing-SWT-Integration/index.html. [Online; accessed 21-Dezembro-2011].

[Gpsdrive, 2011] Gpsdrive (2011). Gpsdrive. http://www.gpsdrive.de/index.shtml. [Online; accessed 18-Dezembro-2011].

[Gryc, 2009a] Gryc, A. (2009a). Design challenges for digital instrument clusters. http://autoelectronics.com/body_electronics/instrumentation/digital-instrument-clusters-0727/index.html. [Online; accessed 8-Dezembro-2011].

[Gryc, 2009b] Gryc, A. (2009b). Digital instrument clusters: Technical challenges, market opportunities. Technical report, QNX Software Systems.

[Harris, 2007] Harris, W. (2007). *How Speedometers Work*. http://auto.howstuffworks.com/car-driving-safety/safety-regulatory-devices/speedometer.htm. [Online; acedido 9-Outubro-2011].

[Kanemitsu et al., 1991] Kanemitsu, H., Saito, T., Shima, J., and Tanaka, Y. (1991). Automobile navigation system using individual communication beacon. In *Vehicle Navigation and Information Systems Conference, 1991*, volume 2, pages 241 – 245.

[Mark, 2009] Mark (2009). *Ford Brings You the Future of Dashboards Gauges in Cars*. http://www.zurb.com/article/180/ford-brings-you-the-future-of-dashboards-. [Online; accessed 17-November-2011].

[Marques et al., 2011] Marques, L., Vasconcelos, V., Pedreiras, P., and Almeida, L. (2011). A flexible dashboard panel for a small electric vehicle. In *Information Systems and Technologies (CISTI), 2011 6th Iberian Conference on*, pages 1 –4.

[Mascarenhas, 2010] Mascarenhas, R. (2010). Uso do gps no mundo. `http://www.slideshare.net/renatomasca7/uso-do-gps-no-mundo-atual`. [Online; accessed 28-Dezembro-2011].

[Merriam-Webster, 2011] Merriam-Webster (2011). *Dashboard*. `http://www.merriam-webster.com/dictionary/dashboard`. [Online; acedido 6-Outubro-2011].

[Moere, 2009] Moere, A. V. (2009). *Ford's New Prototype Dashboard Concept*. `http://infosthetics.com/archives/2009/01/fords_new_prototype_dashboard_concepts.html`. [Online; accessed 17-November-2011].

[Moura, 2010] Moura, M. (2010). Plataforma/middleware para estudo de comunicações sem fios. Master's thesis, Universidade de Trás-os-Montes e Alto Douro.

[Navit, 2011a] Navit (2011a). About navit. `http://www.navit-project.org/`. [Online; accessed 18-Dezembro-2011].

[Navit, 2011b] Navit (2011b). Navit. `http://wiki.navit-project.org/index.php/Main_Page`. [Online; accessed 18-Dezembro-2011].

[Newcomb, 2011] Newcomb, D. (2011). How to decide if a factory nav system is right for you. `http://www.edmunds.com/car-technology/car-tech-101-in-dash-navigation-basics.html`. [Online; accessed 6-Dezembro-2011].

[Newmarch, 1999] Newmarch, J. (1999). Testing java swing-based applications. In *Technology of Object-Oriented Languages and Systems, 1999. TOOLS 31. Proceedings*, pages 156 –165.

[Ofria, 2007] Ofria, C. (1983-2007). *Understanding your dashboard gauges*. `http://www.familycar.com/classroom/dashboard.htm`. [Online; acedido 10-Outubro-2011].

[Oracle, 2011a] Oracle (1995-2011a). Class canvas. `http://docs.oracle.com/javase/1.4.2/docs/api/java/awt/Canvas.html`. [Online; accessed 21-Dezembro-2011].

[Oracle, 2011b] Oracle (1995-2011b). A closer look at the paint mechanism. `http://docs.oracle.com/javase/tutorial/uiswing/painting/closer.html`. [Online; accessed 21-Dezembro-2011].

[Oracle, 2011c] Oracle (1995-2011c). Creating the demo application. `http://docs.oracle.com/javase/tutorial/uiswing/painting/step2.html`. [Online; accessed 21-Dezembro-2011].

[Oracle, 2011d] Oracle (1995-2011d). Double buffering and page flipping. `http://docs.oracle.com/javase/tutorial/extra/fullscreen/doublebuf.html`. [Online; accessed 27-Dezembro-2011].

[Oracle, 2011e] Oracle (1995-2011e). How to make frames (main windows). `http://docs.oracle.com/javase/tutorial/uiswing/components/frame.html`. [Online; accessed 21-Dezembro-2011].

[Oracle, 2011f] Oracle (1995-2011f). How to use jbutton features. `http://docs.oracle.com/javase/tutorial/uiswing/components/button.html#jbutton`. [Online; accessed 21-Dezembro-2011].

[Oracle, 2011g] Oracle (1995-2011g). How to use labels. `http://docs.oracle.com/javase/tutorial/uiswing/components/label.html`. [Online; accessed 21-Dezembro-2011].

[Oracle, 2011h] Oracle (1995-2011h). How to use panels. `http://docs.oracle.com/javase/tutorial/uiswing/components/panel.html`. [Online; accessed 21-Dezembro-2011].

[Oracle, 2011i] Oracle (1995-2011i). How to write an action listener. `http://docs.oracle.com/javase/tutorial/uiswing/events/actionlistener.html`. [Online; accessed 21-Dezembro-2011].

[Oracle, 2011j] Oracle (1995-2011j). Lesson: A brief introduction to the swing package. `http://docs.oracle.com/javase/tutorial/ui/overview/index.html`. [Online; accessed 21-Dezembro-2011].

