Universidade de Évora

Escola de Ciências e Tecnologia

Mestrado em Engenharia Informática

Interface de instrumentos para veículo eléctrico Gecko Merula - Redes Sociais

Paulo Alexandre Paixão Amaral

Orientador
Professor Luís Miguel de Mendonça Rato

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

Évora, Dezembro de 2011

Mestrado em Engenharia Informática

Interface de instrumentos para veículo eléctrico Gecko Merula - Redes Sociais

Paulo Alexandre Paixão Amaral

Orientador
Professor Luís Miguel de Mendonça Rato

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

Aos meus pais
Aos meus irmãos
À minha família
Aos meus amigos

Sumário

O principal objetivo deste trabalho é o desenvolvimento do painel de instrumentos do veículo elétrico *Gecko Merula*. Para tal, foi definido um conjunto de princípios de apresentação de informação em veículos e foi também definido, para a plataforma *Java* e para a plataforma *Android*, um ambiente de desenvolvimento que permite a definição de painéis de instrumentos de modo flexível. De forma a fazer uso das ligações sociais estabelecidas entre os utilizadores do *Facebook* foi definida uma aplicação denominada VeículoSocial que faz uso destas ligações de forma a filtrar conteúdos de localização num mapa. O ambiente de desenvolvimento definido foi testado numa plataforma teste, de forma a estudar o tempo de desenho dos componentes no ecrã. As interfaces do veículo *Gecko Merula* e do VeículoSocial foram avaliadas com base nos princípios de apresentação de informação em veículos. A partir dos testes de desempenho ao ambiente de desenvolvimento concluiu-se que, para a plataforma de teste, o mesmo não deverá definir um número de componentes demasiado elevado no painel de instrumentos de um veículo, de forma a não afetar a precisão do painel. Da avaliação das interfaces das aplicações concluiu-se que é necessária a realização de testes futuros nas interfaces e também que a aplicação VeículoSocial não garante uma resposta atempada a um pedido de *input* de utilizador.

Gecko Merula electric vehicle dashboard - Social Networks

Abstract

The main goal for this work is the development of a dashboard for the electric vehicle Gecko Merula. For this purpose, it was defined a set of vehicle information presentation principles and it was also defined, for the Java platform and for the Android platform, a development environment that produces in a flexible way a vehicle dashboard. To make use of the social connections between users of Facebook, it was defined an application named VeículoSocial that uses these social connections in order to filter localization content to a map. The development environment was tested, on a test platform, in order to study the drawing time of the components on the screen. Gecko Merula and VeículoSocial's interfaces were evaluated based on the vehicle information presentation principles. From the performance tests performed on the development environment it was concluded that, for the test platform, the environment should not define an excessively high number of components on a vehicle dashboard, so that it doesn't affect the dashboard's precision. From the evaluation of the applications' interface it was concluded that future interface tests are required and that the VeículoSocial application does not guarantee a timely response to a user's input request.

Agradecimentos

Gostaria de agradecer, em primeiro lugar, aos meus pais e irmãos, sem os quais nada disto seria possível.

Depois, gostaria de agradecer ao Professor Luis Rato, pelo acompanhamento e paciência que teve durante o desenvolvimento desta dissertação.

Quero também agradecer ao Professor José Saias pela disponibilidade e pelo apoio prestado em algumas questões relacionadas com esta dissertação, e ao meu colega de trabalho e amigo Miguel Gualdino.

Não posso, no entanto, deixar de agradecer a todos os meus amigos que me ajudaram durante esta fase. A eles um muito obrigado!

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Enquadramento Geral

Os veículos automóveis de hoje em dia são mais do que meros meios de locomoção. Muitos dos automóveis modernos são meios de consumo de entretenimento e meios de comunicação para os seus utilizadores [Kern and Schmidt, 2009]. A utilização das tecnologias digitais em automóveis, em particular, fornecem aos veículos que operamos nos dia de hoje um nível de interatividade que nos permite realizar uma diversidade de operações com os mesmos que há poucos anos não tínhamos acesso: podemos ouvir música de leitores de mp3 portáteis, podemos aceder a sistemas de navegação GPS, podemos ligar-nos a redes sociais, etc.

No entanto, a tarefa primária durante o manuseio de um automóvel é a tarefa da condução [AAM, 2006]. Os diversos componentes a que acedemos nos nossos automóveis não devem distrair a nossa atenção da estrada de forma a que coloquemos em risco a nossa vida e a vida das outras pessoas. Assim, os indivíduos responsáveis pela definição de painéis de instrumentos para veículos automóveis devem ter em conta que o desenho do painel de instrumentos do veículo deve obedecer a algumas regras básicas de forma a que se tente prevenir ao máximo o desvio de atenção dos condutores da estrada.

Imagine-se, a título de exemplo, uma situação na qual um veículo possui um leitor de mp3 na qual a escolha da música é feita, num ecrã *touch-screen*, através do *scroll* vertical de uma lista de músicas. A tarefa de escolha de uma música, no leitor de mp3 do painel de instrumentos deste veículo, desvia a atenção do condutor da estrada por mais tempo que o devido, podendo esta situação ser perigosa para a

segurança rodoviária. Este exemplo espelha a importância do desenho simples e adequado à prática da condução que deve ser seguido durante o desenvolvimento de um painel de instrumentos de um veículo.

O condutor de um veículo deve sentir que as informações que obtém do painel de instrumentos do veículo são corretas e temporalmente precisas. O painel de instrumentos para um veículo deve, portanto, transmitir confiança ao condutor acerca da validade e tempo de resposta da informação que lhe é indicada.

O *Gecko Merula* é um veículo elétrico produzido em Évora pelo Engenheiro Ricardo Melro com o auxílio de alunos da Universidade de Évora. Uma vez construído o protótipo do veículo, foi proposta ao Departamento de Informática da Universidade de Évora, por Ricardo Melro, a definição do painel de instrumentos do *Gecko Merula*. A definição do painel de instrumentos do veículo *Gecko Merula* será, assim, apresentada na presente dissertação.

O painel de instrumentos do veículo elétrico *Gecko Merula* deve motivar as pessoas à utilização do mesmo. Esta motivação pode ser alcançada através da inclusão de tecnologias populares no painel de instrumentos do *Gecko Merula*. As redes sociais são tecnologias que são utilizadas por pessoas de todo o mundo [Report, 2011]. Esta aderência à escala global às redes sociais pode ser aproveitada de modo a atrair o maior número possível de utilizadores para o *Gecko Merula*, através da inclusão destas tecnologias no painel de instrumentos deste veículo. Assim, será necessário compreender de que forma as redes sociais podem dar um contributo útil para um veículo durante o ato de condução do mesmo.

A definição do painel de instrumentos do *Gecko Merula* foi feita em conjunto pelo autor desta dissertação e por Miguel Gualdino. Para além do painel de instrumentos do veículo, que foi implementado em conjunto pelo autor da dissertação e por Miguel Gualdino, foram também desenvolvidas duas aplicações: um sistema de navegação GPS, da autoria de Miguel Gualdino, e uma aplicação que faz uso de informações retiradas de uma rede social, da autoria do autor da presente dissertação.

1.2 Objetivos do trabalho

O objetivo principal da dissertação será o de definir um painel de instrumentos para o veículo elétrico *Gecko Merula*. De forma a que o painel de instrumentos do *Gecko Merula* seja seguro e transmita confiança para o condutor, deve ser definido um conjunto de princípios de disposição de informação em veículos de modo a que a interface do veículo possa ser avaliada.

O painel de instrumentos de um veículo deverá ser facilmente adaptado de forma a que eventuais alterações no mesmo sejam práticas. Portanto, esta adaptação deve ser realizada sem alterar a disposição geral dos elementos presentes no painel.

O grupo de trabalho responsável pela definição do painel de instrumentos de um veículo pode ser um grupo de trabalho interdisciplinar, pelo que o desenvolvimento do painel de instrumentos deve permitir que os vários indivíduos, (de diferentes áreas de trabalho) envolvidos na definição do painel, possam trabalhar conjuntamente sem que esta situação afete o desenvolvimento geral desta interface. Assim, será objetivo do trabalho o desenvolvimento de um ambiente de desenvolvimento de painéis de instrumentos para veículos que permita a alteração de componentes do painel sem afetar os restantes elementos do mesmo e que permita a interdisciplinaridade do grupo de trabalho responsável pela construção do painel.

Será também objetivo desta dissertação estudar as vantagens da utilização de redes sociais em veículos automóveis. Uma vez compreendida a utilidade do uso de redes sociais em veículos, estas serão aplicadas no painel de instrumentos do *Gecko Merula*.

1.3 Estrutura da dissertação

A dissertação está divida em 7 capítulos.

O primeiro capítulo é a **Introdução**, e servirá para apresentar o enquadramento geral do trabalho, para enunciar os objetivos do mesmo e para apresentar alguns conceitos fundamentais à leitura da dissertação.

O segundo capítulo é o **Estado da arte e trabalho relacionado**. Este capítulo iniciar-se-á por uma descrição geral do desenvolvimento de interfaces pessoa-máquina. De seguida, serão referidos alguns aspetos a ter em conta no desenvolvimento de interfaces para veículos automóveis e serão apresentados alguns exemplos de trabalhos relacionados com o estudo de interfaces para automóveis. Pode ainda encontra-se neste capítulo um conjunto de exemplos de trabalhos que fazem uso de redes sociais em veículos automóveis.

O terceiro capítulo refere-se à **Apresentação de informação em Veículos**, e terá como objetivo a apresentação de uma lista de princípios que se devem seguir no desenvolvimento de interfaces para veículos automóveis.

O quarto capítulo denomina-se **Ambiente de desenvolvimento**. Neste capítulo será apresentado o ambiente de desenvolvimento de painéis de instrumentos definido nesta dissertação, sendo que o capítulo se inicia pela listagem de componentes habituais em veículos automóveis. De seguida, serão apresentas as linhas orientadores que foram seguidas na implementação do ambiente de desenvolvimento, de forma a que seja depois apresentada a arquitetura definida para o ambiente. A secção seguinte efetuará a descrição do documento XML de definição de componentes do painel de instrumentos, que será seguida pela descrição de implementação do ambiente. A descrição da implementação do ambiente será composta pela descrição geral da tecnologia *Java*, pela descrição do *parsing* do documento XML de

definição dos componentes, pela descrição da divisão dos componentes efetuada no ambiente de desenvolvimento, pela visão geral do desenho dos componentes no ecrã, pela descrição do desenho de cada um dos componentes do painel de instrumentos no ecrã e pela descrição da implementação do ambiente em *Android*. A descrição da implementação do ambiente de desenvolvimento em *Android* inicia-se por uma abordagem geral ao desenvolvimento de aplicação em *Android*, seguindo depois para a apresentação das interfaces *Java* de auxílio à portabilidade do ambiente de desenvolvimento, finalizando-se a secção de descrição da implementação do ambiente de desenvolvimento em *Android* com a listagem das diferenças de implementação do ambiente para a linguagem *Java* e para a plataforma *Android*. O quarto capítulo será então concluído com um exemplo da utilização do ambiente de desenvolvimento.

O quinto capítulo descreve a **Aplicação para Redes Sociais**. Este capítulo inicia-se com a apresentação da aplicação VeículoSocial, enquadrando a mesma e referindo os serviços disponibilizados. Esta secção continuará para a apresentação da arquitetura da aplicação VeículoSocial e pela listagem das tecnologias utilizadas no desenvolvimento da aplicação (listagem que é composta pelas descrições da comunicação por *sockets* TCP, *JSON*, *Facebook* API, *Google Maps* em *Android* e *PostgreSQL*). A secção de apresentação da aplicação VeículoSocial finaliza-se com a descrição da implementação da mesma, na qual será apresentada a interface da aplicação cliente, a comunicação entre a aplicação cliente e a aplicação servidor, o método de autenticação de um utilizador no *Facebook*, a descrição do mapa da aplicação e da ação do botão "Marcar Posição" e pela especificação dos serviços "Pontos de Interesse" e "Procura de Amigos".

O sexto capítulo diz respeito à **Interface do veículo elétrico *Gecko Merula***. O capítulo é iniciado pela enunciação dos componentes presentes no painel de instrumentos do *Gecko Merula*, seguindo-se a apresentação da interface do painel de instrumentos, na qual é também realizado um teste de desempenho ao ambiente de desenvolvimento. A secção seguinte descreve a interface da aplicação VeículoSocial, finalizando-se o capítulo com a comparação das interfaces do painel de instrumentos e do VeículoSocial com os princípios apresentados no terceiro capítulo.

O sétimo e último capítulo é o respetivo às **Conclusões**. Neste capítulo serão apresentadas as conclusões da dissertação, as principais contribuições e o trabalho futuro da mesma.

1.4 Conceitos

Nesta secção serão referidos alguns conceitos fundamentais relacionados com a temática de *design* de interfaces para veículos e, estando estes conceitos bem definidos na literatura [AAM, 2006], será importante referir os mesmos para uma melhor compreensão da dissertação.

A noção de ***design*** não possui uma definição consensual [Ralph and Wand, 2009]. Neste trabalho define-se ***design*** como um processo de planeamento e construção apoiado num objetivo previamente traçado. O resultado final do processo de ***design*** pode atingir várias formas, tal como o de dar uma forma física à interface de um veículo elétrico.

A interface desenhada irá então ser responsável pela **informação visual** dada pelo sistema, definindo-se **informação visual** como tudo aquilo que é apresentado pelo sistema através da modalidade visual (imagens, textos, etc.). A modalidade visual será representada num ***display***, que é um aparelho que fornece informação visual ao condutor.

A utilização das diversas funcionalidades disponibilizadas pelo *display* pode levar a uma insuficiente atenção do condutor à tarefa da condução, tarefa essa que deve ser sempre a principal no manuseio de um veículo [Kern and Schmidt, 2009]. Assim, é fulcral que o *design* de uma interface para veículos tenha em conta a **distribuição de informação ao condutor**. Esta distribuição deverá ser realizada de modo a alocar apropriadamente a quantidade de atenção necessária ao condutor na realização das mais diversas tarefas passíveis de realizar através da interface, dado o limitado número de recursos físicos e mentais que se podem exigir ao utilizador enquanto conduz o veículo.

A atenção necessária despendida pelo condutor ao sistema é tanto maior quanto mais complexa for a **interação** com o sistema. Refere-se interação como todo o *input* que é dado ao sistema por parte do condutor, seja em resposta a uma informação dada pelo sistema ou pelo desejo do utilizador em realizar uma ação sobre o mesmo. Ao interagir com o *display*, o condutor dirige parte da sua atenção para a interface do sistema, através de uma breve olhadela, ou relance, ou ***glance***. Uma ***glance***, tal como definido na ISO 15007, SAE J2396, é "o tempo desde o momento em que o olhar do condutor se movimenta para um alvo e o momento em que se afasta desse mesmo alvo. Isto inclui o tempo em que efetivamente se encara o alvo, bem como o tempo de transição de início ou fim de olhar para o alvo (mas não ambos)".

A eficiente construção de uma interface de um veículo deverá então reduzir ao máximo a **distração** da tarefa de condução. **Distração** define-se como o desvio de atenção por parte do condutor para obter informação relevante para si. Essa informação pode ou não estar relacionada com o ato de conduzir. Quando esta informação provém do *display*, a possibilidade da mesma causar distração ao condutor deve-se sobretudo a um deficiente *design* da interface na medida em que os dados expressos pela mesma não estão dispostos de forma a que sejam apreendidos pelo condutor através de breves *glances*.

Capítulo 2

Estado da arte e Trabalho Relacionado

2.1 Aspetos do desenvolvimento de uma interface pessoa-máquina

As interfaces pessoa-máquina são a matéria de estudo predominante no campo das interações homem-máquina. O estudo da interação entre um indivíduo e uma máquina engloba um rigoroso estudo e planeamento dos objetivos a que se pretende responder através desta interação. Este estudo requer um detalhado levantamento das necessidades que o utilizador pretende satisfazer através da sua interação com a máquina. O bom desenho de uma interface pessoa-máquina permite ao utilizador uma fácil e rápida apreensão da informação transmitida por uma máquina (i.e., permite ao utilizador a correta apreensão do *output* produzido por uma máquina). Por outro lado, o bom desenho de uma interface pessoa-máquina permite também que o utilizador possa fornecer facilmente à máquina os dados relevantes ao seu funcionamento (i.e., permite ao utilizador a correta introdução de *input* na máquina) [Galitz, 2002].

O desenvolvimento de uma interface pessoa-máquina deve ser realizado de uma forma orientada ao utilizador [Norman, 2002]. O utilizador de um sistema informático, para qualquer modalidade de *input* utilizada (teclado, rato, reconhecimento de voz, ...), deve ser sempre o agente a controlar a interação pessoa-máquina. O utilizador deve, portanto, aperceber-se claramente **quais** as possibilidades de interação que o sistema lhe permite e devem ser fornecidos mecanismos ao utilizador de modo a que

este se aperceba claramente **como** pode comunicar com o sistema. Assim, de forma a que o utilizador assuma o controlo no processo de interação pessoa-máquina, uma interface de um sistema informático deve [Norman, 2002]:

1. Permitir que o utilizador reconheça quais as ações disponíveis no sistema, em determinado momento
2. Tornar claras ao utilizador as consequências de determinadas ações
3. Fornecer ao utilizador uma informação válida e clara do estado atual do sistema

A interface de um sistema deve ser realizada de modo a ter em conta as limitações do sistema. Um sistema informático possui um conjunto de limitações físicas, como a quantidade de memória e a capacidade de processamento que possui. Estas limitações físicas não devem, no entanto, ser impeditivas do estabelecimento de uma eficaz comunicação máquina-utilizador e, por isso, durante o desenvolvimento de uma interface pessoa-máquina devem ser reconhecidas a limitações da máquina (ou máquinas) para a qual se desenvolve a interface. Se esta mesma interface for implementada para um conjunto de máquinas com especificações diferentes entre si, o seu desenvolvimento deverá adotar um conjunto de estratégias de modo a que os utilizadores das diferentes plataformas partilhem uma experiência semelhante durante a utilização do sistema. Um exemplo desta última situação é o da plataforma *Google Android*, para a qual o desenvolvimento de aplicações deverá ter em conta a diversidade de dispositivos que poderão correr a aplicação (*tablets*, vários *smartphones* com diferentes especificações, ...) [BrightHub, 2011]. Ao desenvolver uma aplicação para o sistema *Android* são utilizados vários componentes que se adaptam às características dos dispositivos onde são visualizados (características como a densidade, resolução e tamanho de ecrã), permitindo assim a heterogeneidade de dispositivos que executam aplicações desenvolvidas para *Android*.

Por outro lado, a interface de um sistema deve também ser realizada de modo a ter em conta as limitações físicas e espaciais do utilizador. No processo de *design* da interface, o construtor da mesma deve interrogar-se: "quais são as pessoas que vão utilizar este sistema?". Todas as pessoas possuem capacidades e limitações diferentes, e a resposta a esta pergunta determinará o público-alvo para o qual o sistema deverá proporcionar uma interface que permitirá a estas pessoas a realização das tarefas permitidas pelo sistema. O desenho de uma interface deve ser feita de modo a incluir o maior tipo de indivíduos possível de modo a reduzir a limitação de utilização de um sistema por indivíduos de capacidades físicas limitadas, sendo que o avanço das tecnologias de *input* e *output* presente nos dias de hoje permitem que, por exemplo, seja utilizada a modalidade de reconhecimento de voz para indivíduos que possuam uma paralisia física. O contexto espacial no qual o sistema irá ser utilizado deverá também ser estudado no desenvolvimento de uma interface. Nem todas as tarefas que um utilizador pretende realizar num sistema informático podem

ser realizadas de uma forma segura, para o utilizador, se esse mesmo sistema for utilizado num ambiente no qual o utilizador não poderá dedicar toda a sua atenção às necessidades de *input* e *output* exigidas pela interface (por exemplo, o sistema utilizado durante o manuseio de um veículo). Esta última questão possui particular importância para a presente dissertação e será, por isso, abordada de forma recorrente ao longo da mesma.

2.2 *Design* de interfaces para veículos automóveis

A quantidade e diversidade de funcionalidades existentes nos veículos automóveis aumentou significativamente nos últimos anos. Atualmente, o condutor de um veículo pode efetuar várias tarefas enquanto conduz, como procurar direções de navegação num sistema de navegação GPS, efetuar chamadas telefónicas através do veículo, enviar mensagens de texto, etc [Schmidt et al., 2010]. As condições exteriores ao automóvel têm também sofrido um conjunto de alterações que afetam a prática da condução: o aumento do tamanho das estradas, o maior número de veículos e peões e a grande diversidade de paisagens que um condutor encontra (que alternam entre paisagens vazias e monótonas e entre paisagens dinâmicas e altamente populadas) são exemplos de aspetos que alteraram a forma como os condutores interagem com os seus veículos [de Oliveira Pereira, 2009].

O nível de atenção que o condutor emprega nas diversas tarefas que executa durante a condução do seu veículo devem ser tomadas em conta aquando do desenho de interfaces que fornecem interações entre veículos e os seus respetivos condutores. A segurança do condutor deve ser sempre o aspeto mais importante a ter em conta no desenho da interface de um veículo. O elevado número de acidentes de viação que ocorrem anualmente são uma das maiores causas de morte nos países industrializados, sendo que o fator responsável pela maioria destes acidentes é o erro humano [de Oliveira Pereira, 2009]. A diminuição do erro humano na tarefa da condução deve ser o objetivo primário do desenho de um sistema de interação entre um indivíduo e um veículo, pelo que este sistema deve ser desenhado de forma a evitar ao máximo o desvio de atenção por parte do condutor [Kern and Schmidt, 2009].

A forma de representação das variadas informações a que se pode aceder presentemente no interior de um veículo é uma questão que carece ainda de um aprofundado estudo devido ao elevado crescimento tecnológico a que se tem assistido nos últimos anos [Schmidt et al., 2010]. Existem já alguns trabalhos destinados ao estudo da implementação de sistemas para veículos que tentam prevenir o desvio de atenção dos condutores, como é o caso do trabalho de *Salvucci* no qual o autor apresenta um sistema, denominado *Distract-R* [Salvucci, 2005]. Este tem por objetivo prever e compreender a distração dos condutores e construir ferramentas pouco distrativas para a tarefa da condução. A ferramenta descrita neste trabalho possui um conjunto de componentes divididos em diferentes modalidades (botões, microfones, *display*,

etc.) e permite a fácil especificação de um novo componente a utilizar na interface de um veículo. Esta ferramenta permite ainda estudar o comportamento do condutor, face à interface apresentada, através da simulação de um cenário de condução, permitindo posteriormente a visualização e análise do comportamento do condutor nesta simulação.

A forma de interação entre um condutor e um veículo sofreu profundas alterações nos últimos anos. Ao contrário do que se verificava há alguns anos atrás, muitos dos modernos sistemas de interação humano-veículo evoluíram de painéis com diversos componentes físicos, como botões e velocímetros analógicos, para *displays* digitais que possuem geralmente um ecrã com *touch-screen*. Para além das diferentes formas de visualização dos dados e as diferentes formas de interação que um *display touch-screen* permite, verificou-se também que a satisfação dos condutores na utilização desta modalidade de *input* e *output* é maior do que a satisfação que possuíam ao utilizar sistemas mais antigos [Sudarshan et al., 2010].

A interação entre um indivíduo e um veículo pode não ser necessariamente realizada através da modalidade visual. A utilização de sistemas multimodais para interagir com um veículo é frequentemente abordada em trabalhos que estudam o modelo de interação humano-veículo. No entanto, similarmente ao que acontece na definição de interfaces visuais, é necessário compreender quais os aspetos que devem ser considerados durante a implementação de sistemas que utilizem diferentes modalidades de *input* e *output*. *Chang et al.* [Chang et al., 2009] realizaram um conjunto de testes a um sistema de reconhecimento de voz de um veículo da *Volkswagen Group* e, como resultado desse estudo, apresentaram um conjunto de princípios a ser seguidos na implementação de um sistema de reconhecimento de voz para um veículo. Semelhantemente, *Hua* e *Ng* descreveram uma *framework* para a utilização da modalidade de reconhecimento de voz em veículos automóveis [Hua and Ng, 2010]. Neste trabalho, os autores apresentaram um conjunto de princípios destinados à resolução dos principais problemas que se encontram na utilização de sistemas de reconhecimento de voz em veículos, de modo a facilitar a sua adoção em trabalhos futuros.

A utilização da modalidade de reconhecimento gestual em interfaces humano-veículo foi já também estudada por alguns autores, como é o caso do trabalho de *Alpern* e *Minardo*, no qual os autores estudaram a utilização de gestos no controlo de tarefas secundárias ao manuseio do veículo [Alpern and Minardo, 2003] (i.e., tarefas que não estão diretamente relacionadas com o ato da condução, tal como a consulta do sistema de navegação GPS). Neste estudo, os autores desenvolveram uma interface, visível no vidro da frente do veículo, que podia ser acedida através de simples gestos do condutor. De seguida, os autores realizaram um conjunto de experiências nas quais um conjunto de indivíduos utilizavam os movimentos de apenas uma das suas mãos para interagir com o veículo. Através destas experiências, os autores concluíram que o reconhecimento gestual é uma modalidade viável para utilização em interfaces humanos-veículos e que as operações requeridas para completar uma ou

mais tarefas não devem exceder breves *glances* da atenção do utilizador.

A implementação de uma interface para um veículo, independentemente da modalidade ou modalidades de interação utilizadas, deve ser realizada com precaução devido aos problemas de segurança que pode causar ao utilizador. Com o intuito de tentar padronizar a tarefa de *design* de interfaces para veículos, foram já realizados trabalhos que, ao apresentar conjuntos de princípios a seguir pelos responsáveis da definição de sistemas de interação para veículos, fornecem medidas de minimização da distração do condutor quando interage com o seu veículo. A *Alliance of Automobile Manufacturers* é uma entidade composta por representantes de vários fabricante de carros (*BMW Group*, *Ford Motor Company*, *General Motors Company*, etc.) que se encontram regularmente com o objetivo de definir um conjunto de orientações gerais que devem ser utilizadas na definição de painéis de instrumentos para veículos automóveis [AAM, 2006]. Neste conjunto de princípios, a *Alliance of Automobile Manufacturers* fornece orientações importantes acerca do *design* e instalação de sistemas de interação em veículos que se focam na tarefa da condução. Alguns destes princípios serão abordados na secção 2.

O tipo de informações que geralmente se encontra nos painéis de instrumentos de veículos são informações acerca do estado atual da máquina. No entanto, existem estudos nos quais os autores estudaram formas de incluir, nos painéis dos veículos, ferramentas que auxiliam o utilizador a praticar uma condução mais segura ou mais económica, como é o caso de [Kumar and Kim, 2005] e [Meschtscherjakov et al., 2009]. No primeiro caso, os autores debruçaram-se sobre a definição de um velocímetro que auxilia no controlo da velocidade praticada pelo condutor. Neste trabalho, os autores apresentaram um velocímetro que não só apresenta a velocidade atual do veículo como apresenta também o limite de velocidade definido para a localização atual do veículo. Existe também uma variação na cor de algumas regiões do velocímetro de acordo com a proximidade da velocidade atual do veículo ao limite legal de velocidade imposto, sendo que à medida que a velocidade da máquina se aproxima do limite legal, esta cor torna-se mais avermelhada. Os resultados apresentados neste estudo concluíram que a utilização deste velocímetro pode desencorajar os condutores a ultrapassar os limites legais de velocidade e, por consequência, reduzir o número de acidentes de viação. No segundo caso, os autores descreveram cinco ferramentas que auxiliam o condutor a poupar recursos do veículo: a ferramenta **EcoMatic** que pode ser ativada ou desativada de acordo com a vontade do utilizador e que ajusta automaticamente vários parâmetros do veículo de modo a reduzir os níveis de consumo de combustível; a ferramenta **EcoPedal** que se assemelha a um tradicional acelerador de um veículo e que, ao detetar uma aceleração exagerada por parte do condutor, aplica uma força contrária ao movimento de aceleração no pedal, diminuindo assim os níveis de consumo de combustível; a ferramenta **EcoSpeedometer** na qual o velocímetro muda de cor de acordo com o nível de consumo de combustível; a ferramenta **EcoDisplay** que permite a visualização de um conjunto de ícones semelhantes a "folhas verdes" (como as folhas de uma árvore) e que

apresenta mais ou menos folhas de acordo com o nível de consumo de combustível; a ferramenta **EcoAdvisor** que fornece indicações auditivas de forma a informar o condutor acerca do estado atual do veículo e que fornece também indicações de poupança de combustível. As diversas funcionalidades apresentadas foram testadas por vários indivíduos de forma a avaliar as diferentes ferramentas e os autores concluíram então que houve uma aceitação global das mesmas e que a finalidade para as quais estas se destinam (auxiliar na poupança de recursos do veículo) ajudaram à forte aceitação por parte dos participantes.

2.3 Redes sociais em veículos automóveis

A integração de redes sociais em veículos automóveis é uma ideia que não surgiu com a expansão mundial das redes sociais nos finais da década de 2000. Como exemplo desta situação, existe uma dissertação de mestrado datada de 2006 na qual o autor propôs que se utilizassem as redes sociais mais populares da altura (*Orkut*, *Friendster*, etc.) de forma a desenvolver um sistema para um veículo em que se combinassem informações de localização e informações sociais [Liang, 2006]. As informações de localização dos veículos eram dadas pelos sistemas GPS dos mesmos, enquanto que as informações sociais eram dadas pela ligação dos veículos a redes sociais. Segundo o autor da dissertação, o acesso ao grupo de amigos do condutor, bem como às informações de localização dos mesmos, permitia ao sistema de navegação por GPS do veículo uma filtragem do conteúdo demonstrado no mapa. Este conteúdo podia materializar-se em "sítios de interesse" marcados pelos utilizadores, ou mesmo na localização atual ou futura do veículo. A utilização da filtragem de conteúdos com auxílio a redes sociais significa que o sistema ganhou a capacidade de apresentar informação socialmente relevante ao condutor, i.e., ao limitar os conteúdos apresentados no mapa (apresentando somente aqueles fornecidos pelos utilizadores considerados amigos do condutor na rede social), o condutor pode limitar a sua atenção a informações relevantes dentro do seu círculo social.

Mais recentemente, uma equipa de alunos da Universidade de Michigan desenvolveu uma aplicação, denominada *Caravan Tracker*, que apresentou um conceito diferente do proposto na dissertação de mestrado referida no parágrafo anterior, visto que a aplicação *Caravan Tracker* não pretende a utilização de redes sociais já existentes, mas sim estabelecer ligações sociais entre um conjunto de viajantes [GigaOm, 2010]. As ligações estabelecidas através da utilização da aplicação permitem aos seus utilizadores a partilha de informações acerca da quantidade de combustível existente nos seus veículos (e comparar essa informação), bem como a partilha de "pontos de interesse" entre os indivíduos deste círculo social.