[Oracle, 2011k] Oracle (1995-2011k). Painting in awt and swing. `http://www.oracle.com/technetwork/java/painting-140037.html#awt`. [Online; accessed 21-Dezembro-2011].

[Patrícia Goncalves, 2008] Patrícia Goncalves, J. T. (2008). A gui for a software that analyses a composite bolted joint. *International Journal on Human-Computer Interaction*, I(4):25–41.

[Rúben Fonseca, 2007] Rúben Fonseca, A. S. (2007). Alternativas ao xml: Yaml e json. In *Xata 2007 - Aplicações e Tecnologias Associadas*, pages 33–46.

[Silva, 2010] Silva, R. (2010). Sistema de recolha e execução de rotas (gps) e partilha em rede social recgps. Master's thesis, ISEL - Instituto Superior de Engenharia de Lisboa.

[Sol, 2011] Sol, S. (2011). Xml advantages and disadvantages. `http://www.theukwebdesigncompany.com/articles/xml-advantages-disadvantages.php`. [Online; accessed 20-Dezembro-2011].

[Steve Northover, 2004] Steve Northover, M. W. (2004). *SWT: The Standard Widget Toolkit, Volume 1*. Addison Wesley.

[Tapken, 2011] Tapken, R. (2011). Java: Howto embed swt widget into swing jframe. http://www.blogs.uni-osnabrueck.de/rotapken/2011/01/05/java-howto-embed-swt-widget-into-swing-jframe/. [Online; accessed 21-Dezembro-2011].

[UMTRI, 2002] UMTRI (2002). Statement of principles, criteria and verification procedures on driver interactions with advanced invehicle information and communication systems. Technical report.

[Wikipedia, 2011a] Wikipedia (2011a). Abstract window toolkit — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Abstract_Window_Toolkit&oldid=459135323. [Online; accessed 24-December-2011].

[Wikipedia, 2011b] Wikipedia (2011b). Electronic instrument cluster — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Electronic_instrument_cluster&oldid=458391281. [Online; accessed 18-November-2011].

[Wikipedia, 2011c] Wikipedia (2011c). Euclidean vector — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Euclidean_vector&oldid=467462318. [Online; accessed 25-December-2011].

[Wikipedia, 2011d] Wikipedia (2011d). Parsing — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Parsing&oldid=466511251. [Online; accessed 24-December-2011].

[Wikipedia, 2011e] Wikipedia (2011e). Rotation matrix — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Rotation_matrix&oldid=465484454. [Online; accessed 26-December-2011].

[Wikipedia, 2011f] Wikipedia (2011f). *Head-up display — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Head-up_display&oldid=457534079. [Online; accessed 8-November-2011].

[Wikipedia, 2011g] Wikipedia (2011g). Transmission control protocol — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Transmission_Control_Protocol&oldid=463245067. [Online; accessed 28-December-2011].

[Wikipédia, 2011] Wikipédia (2011). *Baby boomer* — wikipédia, a enciclopédia livre. http://pt.wikipedia.org/w/index.php?title=Baby_boomer&oldid=26251197. [Online; accessed 5-novembro-2011].

[Wood, 2008] Wood, L. (2008). *Inside the Cockpit of Tomorrow's Car*. http://www.livescience.com/5005-cockpit-tomorrow-car.html. [Online; accessed 8-November-2011].

Anexos

Anexo A

Documento XML da interface do painel

```
<?xml version="1.0"?>

<skin>
        <nome>home</nome>
                <screen>
                        <with>1280</with>
                        <heigth>798</heigth>
                </screen>

                <background>
                        <file>back.png</file>
                        <refreshable>false</refreshable>
                </background>

                <speedometer>
                        <file>speed2.png</file>
                        <imgx>163</imgx>
                        <imgy>168</imgy>

                        <pointer>
                                <color>red</color>
                                <p0x>317</p0x>
                                <p0y>321</p0y>
```

```
                <p1x>184</p1x>
                <p1y>372</p1y>
                <vm>0</vm>
                <vmax>160</vmax>
                <angle>231.3</angle>
        </pointer>

        </speedometer>

<speedometer>
        <file>rpm.png</file>
        <imgx>827</imgx>
        <imgy>163</imgy>

        <pointer>
                <color>red</color>
                <p0x>974</p0x>
                <p0y>313</p0y>
                <p1x>839</p1x>
                <p1y>360</p1y>
                <vm>0</vm>
                <vmax>12</vmax>
                <angle>231.3</angle>
        </pointer>

</speedometer>

<gauge>
        <file>bateria2.png</file>
        <imgx>612</imgx>
        <imgy>346</imgy>


        <gaugeElement>
                <elements>4</elements>
                <file>barra_bateria.png</file>
                <px>615</px>
                <py>450</py>
                <length>78</length>
                <heigth>28</heigth>
        </gaugeElement>
</gauge>
```

```
        <indicator>
                <imageActive>oleo.png</imageActive>
                <imageInactive>oleo2.png</imageInactive>
                <px>447</px>
                <py>567</py>

        </indicator>

        <indicator>
                <imageActive>temperatura.png</imageActive>
                <imageInactive>temperatura2.png</imageInactive>
                <px>610</px>
                <py>569</py>

        </indicator>

        <indicator>
                <imageActive>travao.png</imageActive>
                <imageInactive>travao2.png</imageInactive>
                <px>800</px>
                <py>569</py>

        </indicator>

        <staticImage>
                <file>barra_fundo.png</file>
                <px>266</px>
                <py>548</py>
        </staticImage>
</skin>
```