Algumas marcas comerciais começam já também a integrar redes sociais nos seus veículos, como é o caso da *Toyota*, que se encontra a desenvolver, no presente momento de escrita desta dissertação, uma rede social para os seus veículos elétricos.

Esta rede social pretende oferecer diversas funcionalidades ao seu utilizador, tal como o envio de um aviso, para o telemóvel do condutor, de que o nível de energia do seu veículo é baixo [Reports, 2011]. A *Toyota* pretende ainda que o veículo possa comparar o seu estado atual com o estado de outros veículos idênticos, recorrendo-se dessa mesma ligação à rede social. Outra marca comercial que já explorou as possibilidades da utilização das redes sociais em veículos foi a *Ford*, cujos engenheiros realizaram uma viagem pelos Estados Unidos na qual o veículo em que viajaram enviava mensagens para a rede social *Twitter* [Times, 2010]. As mensagens enviadas pelo veículo da *Ford* eram informações acerca do estado do próprio veículo ou de condições alheias ao veículo (o estado do tempo, o trânsito, etc.), sendo que as informações alheias ao veículo eram dadas por sensores ligados ao mesmo.

Capítulo 3

Apresentação de Informação em Veículos

A construção da interface para um veículo traz consigo um conjunto de preocupações que a construção de interfaces para outro tipo de sistemas não possui. A apresentação de informação e pedidos de *input* para este tipo de sistemas deve ter em conta as limitações intrínsecas à prática da condução, na qual os desvios de atenção em relação à informação vinda da estrada podem ser desastrosos.

Atualmente, pode verificar-se que existe um variado conjunto de novas tecnologias que são utilizadas nos veículos modernos, pelo que se torna indispensável pensar na forma como as mais variadas informações são apresentadas e pedidas ao condutor [de Oliveira Pereira, 2009].

Devido à multiplicidade de informações com que um condutor tem de lidar na condução do seu veículo, é necessário que quem constrói a interface para um veículo possa guiar-se por um conjunto de regras ou princípios que lhe permitam uma satisfatória e segura apresentação dos dados pertencentes a este sistema. Assim, nesta secção serão discutidos alguns princípios de apresentação de informação em veículos, que auxiliarão no desenho final da interface relativa a este trabalho. Os princípios discutidos nesta secção estão de acordo com o documento *"Statement of Principles, Criteria and Verification Procedures on Driver Interactions with Advanced In-Vehicle Information and Communication Systems"*, produzido pela *Alliance of Automobile Manufacturers* [AAM, 2006] e que produziu este documento de forma a criar um conjunto de critérios e procedimentos específicos que devem ser seguidos pelos fabricantes de automóveis de forma a definir sistemas de informação e comunicação em veículos automóveis. Apesar dos princípios contidos neste docu-

Princípio	**Descrição**
(a)	A Interface deve ser desenhada de modo a que as tarefas que permite desempenhar sejam feitas com breves *glances* que não afetem gravemente a tarefa de condução
(b)	Devem ser usados *standards* de legibilidade, ícones, símbolos, palavras, acrónimos ou abreviações. Se tais *standards* não existirem, devem ser usadas diretrizes de *design* ou dados empíricos
(c)	A informação relevante à tarefa de condução deve ser precisa e atempada
(d)	O sistema não deve requerer que o condutor retire mais do que uma mão do volante
(e)	O sistema não deve requerer sequências ininterruptas de interações manuais ou visíveis
(f)	O condutor deve poder controlar, salvo algumas exceções, o ritmo de interação com o sistema. O sistema não deve requerer que o condutor forneça *inputs* urgentes
(g)	A resposta do sistema a um *input* do condutor deve ser atempada e claramente percetível
(h)	A informação visual que não está relacionada com a prática da condução e que poderá distrair significativamente o condutor deverá estar inacessível ao condutor durante a condução do veículo
(i)	O sistema não deve permitir que o condutor tenha acesso a tarefas que não sejam dirigidas ao mesmo enquanto o veículo se encontra em movimento
(j)	Informações acerca do estado do veículo que possam ter impacto na segurança dos seus ocupantes devem ser apresentadas ao condutor

Tabela 3.1: Princípios de apresentação de informação em veículos

mento se dirigirem a sistemas de informação e comunicação em automóveis, foram selecionados, para apresentação e discussão nesta secção, os princípios considerados relevantes para o *design* e implementação de interfaces para veículos automóveis em geral. Assim, foram selecionados princípios que se aplicam a painéis de instrumentos "tradicionais" (que contenham os elementos básicos de uma interface de um veículo automóvel, como o velocímetro e o odómetro) e que também se aplicam a sistemas de comunicação e informação em veículos automóveis (tal como sistemas de navegação GPS).

Os princípios selecionados podem ser consultados na tabela 3.1.

A informação visual fornecida pelo *display* de um veículo deve ser providenciada de modo a evitar ao máximo a distração do condutor. O **processamento visual** efetuado pelo condutor ao controlar um veículo deve ser primariamente dirigido às tarefas primárias da prática da condução, i.e., o **processamento visual** do condutor deve procurar atender prioritariamente às questões relacionadas com a condução segura do veículo. De modo a reduzir ao máximo a distração do condutor, é necessário que a tarefa de *design* da interface para veículos se concentre na minimização da

atenção visual do condutor requerida pelo *display* [AAM, 2006]. De modo a responder este problema foi proposto o **princípio (a)**, cuja materialização se divide em duas possíveis alternativas:

1. A definição de um conjunto de regras para minimização da distração do condutor na construção de interfaces para veículos pode ser feita em função do tempo de *glance* que as diversas tarefas da interface requerem. Para tal, devem obedecer-se a dois critérios na implementação da interface para tarefas destinadas a serem realizadas quando o veículo estiver em movimento:

   (a) *Glances* esporádicas(*glances* destinadas a obter informações simples) não devem exceder 2 segundos;

   (b) A realização de uma tarefa na interface não deve exceder um total de 20 segundos de tempo de *glance*.

2. As regras para minimização da distração do condutor podem ainda ser obtidas em função da avaliação do impacto que a interface visual possui na condução do veículo. Para tal, deve-se definir um conjunto de tarefas de referência da interface e efetuar uma comparação da performance de condução(utilizando o mesmo condutor) entre 2 situações: condução sem recurso ao *display* e condução com ocasional uso de *display*. Ao comparar estas duas situações, deve verificar-se que:

   (a) O número de *lane exceedances* ocorridas durante a condução com uso de *display*, na qual se efetua pelo menos uma tarefa de referência da interface, não é maior que o número de *lane exceedances* ocorridas durante a condução sem uso de *display*. Uma *lane exceedance* define-se como a condição em que uma das rodas do veículo de teste excede uma marcação lateral previamente definida para efeitos de teste, como se pode verificar na figura 3.1.

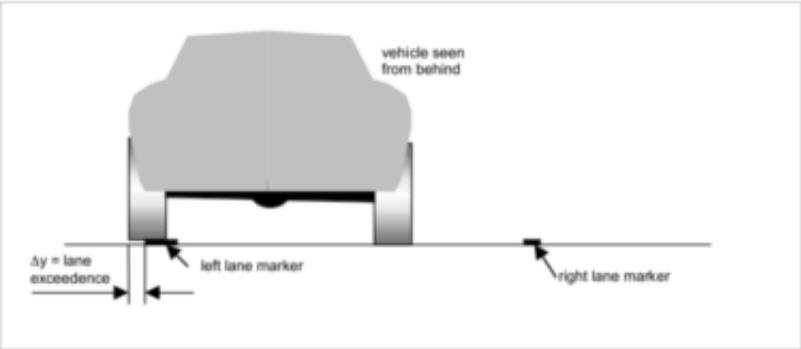

Figura 3.1: *Lane Exceedance*(retirado de [AAM, 2006])

   (b) A variação de *following headway* ocorrida durante a condução com uso de *display*, na qual se efetua pelo menos uma tarefa de referência da interface,

não é menor que a variação de *following headway* ocorrida durante a condução sem uso de *display*. O *following headway* entre dois veículos define-se como a a fração entre a aproximação entre os dois veículos e a sua velocidade.

O **princípio (b)** defende que o uso de *standards* na construção da interface aumenta a possibilidade de compreensão e rápida familiarização do utilizador com o sistema. O uso de *standards* de legibilidade, ícones e símbolos fornece ao sistema um conjunto de características físicas e geométricas que dão à informação visual fornecida pelo mesmo uma maior probabilidade de ser compreendida e rapidamente apreendida. Por outro lado, o crescente número de termos, acrónimos e abreviações necessários para a interação com o veículo tornam indispensável o uso de padrões reconhecíveis pelo condutor.

Para apresentação de ícones, símbolos, palavras, acrónimos e abreviações devem usar-se, sempre que possível, os padrões FMVSS 101, CMVSS 101 e ISO 2575. Em relação à legibilidade do sistema, deve usar-se como referência o padrão ISO 15008. O padrão FMVSS 101 trata-se de um padrão para controlos e *displays* em veículos e contém, entre outras informações, as abreviaturas de unidades físicas que devem ser apresentadas em velocímetros e odómetros. O padrão CMVSS 101 contém informações acerca da localização e identificação dos controlos e *displays* de veículos. O padrão ISO 2575 contém as especificações acerca dos símbolos que devem ser utilizados em controlos e indicadores de veículos. Finalmente, o padrão ISO 15008 especifica os requerimentos mínimos em relação à qualidade de imagem e legibilidade dos *displays* que apresentam informações dinâmicas ao condutor enquanto o veículo se encontra em movimento.

Durante a condução de um veículo, podem ocorrer situações em que a sobrecarga de informação, a apresentação de informação errada ou a apresentação de informação pouco relevante para a tarefa atual do condutor pode levar o condutor a cometer erros críticos. O **princípio (c)** tem por objetivo evitar situações que levem o condutor a tomar decisões erróneas, sendo que para tal a informação apresentada pelo sistema deve ser precisa, atempada e adequada à situação atual do veículo e do condutor. A informação providenciada pelo sistema deve, portanto, poder ser integrada com a informação já apresentada pelo sistema, acentuando a informação existente e reduzindo ao máximo o nível de incerteza e indecisão do condutor em relação às decisões que tem de tomar enquanto conduz.

A quantidade de informação dada pelo sistema deve também ser gerida de modo a evitar que se verifiquem situações de demasiada informação apresentada simultaneamente. Se necessário, deve suprimir-se toda a informação não relacionada com a tarefa atual do condutor, de modo a que a atenção do mesmo se foque nos aspetos fundamentais da ação que realiza em determinado momento.

Manter as duas mãos no volante é um fator que oferece ao condutor uma maior estabilidade e precisão na condução de um veículo, pois existem situações na prática da condução que requerem a utilização das duas mãos no volante de modo a permitir a condução da viatura. No entanto, existem também situações em que é aceitável que o condutor controle o volante, momentaneamente, com uma mão, deixando a outra mão livre para outras ações, tendo em conta que a mão "livre"deve estar imediatamente disponível para segurar o volante caso seja necessário.

A interação com o sistema de controlo de um veículo, de acordo com o **princípio (d)**, deve ser desenhada de modo a que esta mesma interação seja feita de modo a que o utilizador necessite apenas de uma mão para a realização de todas as ações disponíveis pelo sistema. O sistema, por sua vez, deve ser desenhado de modo a que permita, fisicamente, que pelo menos uma mão se encontre a segurar o volante.

Foi já definido um conjunto de critérios de modo a que o sistema obedeça a este princípio:

1. Quando existem controlos de sistema colocados fora do volante, não deve ser necessária a utilização de mais de uma mão para a realização de uma tarefa do sistema durante o ato da condução;
2. Quando os controlos do sistema se encontram no volante e ambas as mãos também se encontram no volante, o sistema não deve exigir uma utilização simultânea de ambas as mãos (exceto quando uma dessas ações requerer apenas um *input* que necessite de apenas um dedo);
3. A distância entre o condutor e os controlos do sistema deve permitir que uma das mãos permaneça no volante;
4. O sistema não deve ser desenhado de forma a que alguma das suas ações necessite da utilização de um "buraco"existente no volante.

A verificação do critério (a) pode ser realizada através da construção de um diagrama de transição de estados que represente as diferentes ações disponíveis no sistema. Após a construção do diagrama, deve proceder-se à sua análise, identificando-se as transições de estado que requerem que o condutor retire uma ou ambas as mãos do volante. Uma vez identificadas estas transições, deve-se contar o número de transições nas quais o condutor retira as duas mãos do volante. O critério (a), que indica que caso os controlos do sistema estejam colocados fora do volante, não deve ser necessária a utilização de mais do que uma mão para a realização de tarefas do sistema durante o ato da condução, considera-se positivamente verificado se o número total de transições de estado pertencentes ao diagrama representativo de ações disponíveis do sistema, for igual a zero.

A verificação do critério (b) pode também ser realizada com recurso a um diagrama de transição de estados. Análogamente ao procedimento efetuado na verificação do

critério (a), devem-se identificar todas as transições de estado que requerem que o condutor retire ambas as mãos do volante, procedendo-se então à sua contagem. O critério (b), que indica que caso os controlos do sistema estejam colocados no volante, não devem ser necessárias ambas as mãos para realização de uma tarefa do sistema (exceto de essa tarefa for possível de realizar com apenas um dedo), considera-se positivamente verificado se a contagem de transições de estado que requerem a utilização das duas mãos for zero. Se existirem transições de estado que requerem mais do que uma mão para a realização da tarefa pretendida, o critério (b) pode considerar-se positivamente verificado se em qualquer uma dessas ações uma das mãos requerer apenas a utilização de um dedo.

A verificação do critério (c) pode ser realizada através da análise de um grupo de teste. Este grupo de teste deverá ser representativo dos condutores do veículo onde se encontra o sistema a testar, sendo que se pretende com a avaliação deste grupo verificar qual a percentagem de condutores que utilizam apenas uma mão para executar tarefas do sistema enquanto conduzem o veículo. Se esta percentagem for de 100%, o critério (c) considera-se positivamente verificado.

Por último, a verificação do critério (d) pode ser também realizada através da análise de um grupo de teste. Esta análise pretende verificar a percentagem de indivíduos que executam todas as tarefas possíveis do sistema, sem utilizar uma única vez um buraco no volante para alcançar um dos controlos do sistema. Se esta percentagem de indivíduos for de 100%, o critério (d) considera-se positivamente verificado.

O ato da condução de um veículo requer bastante atenção do condutor para a situação do trânsito de veículos e de outros aspetos da estrada. Devido a esta necessidade de foco na estrada, é importante que a interação que o condutor realiza com o sistema seja de curta duração (como já foi discutido anteriormente neste trabalho, no princípio (a)). No entanto, existem tarefas que necessitam de um variado número de *inputs* por parte do condutor (como por exemplo, tarefas de sistemas de navegação).

De modo a atender à necessidade de introdução de um variado número de *inputs* por parte do condutor, temos que de acordo com o **princípio (e)** o *design* do sistema deverá ser realizado de forma a que seja possível que o condutor possa interromper uma sequência de inputs de uma tarefa, podendo mais tarde voltar ao mesmo ponto lógico de execução dessa mesma tarefa sem repetir os inputs anteriores. Este princípio foi definido devido ao fato de caso o sistema permita a perda de *inputs* do utilizador durante a execução de uma tarefa, tal possibilidade de perda de *input* pode levar o condutor a tentar fornecer ininterruptamente todos os inputs necessários à tarefa que pretende realizar, visto que o sistema não lhe permite interromper tal sequência de *inputs* sem uma perda total das interações necessárias.

Esta sequência ininterrupta de *inputs* por parte do condutor pode levar a uma distração do mesmo em relação à estrada. Existem estudos que demonstram uma

tendência humana para tomar atenção frequente à situação atual da estrada durante o processo de condução [Green, 1999] [Rockwell, 1988]. Assim, deve ser permitido que o condutor interrompa temporariamente uma sequência de *inputs* com o sistema, de modo a dar atenção à estrada, podendo voltar a esse mesmo ponto de interação quando desejar.

A *Auto Alliance* definiu o seguinte conjunto de critérios para obedecer a este princípio;

1. O sistema não deve permitir nenhuma perda automática de *input* pelo utilizador;
2. Os condutores podem, no entanto, iniciar comandos de sistema que cancelam *inputs* anteriores;
3. Deve ser disponibilizado ao condutor o conjunto de dados anteriormente fornecidos pelo utilizador, durante a execução da tarefa atual, de modo a que o condutor possa retomar a tarefa interrompida;
4. Se necessário, o sistema deve auxiliar o condutor a voltar ao ponto correto da sequência de *inputs* introduzida, ou a determinar a próxima ação necessária. O sistema poderá também fornecer informação relevante ao condutor em relação ao *input* necessário para continuar a sequência;
5. O sistema pode reverter automaticamente a um estado anterior (após um período de tempo previamente definido de inatividade do condutor) sem necessidade de *input* do utilizador para esse efeito, desde que não se trate de uma tarefa que não afete funções de segurança (como tarefas de ajuste de som ou contraste) e que o estado atual da tarefa possa ser de novo atingido com pouco esforço do condutor.

Pode afirmar-se que o condutor controla, parcialmente, o ritmo de interação com o sistema se esse mesmo sistema estiver de acordo com o princípio (e). No entanto, para se poder efetivamente afirmar que o ritmo de interação com o sistema é controlado pelo condutor é necessário que o sistema esteja de acordo com o conjunto de critérios que garantem que o sistema obedece ao **princípio (f)**, e que serão enumerados mais abaixo. Estes critérios permitem também assegurar que o sistema não obrigue o condutor a fornecer *inputs* urgentes.

É fundamental que o sistema de interação do condutor com o veículo não obrigue o condutor a fornecer informações urgentes (i.e., que não obrigue o condutor a retirar a sua atenção da estrada para fornecer um *input* imediato ao sistema) para tarefas temporalmente discretas. Um exemplo desta situação é o de o condutor receber uma chamada telefónica. Ao receber a chamada, o sistema não deve tratar essa informação da mesma forma que trata informações críticas ao funcionamento do veículo (como por exemplo, um aviso de falta de combustível), devendo o sistema

possuir mecanismos que permitam que o condutor ignore a chamada telefónica ou mecanismos que gravem a mensagem telefónica. Se o sistema não se comportar de forma a permitir que o condutor não forneça informações críticas enquanto conduz, o condutor pode sentir-se pressionado a fornecer esse *input* e a provocar uma distração do mesmo em relação à estrada.

Os critérios estabelecidos para que o sistema se comporte de acordo com o **princípio (f)** são:

1. No caso do sistema fornecer um pedido de *input* ao condutor, deve ser possível que o condutor ignore esse *input*, atrase esse *input* ou suspenda temporariamente os pedidos de *input* do sistema;
2. As opções de não resposta imediata ao sistema referidas no critério 1 devem ser claramente expostas ao condutor (através de mensagens do sistema, por exemplo);
3. No caso do condutor optar por não responder a um pedido de *input* do sistema, deve existir um período de tempo razoável, após o qual o sistema deve voltar para um estado *default* previsível e apropriado para a tarefa iniciada pelo condutor;
4. As ocasiões nas quais o sistema efetua pedidos de *input* ou mostra mensagens ao condutor devem ser apropriadas para a tarefa que designam;
5. As mensagens visuais de sistema que não implicam uma resposta imediata do condutor devem ser claramente distintas das mensagens visuais de sistema que implicam uma resposta imediata do condutor.

Durante o processo de implementação do sistema de interação entre o condutor e o veículo deve garantir-se que o sistema não forneça respostas ambíguas, incertas ou atrasadas em relação ao estado atual do veículo. Se as respostas de sistema não forem precisas, a confiança do condutor em relação ao sistema pode ser baixa, o que dificulta a tarefa da condução. A incerteza do condutor acerca da correta receção do sistema de um *input* de utilizador pode também levar o mesmo a distrair-se da situação atual da estrada.

Uma resposta do sistema pode ser um *feedback* de aviso em relação à receção de um *input* de utilizador (como por exemplo, um sinal sonoro) ou a resposta efetiva do sistema ao *input* de utilizador (como por exemplo, a apresentação de uma rota no sistema de navegação). Considera-se que uma resposta de sistema é atempada se for facilmente percetível que o sistema está a responder de forma correta. Considera-se também que uma resposta de sistema é claramente percetível se for óbvio para o condutor que o sistema recebeu o *input* do condutor e se a consequência de tal *input* de utilizador é igualmente óbvia para o condutor.

O critério que deve ser seguido de forma a assegurar que o sistema está de acordo com este princípio, o **princípio (g)**, é o critério definido na ISO 15005, que indica que uma resposta de sistema para um *input* de utilizador não deve exceder os 250 milisegundos. No entanto, se este tempo de resposta for superior a 2 segundos, o sistema deve apresentar uma mensagem informativa que indique que o pedido do condutor está a ser processado.

A modalidade visual é fulcral para a segura condução de um veículo. Assim, qualquer limitação, fornecida ao condutor, da informação visual da estrada poderá causar distrações para a tarefa primária de condução do veículo. De modo a evitar situações potencialmente perigosas para a segurança dos ocupantes do veículo, toda a apresentação de informação que não está relacionada com a prática da condução (como a visualização de um vídeo, ou um *scroll* de texto) deve ser desligada ou deve ser apresentada de uma forma a que não permita que o condutor tenha acesso visual à mesma.

De modo a que o sistema obedeça ao **princípio (h)**, o condutor não deve ter acesso visual a:

1. Televisão;
2. Vídeos não relacionados com a atual tarefa de condução;
3. Jogos e outras imagens dinâmicas;
4. Imagens detalhadas, como o interior de um edifício ou de um produto;
5. Outros itens que não estão discriminados nesta lista, considerados como potenciais distratores da tarefa da condução, e que deverão ser analisados durante o processo de implementação do sistema.

As funções de sistema que não são dirigidas ao condutor são aquelas que assim o são consideradas por quem implementou o sistema. Este princípio tem como objetivo tornar claras ao condutor as intenções de uso do sistema. Assim, se o sistema estiver de acordo com o **princípio (i)**, o condutor será responsável pelas consequências de tentar aceder a funções às quais não deve aceder enquanto conduz o veículo.

De modo a que o sistema se considere de acordo com este princípio, o seu *design* deve estar feito de modo a que as funções de sistema que devem ser inacessíveis ao condutor sejam claramente distinguidas das funções de sistema que devem estar acessíveis ao condutor.

Pode tornar-se perigoso para a segurança dos ocupantes de um veículo se o condutor não possuir informação suficiente acerca do estado atual do veículo. Deste modo, qualquer alteração relevante ao estado do veículo (como uma avaria que afete a performance do veículo) deve ser claramente visível ao condutor. Ao seguir o

princípio (j), assegura-se que o condutor possui informação suficiente para tomar decisões responsáveis na condução do veículo.

Usualmente, os veículos fornecem informações acerca da sua mudança de estado, apresentando indicações do "disparo" do sistema de travagem de ABS, por exemplo. De modo a estar de acordo com este princípio, o sistema de interação do condutor com o veículo deve apresentar indicações relevantes sempre que for detetado um problema no veículo que afete a condução do mesmo.

Capítulo 4

Ambiente de desenvolvimento

Este capítulo irá descrever a implementação de um sistema que permite a definição de um painel de instrumentos para um veículo. O sistema descrito neste capítulo permite a definição de um conjunto de componentes que serão visualizados no *display* de um veículo e permite também a definição do comportamento destes componentes.

Os painéis de instrumentos que se podem encontrar em veículos modernos apresentam diversas modalidades de *input* e *output* na representação de informação para os seus condutores, como já foi referido na secção 2.2. No entanto, o trabalho desenvolvido para esta dissertação refere-se apenas a interfaces de veículos que utilizam somente a modalidade visual como *input* e *output* de informação. Assume-se então que a representação de informações do veículo será feita através de um *display* e que a interação do condutor com o veículo será também feita através deste *display*. De modo a que a introdução de informação do utilizador ao veículo seja feita de uma forma simples e segura, assume-se também que este *display* é *touch-screen*, de forma a que o mesmo possa receber informações sem o auxílio de dispositivos adicionais (como um rato ou um teclado). É no entanto possível efetuar extensões ao sistema desenvolvido de modo a que sejam adicionadas modalidades de interação alternativas (como o reconhecimento de voz), mas esta abordagem está fora do âmbito desta dissertação.

4.1 Componentes habituais no painel de instrumentos de um automóvel

O painel de instrumentos de um veículo automóvel é uma ferramenta que contém um conjunto de componentes que fornecem, ao condutor do automóvel, um conjunto de informações fundamentais acerca do veículo. Os componentes que habitualmente se encontram disponíveis em painéis de instrumentos de veículos automóveis são:

1. **Velocímetro** - componente que indica a velocidade instantânea de um veículo.
2. **Taquímetro** - componente que indica o número de rotações do motor do veículo por minuto (é também geralmente designado por conta-rotações).
3. **Odómetro** - componente que indica a distância percorrida pelo veículo ao longo do seu tempo de vida.
4. **Termómetro** - componente que indica a temperatura do líquido de refrigeração do veículo.
5. **Indicador de nível de combustível** - componente que indica a quantidade de combustível existente no depósito de combustível do veículo.
6. **Indicadores de luzes** - os indicadores de luzes existentes em veículos automóveis são vários, pelo que os mais usuais são os indicadores de mudança de direção para a esquerda e para a direita (as usuais luzes de "piscas"), os indicadores do nível de luz de um veículo (nível mínimo, médio ou máximo), e os indicadores de farol de nevoeiro e de luzes de perigo (que sinalizam, respetivamente, se o farol de nevoeiro do veículo está ligado ou se as luzes de emergência do veículo estão ligadas).
7. **Indicador de Aviso de Pressão do óleo** - componente que indica se o nível de pressão do óleo está normal para o bom funcionamento do veículo.
8. **Indicador de Aviso de Bateria** - componente que indica se a bateria do veículo está ou não a receber carga.
9. **Indicador de Travão de mão** - componente que indica se o veículo está ou não travado com o travão de mão.
10. **Indicador de ABS** - componente que indica se o sistema de travagem ABS do veículo se encontra ou não ativo.

4.2 Linhas orientadoras do desenvolvimento do sistema

Considerou-se que o desenvolvimento de um sistema que permita a definição de painéis de instrumentos para veículos deve ser feito de forma a que a **elaboração**

de um painel possa ser facilmente modificada durante o seu ciclo de desenvolvimento. Existem diversas formas de dispor, num *display*, os diversos componentes que podem fazer parte do painel de instrumentos de um veículo. Devido a esta multiplicidade de componentes disponíveis, é natural que o indivíduo (ou indivíduos) responsável pela elaboração desta interface deseje experimentar diversas formas de representação possíveis para o painel. Assim, decidiu-se que o sistema a desenvolver deve permitir que sejam facilmente efetuadas alterações no aspeto gráfico da interface do veículo. A alteração do aspeto ou posicionamento de um ou mais componentes da interface não deve, porém, afetar o comportamento dos restantes componentes previamente definidos.

Considerou-se também que o sistema a desenvolver deve fazer uma **separação entre a representação gráfica e o comportamento dos componentes do painel de instrumentos**. Esta separação é feita de forma a permitir que o desenvolvimento destas duas facetas distintas do desenvolvimento do painel sejam realizadas por indivíduos cujos conhecimentos se adequem ao trabalho que se lhes apresenta. Assim, assume-se que o indivíduo responsável pela elaboração da representação gráfica da interface (o *designer*) pode não possuir um vasto conhecimento informático. Por outro lado, assume-se que o indivíduo responsável pela definição do comportamento dos componentes presentes no painel de instrumentos pode não possuir um vasto conhecimento em *design*. Ao separar a representação gráfica dos componentes da definição de comportamento dos mesmos, facilita-se a interdisciplinaridade do grupo de trabalho responsável pela elaboração do painel de instrumentos.

Uma vez separada a definição do comportamento dos componentes do painel de instrumentos da representação gráfica dos mesmos, foi também pensado como se podia permitir a definição visual da interface de forma a que a determinação do comportamento dos componentes possa ser realizada de várias formas possíveis. Deve permitir-se que o indivíduo responsável pela definição do comportamento dos componentes utilize as ferramentas que desejar para alcançar tal objetivo (i.e., deve permitir-se que este indivíduo possa alcançar este objetivo utilizando as linguagens de programação, *frameworks*, etc. que desejar). Deste modo, a **componente de elaboração da representação gráfica do painel de instrumentos deve ser suficientemente flexível de forma a permitir uma simples interação com as mais diversas tecnologias que podem ser utilizadas na componente de definição do comportamento dos componentes presentes na interface do veículo**.

Finalmente, considerou-se que os componentes que serão visualizados no veículo possuem um nível de importância diferente entre eles aquando do momento de condução do veículo. Num cenário de condução real, existem componentes do painel de instrumentos que são mais importantes para o condutor que outros. Um exemplo desta situação é a dos componentes velocímetro e sistema de navegação GPS, pois não deve ser permitido que um veículo opere sem fornecer a sua velocidade instantânea ao seu

condutor mas, por outro lado, um veículo pode ser conduzido sem que o seu condutor tenha acesso às rotas disponibilizadas pelo sistema de navegação GPS. Assim, o sistema que permite a definição de um painel de instrumentos deve **efetuar uma separação entre os componentes indispensáveis à condução do veículo e os componentes que não são indispensáveis à condução do mesmo**. Este sistema não deve também permitir o funcionamento do painel se algum dos seus componentes indispensáveis não funcionar. Por outro lado, se um dos componentes não indispensáveis do painel apresentar algum problema, o sistema deve garantir o funcionamento dos restantes componentes.

4.3 Arquitetura do sistema

Antes de proceder à implementação do sistema de definição da interface de um veículo foi definida a arquitetura do mesmo de modo a obedecer às propriedades apresentadas na secção anterior. Para uma melhor compreensão da arquitetura do sistema, a mesma encontra-se dividida em duas partes: **arquitetura geral** e **arquitetura do módulo da interface gráfica**.

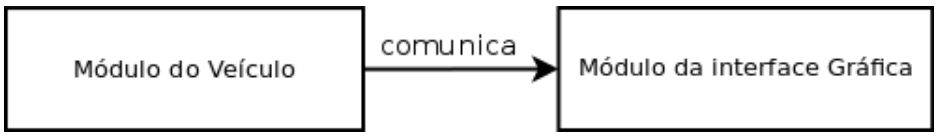

Figura 4.1: Arquitetura geral do sistema

A **arquitetura geral** do sistema pode ser consultada na figura 4.1. Como se pode verificar, o sistema está dividido em dois módulos principais: o **módulo do veículo** e o **módulo da interface gráfica**. A figura 4.1 diz respeito à relação entre o módulo que obtém e interpreta as informações que provém do veículo (módulo do veículo) e que serão posteriormente enviadas para o módulo que é responsável pela visualização destas informações no *display* do veículo (módulo da interface gráfica). O objetivo do módulo do veículo será então o de obter e interpretar informações do veículo que se pretendem apresentar na interface do mesmo. Estas informações devem ser propriamente normalizadas de forma a que o módulo do veículo possa apresentá-las sem que seja necessário um esforço adicional de interpretação das mesmas. A elaboração do módulo dos sinais não faz parte deste trabalho, pelo que os detalhes acerca da sua implementação não serão apresentados nesta dissertação.

A **arquitetura do módulo da interface gráfica**, que pode ser consultada na figura 4.2, diz respeito à arquitetura do módulo da interface gráfica, referido no parágrafo anterior, e é composta por quatro componentes: **Módulo Indispensáveis**, **Módulo Adicionais**, **XML** e **Interface Gráfica**. O primeiro e o segundo componentes dizem respeito ao à definição dos componentes indispensáveis e componentes adicionais da interface. A divisão destes módulos garante a separação dos compo-

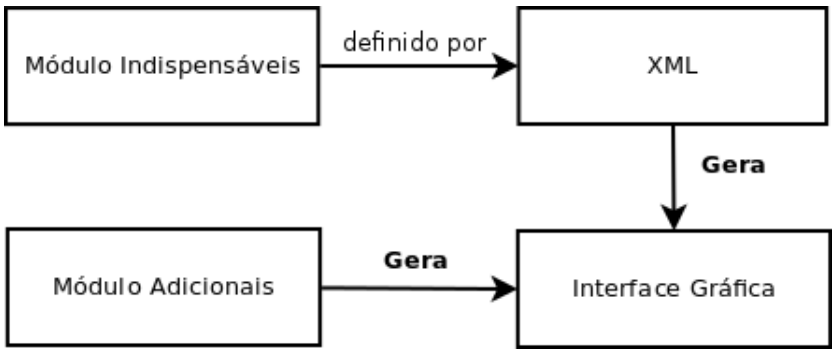

Figura 4.2: Arquitetura do módulo da interface gráfica

nentes que são indispensáveis dos componentes que não são indispensáveis durante a condução de um veículo automóvel, permitindo assim que a interface do veículo continue operacional se alguns dos componentes não indispensáveis não funcione. O terceiro componente é o componente que permite a definição dos componentes que serão incluídos na interface do veículo. O nome deste componente é **XML** pois esta é precisamente a tecnologia utilizada para a sua implementação. A tecnologia XML[1] é uma *Markup Language* (ou linguagem de marcação, que é uma linguagem que utiliza códigos que são sintaticamente distinguíveis do restante texto [webopedia, 2011]) é uma especificação da *XML Working Group* cujo atributo principal é o de poder ser utilizada na comunicação entre várias tecnologias diferentes. A especificação de um documento XML permite a organização hierárquica de um conjunto de dados através da representação de entidades e de propriedades dessas entidades em *tags* (ou etiquetas) e dos seus respetivos valores [W3C, 2008] [Tecmundo, 2009]. A utilização desta tecnologia possibilita a definição dos vários componentes que vão ser incluídos na interface do veículo de uma forma simples para o indivíduo responsável pela definição do painel de instrumentos do veículo. Um exemplo disto é o da definição do componente **velocímetro** e das suas propriedades de posicionamento no *display*, que pode ser definido em XML da seguinte forma:

<velocimetro>
<posicao-x> 10 **</posicao-x>**
<posicao-y> 20 **</posicao-y>**
...
</velocimetro>

Neste exemplo, a entidade **velocimetro** é dada pelas *tags* de entrada e saída **<velocimetro>** e **</velocimetro>**, as propriedades desta entidade (a posição do velocímetro no *display*) são dadas pelas *tags* de entrada e saída **<posicao-x>** **</posicao-x>** e **<posicao-y>** **</posicao-y>** e os valores 10 e 20 representam os valores pertencentes às propriedades de posicionamento do velocímetro. A

[1] XML - Extensible Markup Language

utilização do XML na definição dos componentes da interface permite então que a representação gráfica do painel de instrumentos seja definida por várias tecnologias diferentes.

A separação entre o componente **XML** e o componente **Interface Gráfica** separa a representação gráfica do painel de instrumentos e o comportamento dos seus componentes, o que facilita a modificação do aspeto gráfico do painel de instrumentos durante o seu ciclo de desenvolvimento e facilita também a interdisciplinaridade dos indivíduos responsáveis pela definição do painel. Os componentes presentes no **Módulo Indispensáveis** irão ser definidos no módulo **XML**. Por outro lado, os componentes presentes no **Módulo Adicionais** não irão estar definidos no documento XML pois estes não fazem parte dos componentes básicos do painel de instrumentos de um veículo e, por isso, considerou-se que não seria possível definir entidades no documento XML que englobassem todas os possíveis componentes que podiam ser incluídos neste módulo. A interface gráfica do veículo será então constituída pelos elementos descritos no módulo **XML** e pelos elementos implementados no **Módulo Adicionais**.

4.4 Documento XML de definição de componentes

Tal como foi referido na secção anterior, os componentes do **Módulo Indispensáveis** que estarão presentes no painel de instrumentos do veículo serão definidos num documento XML que contém *tags* para cada um destes componentes e *tags* relativas às suas propriedades. Denote-se que um componente, à exceção dos componentes *panel*, *background* e *screen*, é identificado pelo seu tipo e pela propriedade *id*, pelo que esta é referida na maioria dos componentes e é uma propriedade de valor alfanumérico que distingue os diversos componentes presentes no documento XML. O conjunto de elementos, presentes no documento XML, definidos para o trabalho apresentado nesta dissertação são:

1. **panel**
   Propriedades: *Não aplicável*

   O elemento **panel** será o elemento raiz de todos os restantes elementos, e especifica o início da definição de um novo painel de instrumentos.

2. **name**
   Propriedades: *Não aplicável*

   O elemento **name** é um elemento que não contém propriedades e que serve para nomear o painel que se pretende definir, pelo que o seu valor deverá ser um valor alfanumérico que define o nome do painel.

3. **screen**
   Propriedades: *width* e *height*

   O elemento **screen** especifica o *display* do veículo, pelo que as suas propriedades, **width** e **height**, possuirão valores numéricos que especificam o comprimento e a altura deste *display*.

4. **background**
   Propriedades: *file* e *refreshable*

   O elemento **background** define uma imagem de fundo estática que ocupará todo o *display* do veículo. A propriedade **file** possuirá um valor alfanumérico que indicará a localização da imagem que ocupará o fundo do *display*. A propriedade **refreshable** possuirá um valor booleano (i.e., um valor *true* ou *false*) e especificará se esta imagem de fundo deverá poder ser redesenhada. Esta propriedade de redesenho da imagem de fundo poderá, por exemplo, ser colocada a *false* por questões de performance ou a *true* para permitir a definição de uma imagem de fundo animada.

5. **pointerGauge** e **pointer**
   Propriedades pointerGauge: *id*, *file*, *x* e *y*
   Propriedades pointer: *id*, *color*, *p0x*, *p0y*, *p1x*, *p1y*, *p2x*, *p2y*, *vmin* e *vmax*

   O elemento **pointerGauge** especifica um componente que "imita" um componente analógico que utiliza um ponteiro para referir determinados valores, tal como o componente representado na figura 4.3. A propriedade **file** possuirá um valor alfanumérico que especifica a localização da imagem de fundo deste componente, enquanto que as propriedades **x** e **y** especificarão a sua posição. Dentro das *tags* limitadores do elemento **pointerGauge** deverá encontrar-se um elemento **pointer**, que especificará o ponteiro do componente anterior, e no qual a propriedade **color** especificará a cor do ponteiro, as propriedades **p0x**, **p0y** e **p1x**, **p1y** especificarão, respetivamente, a posição inicial do ponteiro e a posição final do mesmo quando este está apontado para o valor mínimo da imagem do elemento **pointerGauge**, enquanto que as propriedades **p2x** e **p2y** especificarão a posição final do ponteiro quando este está apontado para o valor máximo da imagem do elemento **pointerGauge**. As propriedades **vmin** e **vmax** especificam os valores mínimo e máximo entre os quais poderão variar os valores apontados pelo ponteiro. O elemento **pointerGauge** poderá ser utilizado, por exemplo, na implementação de componentes do painel de instrumentos que normalmente utilizam ponteiros, tais como o velocímetro e o taquímetro.

Figura 4.3: Componente de um veículo que utiliza um ponteiro

6. **levelGauge** e **levelGaugeElement**
   Propriedades levelGauge: *id*, *file*, *x* e *y*
   Propriedades levelGaugeElement: *id*, *elements*, *file*, *x*, *y*, *length* e *height*

   O elemento **levelGauge** especifica um componente que é composto por vários níveis, tal como o elemento representado na figura 4.4. A propriedade **file** especifica a localização da imagem de fundo do elemento, enquanto que as propriedades **x** e **y** especificam a sua posição. Este elemento deverá estar acompanhado por um elemento **levelGaugeElement**, que especifica uma imagem que constituirá um nível do elemento **gaugeElement**. A propriedade **elements** representa o número máximo de elementos do componente, enquanto que o elemento **file** será utilizado para indicar a imagem de fundo que representará os níveis do elemento. As propriedades **x** e **y** representam a posição da imagem referente ao primeiro nível do componente, enquanto que as propriedades **length** e **height** representam, respetivamente, o comprimento e a altura da imagem referente a um nível do **levelGauge**. O elemento **levelGauge** poderá ser utilizado, por exemplo, na implementação de componentes do painel de instrumentos que possuam vários níveis, como o termómetro ou o indicador do nível de combustível, no qual a quantidade de níveis apresentados representariam o nível de combustível do veículo.

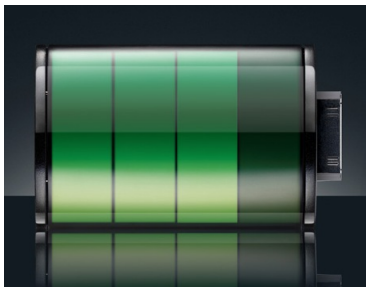

Figura 4.4: Componente de um veículo composta por vários níveis

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

   O elemento **indicator** especifica um indicador referente a uma informação do veículo, tal como o indicador de travão de mão de um veículo (como pode ser consultado na figura 4.5). As propriedades **imageActive** e **imageInactive** referem-se à localização das imagens que representam o indicador, respetivamente, no seu modo ativo ou inativo, enquanto que as propriedades **x** e **y** se referem à posição deste indicador.

Figura 4.5: Indicador do travão de mão de um veículo

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

   O elemento **staticImage** especifica uma imagem estática (i.e., que é imutável no painel de instrumentos do veículo), pelo que as suas propriedades são a propriedade **file**, que especifica a localização da imagem, e as propriedades **x** e **y** que especificam a sua posição.

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

   O elemento **label** especifica uma porção de texto que se pretende apresentar no painel de instrumentos. As propriedades **x** e **y** especificam a sua posição, enquanto que a propriedade **color** representa a cor do texto. As propriedades **fontType** e **fontSyze** dizem respeito à fonte utilizada na escrita do texto e representam, respetivamente, o tipo de fonte utilizada e o tamanho dessa fonte. A propriedade **default** define um valor alfanumérico que será apresentado por defeito, enquanto que a propriedade booleana **refreshable** indicará se este texto poderá ou não ser alterado durante a visualização do painel de instrumentos do veículo. O elemento **label** poderá ser utilizado, por exemplo, para a implementação de um relógio digital no painel de instrumentos do veículo.

10. **button**
    Propriedades: *id*, *imageDefault*, *imagePressed*, *x* e *y*

    O elemento **button** especifica um botão que desencandeará uma ação no painel de instrumentos. As propriedades **imageDefault** e **imagePressed** especificam a localização das imagens do botão quando este não está a ser pressionado e quando este está a ser pressionado, respetivamente, enquanto que as propriedades **x** e **y** especificam a posição do componente.

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

4.5 Implementação

4.5.1 A tecnologia Java

De forma a proceder à implementação do sistema que permite a definição de um painel de instrumentos para um veículo, determinou-se que seria utilizada a linguagem de programação **Java**. O **Java** utiliza uma máquina virtual, a *Java Virtual Machine*, que compila o código escrito na linguagem **Java** para um ficheiro *.class* que contém um código intermediário (o *Java Bytecode*), que será por sua vez interpretado pela *Java Virtual Machine* de modo a traduzir o código intermediário para instruções máquinas nativas ao sistema no qual se executa a aplicação [Oracle, 2011b]. A *Java Virtual Machine* está disponível em vários sistemas operativos (como o *Windows*, o *Linux*, o *Solaris OS*, etc.), conferindo portabilidade ao código que é escrito na linguagem **Java**. Assim, devido à incerteza acerca de qual será o sistema no qual será executado o painel de instrumentos do veículo, e como o código **Java** pode ser executado em diversos sistemas, decidiu-se que a implementação do sistema que permite a definição de um painel de instrumentos para um veículo seria realizada nesta linguagem.

A plataforma **Java** possui uma ferramenta, o *Java Swing*, que é constituída por um conjunto de componentes para a construção de uma GUI[2] [Oracle, 2011f]. Considerou-se que esta ferramenta é ideal para a implementação do painel de instrumentos pois permite a utilização dos componentes *JFrame* e *JPanel*. O *JFrame* é um componente que é utilizado para desenhar janelas num ecrã de acordo com um comprimento e altura especificados pelo programador [Oracle, 2011c], sendo por isso utilizado para a definição de uma janela de visualização do painel de instrumentos no *display* do veículo. De forma a desenhar num ecrã, o *JFrame* é adicionado um componente *JPanel* numa janela *JFrame*. O *JPanel* é um componente que serve de *container* para outros componentes (i.e., o *JPanel* é utilizado para organizar um conjunto de

[2]GUI - Graphical User Interface

componentes filhos) [Oracle, 2011d]. O *JPanel* também pode ser usado como uma superfície para o desenho de componentes "*custom*" (i.e., componentes que não são nativos do Java e que são definidos pelo programador) [Oracle, 2011e], e será por isso utilizado para o desenho dos componentes do painel de instrumentos no *display* do veículo.

A classe *JPanel*, que é a classe da plataforma **Java** que implementa o componente *JPanel*, possui o método *paintComponent()* que deverá conter todo o código de desenho. Este método é invocado por outro método da classe *JPanel*, o método *paint()*, que cada vez que efetua um novo desenho no ecrã irá, por defeito, redesenhar toda a área de desenho do ecrã [Oracle, 2011a] [Oracle, 2011g]. A posição inicial de desenho é a posição *(0,0)*, e corresponde ao primeiro ponto do canto superior esquerdo da superfície de desenho.

No entanto, pode considerar-se que este comportamento do método *paint()* se torna dispendioso em termos de eficiência de execução, pois nem sempre será necessário o redesenho de todo o ecrã para atualizar os componentes do painel de instrumentos, existindo até alguns componentes que nunca necessitarão de ser atualizados. De forma a resolver esta situação, poderá ser feito o *override* (redefinição de um método de uma superclasse por parte de uma subclasse [Martins, 2009]) do método *paint()* de forma a que apenas alguns componentes sejam atualizados constantemente.

4.5.2 *Parsing* do documento XML

A definição dos componentes que serão incluídos no painel de instrumentos do veículo é feita num documento XML, tal como foi referido na secção 4.3. De forma a retirar deste documento os componentes e as respetivas propriedades, é necessário realizar um *parsing* do mesmo. Um *parsing* de um documento XML é a realização de uma leitura do mesmo através de um *parser* e, aproveitando o agrupamento de informação em *tags*, retirar as informações pretendidas [McLaughlin, 2000]. Em *Java* pode ser utilizado o mecanismo SAX[3] para efetuar o *parsing* de um documento XML. O *parsing* do documento XML utilizando o SAX é feito de forma sequencial durante a leitura do documento, o que implica uma vantagem em termos de eficiência, pois isto significa que o *parsing* do documento é realizado sequencialmente, não sendo necessária a leitura de todo o documento XML para memória. Para efetuar este *parsing*, é necessário definir um *handler* que defina as ações a realizar durante a leitura do documento. Um *handler* é um mecanismo no qual se definem as ações a realizar em resposta a um evento, sendo que, neste caso, o *handler* irá ser utilizado para guardar as informações providenciadas durante o evento de leitura do documento XML.

O resultado do *parsing* do documento XML de componentes do painel de instru-

[3] SAX - Simple API for XML

mentos será uma lista de componentes. Para atingir tal objetivo, foi definido um conjunto de classes que especificam cada um dos possíveis componentes que se podem encontrar no documento XML e que possuem os respetivos atributos de cada componente. Estas classes pertencem a um tipo mais geral, denominado **ObjetoInterface**, que será utilizado para especificar o tipo de objetos que são adicionados à lista de componentes devolvida como resultado do *parsing* do documento XML. As classes criadas para especificação dos vários tipos de componentes são as seguintes:

1. **Screen** - classe que especifica o componente *screen*
2. **Background** - classe que especifica o componente *background*
3. **PointerGauge** - classe que especifica o componente *pointerGauge*
4. **Pointer** - classe que especifica o ponteiro do componente *pointerGauge*
5. **LevelGauge** - classe que especifica o componente *levelGauge*
6. **LevelElement** - classe que especifica um nível do componente *levelGauge*
7. **Notifier** - classe que especifica o componente *indicator*
8. **StaticImage** - classe que especifica o componente *staticImage*
9. **Label** - classe que especifica o componente *label*
10. **Button** - classe que especifica o componente *button*

Durante o *parsing* do documento XML, ao receber uma *tag* de entrada de um componente (como por exemplo, <screen>), é dada indicação ao *parser* que as próximas *tags* lidas dirão respeito às propriedades do componente em questão, sendo então instanciado um objeto do tipo do componente respetivo. Durante a leitura das sucessivas *tags* relativas às propriedades do componente que está atualmente a ser acedido no documento XML, os respetivos valores serão guardados no objeto que foi previamente instanciado. Ao receber uma *tag* de saída de um componente (como por exemplo, </screen>), o objeto será adicionado à lista de componentes presentes no documento XML e a tarefa de *parsing* continua até ser atingido o fim do documento.

4.5.3 Divisão dos componentes

Uma vez realizado o *parsing* do documento XML de definição dos componentes do painel de instrumentos, é realizada uma procura por um objeto do tipo *Screen* na lista de componentes resultante deste *parsing*. O objeto *Screen* irá conter os atributos *width* e *height*, correspondentes ao comprimento e altura do ecrã de visualização de componentes, que serão utilizados para a criação de uma *JFrame* cujo propósito será a visualização destes componentes. De seguida, serão criadas duas listas do

tipo *ObjetoInterface*, que servirão para dividir os objetos presentes na lista geral de componentes. Estas listas dividirão os componentes por *refreshable* e não *refreshable* e a divisão será feita de acordo com o valor booleano da propriedade *refreshable* presente em alguns componentes no documento XML. Assume-se que os componentes que não apresentam esta propriedade (tal como os componentes *PointerGauge* e *StaticImage*) são implicitamente não *refreshable*, pelo que irão ser colocados na lista de componentes não *refreshable*. Esta divisão de componentes é realizada devido ao facto de que os componentes não *refreshable* não possuem valores que necessitem de constante atualização e, por isso, não necessitam de ser constantemente atualizados nem redesenhados. Esta divisão é, portanto, feita de forma a poupar recursos de processamento do sistema.

4.5.4 Visão geral do desenho dos componentes no ecrã

Uma vez obtidas as listas de componentes do painel de instrumentos, pode iniciar-se o ciclo de desenho destes componentes no ecrã. Tal como foi referido em 4.5.1, é feito um *override* ao método *paint()* de forma a que não seja necessário redesenhar os componentes que não necessitam de ser constantemente atualizados. Assim, o método *paint()* foi implementado da seguinte forma:

```
por cada iteração de paint(){
        atualiza_componentes_refreshable();

        se primeira iteração:
                pinta_componentes_não_refreshable();

        pinta_componentes_refreshable();
}
```

O método *paint()* inicia-se então pela comunicação com o **Módulo do Veículo** de modo a obter os dados relevantes aos componentes *refreshable*. De seguida, se for a primeira vez que o método *paint()* é invocado, são desenhados no ecrã os componentes não *refreshable*, caso contrário os mesmos não serão atualizados, pois estes componentes não possuem dados que necessitem de atualização. Finalmente, o método *paint()* conclui a sua execução ao desenhar no ecrã os componentes *refreshable*, que por possuírem dados que necessitam de atualização constante, terão de ser redesenhados por cada chamada ao método *paint()*.

Para o desenho dos componentes no ecrã, é utilizada a técnica *double-buffering*. O *double-buffering* é uma técnica na qual os elementos que serão desenhados no ecrã são primeiramente desenhados numa imagem guardada temporariamente em memória. Quando todos os elementos estão desenhados, a imagem temporária é

transferida para a imagem que vai ser desenhada no ecrã [Knudsen, 1999], como pode ser verificado na figura 4.6. Esta técnica é utilizada de modo reduzir os problemas de *flickering* (um problema de desenho de imagens num ecrã no qual ao utilizador é dada a sensação que o ecrã "treme"), uma vez que se esta técnica não fosse utilizada os elementos a desenhar no ecrã eram desenhados um de cada vez, dando ao utilizador a sensação de que as imagens "tremem". No entanto, ao utilizar *double-buffering*, cada desenho dos elementos no ecrã é feito de uma só vez, evitando ao máximo os problemas de *flickering*.

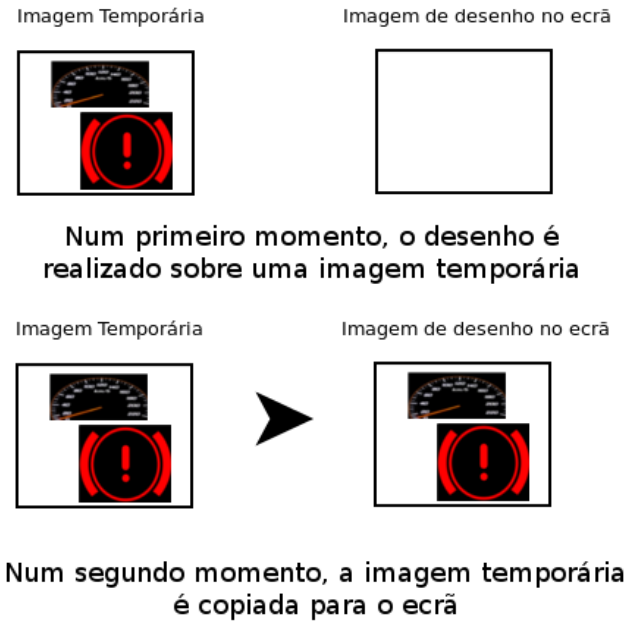

Figura 4.6: Double Buffering

4.5.5 Características específicas do desenho de componentes no ecrã

De forma a que o método *paint()* efetue o desenho dos componentes do painel de instrumentos no ecrã, é necessário que sejam obtidos os valores dos atributos das classes respetivas a cada componente. Uma vez obtidos esses valores, cada componente deverá ser desenhado de acordo com o seu comportamento no painel. Note-se que sempre que nesta secção se referir o desenho de uma imagem no ecrã, refere-se a utilização do método *drawImage()* da classe *Graphics* (presente na API[4] do *Java*) que permite o desenho de uma imagem numa determinada posição do ecrã. Assim, será apresentada nesta secção uma implementação possível do desenho de cada um dos componentes do painel de instrumentos no *display* do veículo.

[4]API - Application Programming Interface

background

A classe *Background* possui apenas um atributo relativo à localização da imagem de fundo que irá ser utilizada no *display* do painel de instrumentos. Para o desenho da imagem de fundo, é instanciado um objeto do tipo *File* (presente na API do *Java*) e é dado a este objeto a localização da imagem. Uma vez obtida a imagem, basta desenhar a mesma na posição (0,0) do ecrã, de modo a que todo o *display* seja preenchido com a imagem de fundo.

pointerGauge

De forma a desenhar no ecrã um elemento que utiliza um ponteiro para especificar valores é necessário obter, por cada iteração do método *paint()*, o respetivo valor para o qual o ponteiro deve apontar. De forma a auxiliar a implementação deste componente, foram criadas as classes *PointerGauge* e *Pointer*. Os atributos que se encontram na primeira classe são a localização da imagem de fundo do componente e a posição desta imagem no ecrã. A segunda classe contém os atributos relativos ao ponteiro: a posição inicial do ponteiro; a posição final do ponteiro quando este está apontado para o valor mínimo da componente; a posição final do ponteiro quando este está apontado para o valor máximo da componente; os valores mínimo e máximos do espaço de valores para o qual o ponteiro irá apontar; a cor do ponteiro. O desenho do ponteiro é feito utilizando o método *drawLine()* da classe *Graphics* que desenha, no ecrã, uma linha entre uma posição inicial e uma posição final. Assim, quando o componente obtém um novo valor, e de forma a desenhar o ponteiro após uma rotação, basta invocar o método *drawLine* com a posição inicial dada no documento XML e terá de ser realizado um cálculo para obter a posição final de acordo com o novo valor obtido. Para atingir o valor da posição final para a rotação do ponteiro, a classe *Pointer* contém um método auxiliar, o método *rotate*, que dado um valor entre o espaço de valores mínimo e máximo, irá devolver a posição final de desenho do ponteiro respetiva a esse valor. A classe *Pointer* contém também um método, *rotationAngle()*, que será invocado cada vez que for instanciado um objeto do tipo *Pointer* e que calcula o ângulo de rotação realizado entre os pontos relativos ao valor mínimo e ao valor máximo do componente apresentado na imagem contida na instância do objeto *PointerGauge*.

O cálculo do ângulo de rotação entre os pontos mínimo e máximo do *PointerGauge* inicia-se pela translação para a origem do ponto inicial (pI) e dos pontos finais ($p1$ e $p2$), relativos ao valor mínimo ($vmin$) e ao valor máximo ($vmax$) do componente, da imagem do componente *PointerGauge*: sejam $xP0$ e $yP0$ a posição inicial do ponteiro, $xP1$ e $yP1$ a posição ($p1$) final do ponteiro quando este está apontado para o valor mínimo do *PointerGauge* e $xP2$ e $yP2$ a posição($p2$) final do ponteiro quando este está apontado para o valor máximo do *PointerGauge*, a translação dos pontos para a origem será dada pelas posições pelas posições ($px1$,$py1$) e ($px2$,$py2$),

na forma:

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

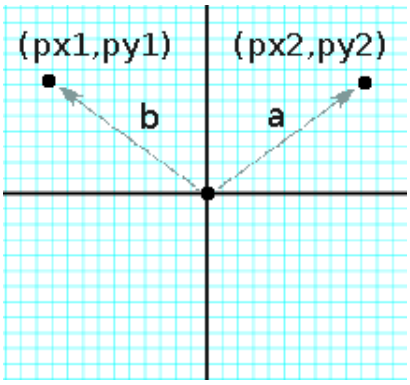

Figura 4.7: Distância a e b dos pontos mínimo e máximo de desenho

Uma vez realizada a translação dos pontos para a origem, será necessário o cálculo da distância entre *pI* e *p1* e o cálculo da distância entre *pI* e *p2*. Estas distâncias serão dadas pelo cálculo da norma do vetor definido pelos pontos *pI* e *p1* e pelo cálculo da norma do vetor definido pelos pontos *pI* e *p2*. Assim, sendo a a distância entre *pI* e *p1*, b a distância entre *pI* e *p2* (como se pode verificar na figura 4.7) e *squareRoot()* a função que calcula a raiz quadrada de um número, as distâncias entre os pontos serão calculadas da seguinte forma:

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

De modo a calcular a posição final de desenho do ponteiro para o valor atual v, terá de ser calculado o ângulo de rotação efetuado entre os pontos *p1* e *p2*. A definição de produto interno entre dois vetores relaciona o ângulo efetuado entre dois vetores (α) com as suas normas, $\|u\|$ e $\|v\|$, na forma:

$$< u, v >= \|u\|.\|v\|.cos(\alpha)$$

Desta definição, retira-se o ângulo realizado entre os dois vetores, na forma:

$$\alpha = arccos(\frac{< u, v >}{\|u\|.\|v\|})$$

Assim, o ângulo, *ang*, realizado entre o vetor definido pelo ponto *p1* e o ponto *p2* necessitará do cálculo do produto interno, *prod*, entre *px1* e *px2*, na forma:

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

Finalmente, o ângulo *ang* será obtido na forma:

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

O ângulo, *ang2*, realizado entre a posição inicial de desenho e a posição atual de v será calculado na forma:

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

Após obtenção do ângulo de rotação para v, o método *rotate* irá utilizar este ângulo(*ang2*) para cálculo da posição para o qual o ponteiro terá de apontar para se referir a v. A posição final de desenho para v, posição *(x,y)*, será obtida através de uma matriz de rotação [et al., 2005]. Os pontos *(xP0,yP0)* e *(xP1,yP1)* serão transladados da origem para a sua posição original, pelo que o cálculo da posição final de desenho para v será obtido por:

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

Assim o cálculo da posição *(x,y)* para o valor v será dado por:

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

levelGauge

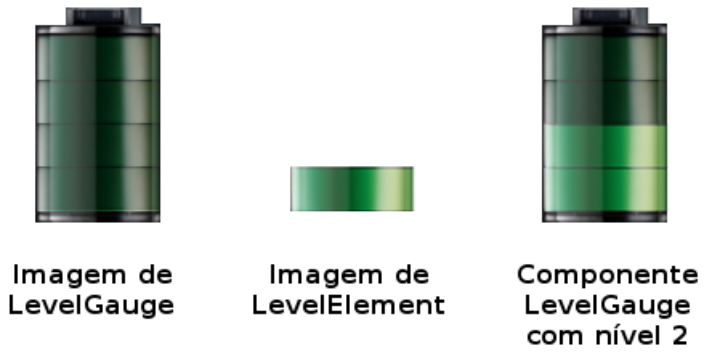

Figura 4.8: Elementos do componente *levelGauge* aplicados no indicador do nível de bateria de um veículo

O componente *levelGauge* é um componente que é constituído por uma imagem e por várias imagens mais pequenas que constituem os vários níveis do componente. O número de níveis apresentados, em determinado instante, no componente permite representar o estado atual do mesmo no painel de instrumentos, como é o caso de um indicador do nível de bateria de um veículo, no qual o número de níveis representa a quantidade de energia restante do veículo (ver figura 4.8). Este componente é implementado utilizando as classes *LevelGauge* e *LevelElement*, que representam, respetivamente, o componente *LevelGauge* e um nível do componente *LevelGauge*. A classe *LevelGauge* possui os atributos de posição(x e y) no ecrã onde se pretende colocar o componente e um atributo correspondente à localização da imagem do componente. A classe *LevelElement* possui os atributos de posição(x e y) do primeiro nível do componente, o seu respetivo comprimento e altura, o número de níveis máximo do componente e a localização da imagem correspondente a um nível do *levelGauge*. A implementação deste componente inicia-se com o desenho, no *display*, da imagem correspondente a uma instância da classe *LevelGauge* na sua respetiva posição. De seguida, e para cada iteração do método *paint()*, deve ser calculado o número atual de níveis que o componente possui. Uma vez obtido o número de níveis do componente, basta desenhar, nas posições de ecrã adequadas, a imagem correspondente a um nível (a imagem contida na instância da classe *LevelElement*) um número de vezes igual ao número de níveis obtidos. A título de exemplo, considere-se um componente *levelGauge* de disposição vertical (como o representado na figura 4.8): sendo *initX* e *initY* a posição do primeiro nível, *alturaNivel* a altura de um nível do componente, *levelImage* a imagem correspondente a um nível e *numberOfLevels* o número de níveis que é necessário representar num determinado instante, o desenho dos níveis do componente *levelGauge* é feito da seguinte forma:

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

O cálculo do número atual de níveis varia consoante a finalidade do componente. A título de exemplo, suponhamos que se pretende utilizar o componente *levelGauge* para implementar um indicador do nível de energia de um veículo elétrico. Para este caso, o número de níveis presentes em cada instante no componente *levelGauge* iria representar a energia restante no veículo. Assim, sabendo a percentagem do total de bateria restante no veículo, *batteryPercentage*, e o número de níveis máximo do componente *levelGauge*, *maxLevels*, o número de níveis do *levelGauge* em determinado instante, *numberOfLevels*, pode calcular-se da seguinte forma:

```
numberOfLevels = (batteryPercentage * maxLevels) / 100;
```

indicator

Um *indicator* é um componente que possui dois estados: o estado ativo e o estado inativo. Para cada um dos estados, terá de ser desenhada no ecrã uma imagem correspondente. Para tal, a classe *Indicator* possui os atributos de localização das duas imagens necessárias, a posição de desenho do componente no ecrã e ainda um atributo booleano que indica se o componente está ou não ativo. A imagem a desenhar no ecrã será então escolhida de acordo com o valor deste atributo booleano.

staticImage

O componente *staticImage* trata-se apenas de uma imagem estática que é desenhada no ecrã, pelo que os únicos atributos contidos na classe *staticImage* são o atributo de localização da imagem que se pretende desenhar no ecrã e os atributos de posição da imagem no *display*. Uma vez obtidos estes valores, basta desenhar a imagem no ecrã.

label

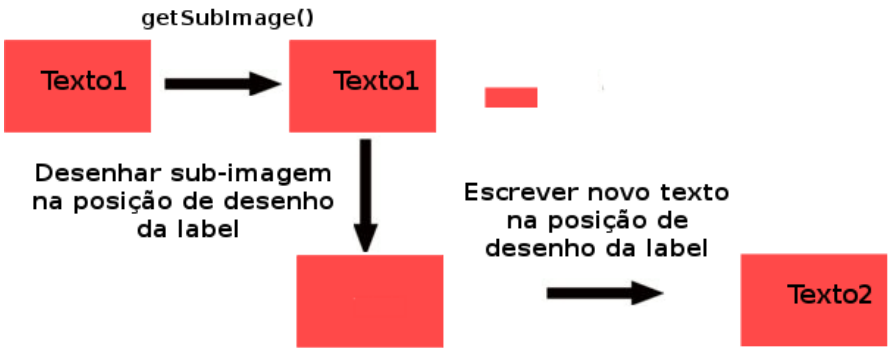

Figura 4.9: Procedimento de atualização do componente *label*

O componente *label* define-se como um pedaço de texto que poderá ser visualizado no ecrã. A classe que está associada ao componente *label*, a classe *Label*, possui os atributos de posição da *label* no painel de instrumentos, o tamanho e o tipo da fonte utilizada para escrever a *label* no ecrã, a cor do texto e um valor (alfanumérico) que será o texto apresentado no ecrã. A implementação do componente possui duas vertentes que estão dependentes do atributo *refreshable* associado a este componente no documento XML de definição de componentes do painel de instrumentos: se o valor de *refreshable* for *False*, o componente tratar-se-á apenas uma porção de texto estático que não necessitará, portanto, ser atualizado por cada iteração do método *paint()*. Assim, basta invocar o método *drawString()* da classe *Graphics* (presente na

API do *Java*) de modo a colocar a *label* no ecrã na posição pretendida; se o valor de *refreshable* for *True*, significa que o componente sofrerá alterações durante as várias iterações do método *paint()*. De modo a tornar mais eficiente a atualização da escrita da *label* no ecrã (ou seja, de modo a não ser necessária a atualização de todo o ecrã quando o texto da *label* for alterado), é utilizado o método *getSubimage()* da classe *Graphics* e os métodos *getHeight()* e *getWidth()* da classe *FontMetrics*(presente na API do *Java*). Os métodos *getHeight()* e *getWidth()* devolvem, respetivamente, a altura e o comprimento que a escrita desta porção de texto vai ocupar no *display*, dado o tipo e o tamanho da fonte escolhida. De seguida, utiliza-se o método descrito na figura 4.9, no qual é invocado o método *getSubimage()* de modo a obter uma sub-imagem que será constituída pelos componentes que estão dentro do espaço ocupado pelo componente *label* no ecrã. Esta sub-imagem não incluirá a própria *label* (i.e., a sub-imagem será constituída pelos elementos "de fundo" que estão nos limites da *label*). Esta sub-imagem é desenhada na posição de desenho da *label*, removendo desta forma o texto que anteriormente se podia ler no ecrã, e, finalmente, é invocado o método *drawString()* de modo a colocar a *label* no ecrã na posição pretendida.

button

O componente *button* é um componente que apresenta o comportamento que um normal botão apresentaria em outras aplicações, ou seja, exibe uma imagem enquanto não é pressionado e exibe uma imagem diferente enquanto está a ser pressionado. Assim, a classe *Button* possui os atributos de localização das duas imagens que serão utilizadas no desenho dos dois diferentes estados do componente e os atributos de posição do componente no ecrã. Uma vez obtidos estes valores, será desenhada no ecrã a imagem relativa ao *button* quando este não está a ser pressionado e, se existir por parte do utilizador um toque dentro dos limites da imagem, será desenhada, na mesma posição, a imagem relativa ao *button* quando este está a ser pressionado. Será também necessária a definição de um método que será invocado cada vez que o botão for pressionado de forma a implementar o comportamento que se pretende dar à ação de pressionar o botão.

4.5.6 Implementação na plataforma *Android*

O *Google Android* é um sistema operativo que foi desenhado para ser utilizado em dispositivos móveis, como telemóveis ou *tablets* [Google, 2011c]. O painel de instrumentos definido nesta dissertação tem por objetivo ser executado num *tablet* presente num veículo cuja interação com o utilizador é realizada através de um ecrã *touch-screen*. Assim, o sistema de definição do painel de instrumentos de um veículo foi também implementado para a plataforma *Android* de forma a aproveitar as facilidades oferecidas por esta plataforma para o desenvolvimento de aplicações móveis que interagem com o utilizador através de um ecrã *touch-screen* [Google, 2011b].

No entanto, apesar do desenvolvimento de aplicações em *Android* ser realizado na linguagem de programação *Java* [DiMarzio, 2008], existem algumas diferenças entre utilização de *Java* para o desenvolvimento de aplicações em *Android* e a utilização de *Java* para o desenvolvimento de aplicações em *Desktop*. Assim, nesta secção serão apresentados alguns detalhes relativos à implementação do sistema de definição do painel de instrumento de um veículo na plataforma *Android*, sendo também apresentadas as principais diferenças entre a implementação deste sistema para *Android* e para *Desktop*.

Visão geral do desenvolvimento de uma aplicação para *Android*

O desenvolvimento de aplicações em *Android* é realizado, geralmente, efetuando a separação entre o código e a aparência (ou *layout*) da aplicação [Murphy, 2010]. Esta separação é realizada através da definição de *layouts* em documentos XML nos quais se definem quais os *containers* e componentes que estarão presentes na aplicação e as relações entre estes elementos. A separação entre o *layout* e o código da aplicação permite que durante a fase de desenvolvimento se altere facilmente a aparência da mesma sem alterar o seu comportamento.

Um dos componentes fundamentais de uma aplicação desenvolvida em *Android* são as **Atividades** [Murphy, 2010]. Uma **Atividade** é um componente que providencia uma janela com a qual os utilizadores de uma aplicação podem interagir com o diverso conjunto de componentes existentes em *Android* (como botões, listas, imagens, menus, etc.). Uma aplicação é geralmente constituída por um conjunto de **Atividades** que formam a interface gráfica entre o sistema *Android* e um utilizador, sendo que cada uma das **Atividades** que compõem uma aplicação são executadas em processos distintos. A implementação de uma **Atividade** é composta por um conjunto de métodos que são invocados durante o ciclo de vida da mesma [DiMarzio, 2008]. Estes métodos, que são os responsáveis pelas ações que a aplicação executa em determinados momentos do ciclo de vida de uma **Atividade**, devem ser implementados pelo programador de modo a controlar os momentos nos quais determinadas ações de uma aplicação são executadas. Os métodos a implementar são:

1. ***onCreate*** - método no qual se define o comportamento da aplicação quando a *Atividade* é criada
2. ***onStart*** - método no qual se define o comportamento da aplicação quando a *Atividade* é colocada à disposição visual do utilizador
3. ***onStop*** - método no qual se define o comportamento da aplicação quando a *Atividade* está parada
4. ***onDestroy*** - método no qual se define o comportamento da aplicação quando a *Atividade* é destruída

O *Android* possui na sua API mecanismos que permitem ao programador o desenho de gráficos numa Atividade [Google, 2011a]. Um destes mecanismos é a classe *Canvas*, que fornece uma superfície para o desenho de gráficos no ecrã. O desenho de gráficos em *Android* é realizado de forma semelhante ao da plataforma *Java*, pois tal como na plataforma *Java* é necessária realizar o *override* de um método, o método *onDraw()*, que permitirá o desenho de pontos, linhas, imagens, etc. O desenho de elementos no ecrã é, também como na plataforma *Java*, realizado de cima para baixo no ecrã, sendo que a posição inicial de desenho do ecrã (i.e., a posição **(0,0)**) se encontra no canto superior esquerdo do mesmo. No entanto, considera-se que existe uma diferença crucial entre o desenho de gráficos entre a plataforma *Android* e a plataforma *Java*: não é possível desenhar parcialmente os elementos presentes no ecrã sem ter que redesenhar todo o ecrã por cada iteração do método *onDraw()*. O desenho parcial dos elementos no ecrã pode ser efetuado na plataforma *Java* através da realização de um *override* ao método *paint()* no qual apenas são redesenhados os componentes que se que pretende atualizar. No ciclo normal de desenho da plataforma *Java*, por cada iteração de *paint()* o ecrã é redesenhado na sua totalidade. Ao efetuar o *override* do método *paint()*, o ciclo normal de desenho do *Java* é ignorado, o que significa que por cada iteração do *paint()* o ecrã só é redesenhado totalmente se for dada essa indicação pelo programador. Isto significa que os elementos que são desenhados no ecrã só "desaparecerão" se forem desenhados novos elementos por cima destes, sendo assim possível desenhar um elemento apenas uma vez se este não necessitar de ser alterado durante todo o ciclo de execução do método *paint()*. Na plataforma *Android*, no entanto, não é possível implementar mecanismos que procurem aumentar a eficiência do ciclo de desenho através do redesenho parcial de componentes no ecrã. O único método de desenho para o qual se pode realizar um *override* é o método *onDraw()* que, por cada iteração, faz o redesenho total do ecrã. Assim, terá que se assumir que a atualização dos gráficos presentes no ecrã na plataforma *Android* implica o redesenho de todos os componentes no ecrã.

Portabilidade do código

A implementação do sistema de definição do painel de instrumentos de um veículo foi inicialmente realizada na plataforma *Java*, podendo por isso utilizar todas as classes da API do *Java*. A plataforma *Android* utiliza diversas classes da API nativa do *Java*, como é o caso da classe correspondente ao mecanismo de parsing XML *SAX*. Porém, existem diferenças entre as plataformas *Android* e *Java* na implementação de algumas ações, como é o caso do desenho de imagens no ecrã. Estas diferenças não devem, no entanto, dificultar demasiado a tarefa de aproveitar o código escrito para plataforma *Java* na plataforma *Android*. Assim, de forma a facilitar a portabilidade do código entre as diferentes plataformas, foram identificadas algumas ações indispensáveis ao desenvolvimento do sistema de definição do painel de instrumentos de um veículo e que são realizadas de forma diferente entre as plataformas

Java e *Android*. Uma vez identificadas as ações que são realizadas de forma distinta entre as duas plataformas, foram criadas algumas **Interfaces Java** que obrigam a definição de um conjunto de métodos para facilitar a transição de código entre plataformas distintas. A implementação destes métodos procura abstrair alguns detalhes específicos à realização destas ações em cada uma das plataformas de forma a que, durante o desenvolvimento do sistema, seja apenas necessário alterar o código relativo aos métodos presentes nestas **Interfaces Java**.

As **Interfaces Java** definidas foram as interfaces *ImageHandling* e *XMLParsing* que são responsáveis por, respetivamente, abstrair as ações relativas à leitura e escrita de elementos gráficos no ecrã e abstrair a ação de *parsing* do documento XML de definição de componentes do painel de instrumentos de um veículo. A interface *ImageHandling* possui dois valores, **T1** e **T2**, de tipos parametrizados (ou seja, valores de tipos genéricos que podem ser instanciados em tipos mais específicos [Martins, 2009]), nos quais **T1** representa o tipo associado a uma imagem e **T2** representa o tipo associado à superfície de desenho num ecrã. A interface *ImageHandling* é composta pelos seguintes métodos:

1. **createCanvasImages()** - método que inicializa todas as imagens que é necessário desenhar no ecrã.
2. **loadImages()** - método que carrega todas as imagens que é necessário desenhar no ecrã.
3. **loadImageFromFile** - método que dado o nome de um ficheiro de imagem, devolve a imagem, de tipo **T1**, que se encontra representada nesse ficheiro.
4. **drawImage()** - método que dada uma imagem de tipo **T1**, uma superfície de desenho de tipo **T2** e uma posição de desenho, **x** e **y**, desenha a imagem nessa superfície de desenho na posição **(x,y)**.
5. **drawString()** - método que dada uma *String*, uma superfície de desenho de tipo **T2** e uma posição de desenho, **x** e **y**, escreve a *String* nessa superfície de desenho na posição **(x,y)**.
6. **drawLine()** - método que dada uma superfície de desenho de tipo **T2**, uma posição inicial de desenho, **x1** e **y1** e uma posição final de desenho, **x2** e **y2**, desenha uma linha nessa superfície de desenho entre as posições **(x1,y1)** e **(x2,y2)**.

Apesar do mecanismo de *parsing* de documentos XML **SAX** existir nas plataformas *Java* e *Android*, foi criada a interface *XMLParsing* de modo a que um programador possa utilizar os mecanismos de *parsing* XML que quiser em qualquer uma destas plataformas. A interface *XMLParsing* possui um tipo parametrizado, **T3**, que representa o tipo da lista de componentes que serão retirados do documento XML. Assim,

a interface *XMLParsing* possui apenas um único método, o método **parseXML()**, que dado o nome de um ficheiro devolve uma lista do tipo **T3** que contém todos os componentes definidos no documento XML de definição de componentes do painel de instrumentos.

Diferenças na implementação em *Android*

A implementação em *Android* do sistema de definição de componentes do painel de instrumentos segue a mesma estrutura que a implementação realizada para a plataforma *Java* (consultar seções 4.5.2, 4.5.3, 4.5.4 e 4.5.5). Assim, a implementação deste sistema seguirá os seguintes passos:

1. É realizado o *parsing* do documento XML de definição de componentes do painel de instrumentos e, como resultado deste *parsing*, é obtida a lista de componentes do sistema;
2. Os componentes do painel são separados em duas listas: *refreshable* e *nonRefreshable*;
3. É utilizado o método *onDraw()* para desenhar os componentes no ecrã;
4. Por cada iteração de *onDraw()*, são atualizados os componentes *refreshable*.

Como o mecanismo que foi utilizado para o *parsing* do documento XML de definição de componentes do painel de instrumentos foi o mesmo para as plataformas *Java* e *Android*, a implementação do passo 1 é realizada da mesma forma para ambas as plataformas. Denote-se que foi implementado o método *parseXML()* da interface *XMLParsing* de modo a obter a lista de componentes do painel, sendo que no caso do passo 1 a implementação do método *parseXML()* é igual para as plataformas *Android* e *Java*.

A separação dos componentes obtidos do documento XML em duas listas (como referido no passo 2) tinha dois objetivos na implementação do sistema para a plataforma *Java*: tornar mais eficiente a atualização dos dados dos componentes *refreshable* e tornar mais eficiente o redesenho dos componentes *refreshable*. O desenho dos componentes do painel era feito através da realização de um *override* ao método *paint()* no qual era especificado que os componentes não *refreshable* apenas eram desenhados uma vez, enquanto que os componentes *refreshable* eram redesenhados por cada iteração do método *paint()* (consultar secção 4.5.4). No entanto, a separação dos componentes do painel em duas listas na implementação do sistema para *Android* apenas possuirá um objetivo: tornar mais eficiente a atualização dos dados dos componentes *refreshable*. Em *Android*, tal como foi referido em 4.5.6, não é possível desenhar um ou mais elementos no ecrã sem que isso implique o redesenho de todos os elementos no ecrã. Não sendo permitido o redesenho parcial dos componentes

em *Android*, não será possível, por cada iteração do método *onDraw()*, redesenhar apenas os componentes *refreshable* de modo a tornar mais eficiente o redesenho dos componentes no ecrã. Assim, por cada iteração de *paint()*, serão desenhados no ecrã todos os componentes presentes na lista de componentes do painel obtida a partir do documento XML de definição de componentes do painel de instrumentos de um veículo.

Os métodos de desenho gráfico no ecrã são diferentes entre as plataformas *Android* e *Java*. No entanto, ao utilizar a interface *ImageHandling*, o código utilizado nas duas plataformas para desenhar elementos no ecrã é semelhante, variando apenas na implementação dos métodos desta interface. Assim, para a plataforma *Android*, a implementação dos métodos de desenho de imagens, *strings* e linhas não passará da respetiva invocação dos métodos correspondentes da classe *Canvas*, ou seja:

1. Na implementação do método **drawImage()**, será invocado o método **drawBitmap()** da classe *Canvas*;
2. Na implementação do método **drawString()**, será invocado o método **drawText()** da classe *Canvas*;
3. Na implementação do método **drawLine()**, será invocado o método **drawLine()** da classe *Canvas*;

4.6 Utilização do ambiente de desenvolvimento

Tal como referido em 4.2, um dos objetivos definidos durante a definição do ambiente de desenvolvimento de um painel de instrumentos para um veículo é o de garantir a interdisciplinaridade entre os indivíduos responsáveis pela definição do painel de instrumentos. Assim, a definição do ambiente de desenvolvimento efetua a separação entre a definição da aparência do painel de instrumentos e a definição do comportamento dos componentes que fazem parte do painel de instrumentos.

Considerou-se que no grupo de desenvolvimento de um painel de instrumentos existem dois tipos de indivíduos: os ***designers***, responsáveis pela definição da aparência do painel, e os **programadores**, responsáveis pela definição do comportamento dos componentes no painel de instrumentos. Quando um indivíduo pertencente a um destes grupos de trabalho pretende definir um painel de instrumentos através do ambiente de desenvolvimento apresentado nesta dissertação, o mesmo não terá que se preocupar com os pormenores relativos às tarefas realizadas pelos indivíduos do grupo de trabalho oposto.

Através da utilização do ambiente de desenvolvimento apresentado nesta dissertação, um ***designer*** terá apenas de se preocupar com o desenho dos componentes no *display*. De modo a definir a apresentação dos componentes no *display*, o ***designer***

definirá um documento XML de definição de componentes no painel de instrumentos. Neste documento XML, o *designer* terá que incluir uma *tag* identificadora para cada um dos componentes que pretende apresentar no ecrã e, além disso, terá que definir para cada *tag* identificadora de componentes um conjunto de propriedades associadas ao componente respetivo.

Através da utilização do ambiente de desenvolvimento apresentado nesta dissertação, um **programador** terá apenas de se preocupar com o comportamento dos componentes no *display*. A definição do comportamento dos componentes no *display* por parte do **programador** será feita através de três tarefas principais: a leitura do documento XML de definição de componentes no painel de instrumentos, na qual são retirados os componentes que se pretende apresentar no *display* e as respetivas propriedades; a apresentação dos componentes no *display*, na qual é definido o método de desenho de cada componente no *display*; a definição das ações que cada componente realiza durante o ciclo de execução do painel de instrumentos.

Imagine-se, a título de exemplo, uma situação na qual existe um grupo de trabalho, composto por um ***designer*** e um **programador**, cujo objetivo é o de definir um painel de instrumentos para um determinado veículo. Dentro do conjunto de especificações definidas para este painel de instrumentos encontra-se a especificação de que o painel terá de possuir um botão que, ao ser pressionado pelo condutor, inicia o sistema de navegação GPS do veículo. O ***designer*** toma as seguintes decisões:

1. O botão deve ser apresentado na posição *(100,150)* do *display*
2. Quando o botão não está a ser pressionado pelo condutor, deve ser apresentada a imagem *botaoInativo.png*
3. Quando o botão está a ser pressionado pelo condutor, deve ser apresentada a imagem *botaoAtivo.png*

O **programador**, por sua vez, toma a decisão de que quando o botão é pressionado, deve ser verificado se o sistema de navegação GPS se encontra disponível e:

1. Se o sistema de navegação GPS se encontrar disponível, o mesmo será apresentado no *display*
2. Se o sistema de navegação GPS não se encontrar disponível, o mesmo não será apresentado no *display* e será apresentada uma janela a avisar o condutor acerca da indisponibilidade do sistema de navegação GPS

Através da utilização do ambiente de desenvolvimento de painéis de instrumentos para veículos apresentado nesta dissertação, o ***designer*** irá definir um documento XML de definição de componentes do painel de instrumentos no qual incluiria cada um dos componentes que pretende apresentar no *display*. No caso do botão descrito

no exemplo, o ***designer*** poderia incluir neste documento XML um componente do tipo *button*, da seguinte forma:

<button>
<id> BotaoGPS **</id>**
<x> 100 **</x>**
<y> 150 **</y>**
<imageDefault> botaoAtivo.png **</imageDefault>**
<imagePressed> botaoInativo.png **</imagePressed>**
</button>

Através da utilização do ambiente de desenvolvimento de painéis de instrumentos para veículos apresentado nesta dissertação, o **programador** seguirá os seguintes passos:

1. Efetua a leitura do documento XML de definição de componentes do painel de instrumentos, na qual obtém todos os componentes e respetivas propriedades definidos pelo ***designer***

2. Depois de obtidos todos os componentes definidos no documento XML, o **programador** procederá à apresentação dos mesmos no *display*. No caso do botão apresentado no exemplo, o **programador** definirá que o componente *button* será desenhado na posição *(100,50)*

3. De forma a concluir o seu trabalho, o **programador** definirá as ações que cada componente executará durante o ciclo de execução do painel de instrumentos. No caso do botão apresentado no exemplo, o **programador** poderia começar por verificar a disponibilidade do sistema de navegação GPS do veículo. Se o sistema de navegação GPS se encontrar indisponível, o **programador** deverá definir uma janela que será apresentada ao condutor e que possuirá um aviso a indicar a indisponibilidade do sistema de navegação GPS. Se o sistema de navegação GPS se encontrar disponível, o **programador** deverá disponibilizar o mesmo ao condutor

Capítulo 5

Aplicação para Redes Sociais

Este capítulo refere-se à utilização de redes sociais em veículos automóveis. Importa então perceber de que forma as ligações estabelecidas pelos utilizadores de uma ou mais redes sociais podem dar, ao condutor, um contributo útil durante a condução de um veículo. Será discutida a relevância que uma rede social previamente desenvolvida e populada pode ter no decorrer de uma viagem de um veículo que se conecte a uma ou mais redes sociais. É também importante referir que o estudo desenvolvido se limita ao estudo da integração de redes sociais em veículos, e não à utilização de redes sociais para veículos (sem integração da rede social no mesmo). A utilização (sem integração) de redes sociais para veículos é aquela na qual a rede social pode ou não ser acedida, direta ou indiretamente, a partir do veículo e que não fornece informações que auxiliem a realização de alguma tarefa durante o ato da condução (por exemplo, uma rede social na qual os seus utilizadores trocam informações acerca de carros usados). Uma rede social integrada num veículo é aquela na qual a rede social é acedida, direta ou indiretamente, a partir do veículo (podendo também conter serviços específicos que são acedidos fora do veículo) e que fornece informação social que auxilie na realização de uma ou mais tarefas durante o ato da condução, sendo precisamente acerca desta última forma de utilização de redes sociais que este capítulo da dissertação se refere (por exemplo, uma rede social disponível em veículos que efetue trocas de mensagens entre os veículos).

5.1 A aplicação VeículoSocial

5.1.1 Enquadramento e serviços disponibilizados pela aplicação

Podemos definir o termo rede social como "um conjunto de serviços *web-based* que permitem aos seus utilizadores (1) a construção de perfis públicos ou semi-públicos dentro de um sistema limitado, (2) a articulação de uma lista de outros utilizadores com quem partilham uma ligação, e (3) o acesso à lista de ligações de um utilizador, bem como às listas de ligações de outros utilizadores do sistema" [Ellison and Ellison, 2007]. Assistiu-se nos últimos anos a um enorme crescimento da utilização de redes sociais na Internet. As diversas redes sociais disponíveis (*Facebook*, *Twitter*, *MySpace*, etc.) são, hoje em dia, ferramentas do uso comum de várias pessoas e empresas de todo o mundo, assumindo-se por isso como as principais ferramentas de ligação e interação social à escala global [Report, 2011]. A plataforma *Facebook*, em particular, é a plataforma de rede social mais utilizada na Internet, permitindo um número de ligações e partilha de conteúdos a um maior número de pessoas que qualquer outra rede social [Stats, 2011].

Uma rede social pode ser vista como uma ferramenta que contém um conjunto de ligações sociais entre os seus utilizadores. Com base neste facto, considerou-se que a ligação de um veículo automóvel a uma rede social pode ser aproveitada de forma a auxiliar o condutor de um automóvel na realização de uma ou mais tarefas enquanto conduz o seu veículo. Assim, foi desenvolvida uma aplicação para um veículo automóvel, a aplicação **VeículoSocial**, que utiliza a ligação à rede social *Facebook* de forma a utilizar as ligações sociais entre os seus utilizadores como um **filtro de conteúdos de localização**. A aplicação utiliza um mapa que estará disponível no *display* do veículo e disponibiliza ao utilizador dois serviços: o serviço **Pontos de Interesse** e o serviço **Procura de amigos**.

O serviço **Pontos de Interesse** disponibilizará um conjunto de pontos geográficos num mapa, situados numa localização próxima da localização atual do condutor, que estarão associados a pontos de interesse introduzidos pelo condutor ou por outros utilizadores da aplicação. Os pontos geográficos disponibilizados incluem-se numa das seguintes categorias:

1. **Restauração**, onde se encontram todos os pontos relativos a estabelecimentos de restauração (restaurantes, cafetarias, etc.)

2. **Alojamento**, onde se encontram todos os pontos relativos a estabelecimentos que proporcionam alojamento (hotéis, pousadas, etc.)

3. **Postos de abastecimento**, onde se encontram todos os pontos relativos a postos de abastecimento de combustível (postos de abastecimento de gasolina ou gasóleo, postos de abastecimento eléctrico, etc.)

4. **Lazer**, onde se encontram todos os pontos relativos a áreas de lazer (cinemas, teatros, discotecas, etc.)

5. **Espaços Comerciais**, onde se encontram todos os pontos relativos a estabelecimentos destinados ao comércio (centros comerciais, supermercados, etc.)

6. **Parques de Estacionamento**, onde se encontram todos os pontos relativos a parques de estacionamento

7. **Outros**, onde se encontram todos os pontos que não se incluem numa das categorias anteriores

O serviço **Procura de amigos** disponibilizará, tal como o serviço anterior, um conjunto de pontos geográficos num mapa, situados numa localização próxima da localização atual do condutor, sendo que estes pontos estarão associados à posição atual dos veículos pertencentes a utilizadores que sejam amigos do condutor na rede social *Facebook* e que disponibilizaram a sua localização de forma a serem encontrados pelos indivíduos que lhes são socialmente relevantes.

A rede social utilizada na aplicação **VeículoSocial** é o *Facebook* pois é a rede social mais utilizada e, portanto, existe uma maior probabilidade de o condutor possuir uma rede de amigos já estabelecida nesta plataforma. A aplicação utilizará as ligações previamente estabelecidas pelos indivíduos no *Facebook* e aproveitar-se-á dessas mesmas ligações na filtragem de conteúdos. Esta filtragem de conteúdos resultará de uma interpretação que o sistema realiza da informação social que recebe do *Facebook*: para o serviço **Pontos de Interesse**, é dada ao utilizador a possibilidade de escolher entre a visualização, no mapa, dos pontos marcados por todos os utilizadores ou entre os pontos marcados por utilizadores do sistema que são, simultaneamente, amigos do utilizador no *Facebook*. A filtragem de conteúdos é adicionada ao sistema de forma a prevenir que os seus utilizadores utilizem informação de localização que não lhes é relevante, pois se o sistema não permitisse uma filtragem de conteúdos através das ligações sociais do utilizador, qualquer entidade poderia adicionar pontos geográficos no mapa com um intuito contrário ao de fornecer informações de maior utilidade possível ao condutor. Por exemplo, imagine-se uma situação na qual o condutor pretende obter os estabelecimentos de lazer que se encontram nas proximidades da posição atual do seu veículo: ao realizar a filtragem de conteúdos pela experiência dos seus amigos no *Facebook*, existe uma maior probabilidade de o condutor ficar satisfeito com os estabelecimentos que encontrou. O filtro de conteúdos de localização confere ao utilizador, portanto, um significado socialmente relevante à função de visualização de "pontos de interesse" disponível na aplicação **VeículoSocial**. No serviço **Procura de Amigos** a informação social retirada do *Facebook*, ao funcionar novamente como uma filtragem de conteúdos de localização, apresenta-se fundamental para os utilizadores da aplicação, pois não

interessará a um utilizador que, ao utilizar o serviço **Procura de Amigos**, tenha a possibilidade de ser localizado por qualquer pessoa registada na aplicação **VeículoSocial**, mas sim somente por aqueles que mantêm uma ligação social com o utilizador que disponibiliza a sua localização actual.

5.1.2 Arquitetura da aplicação

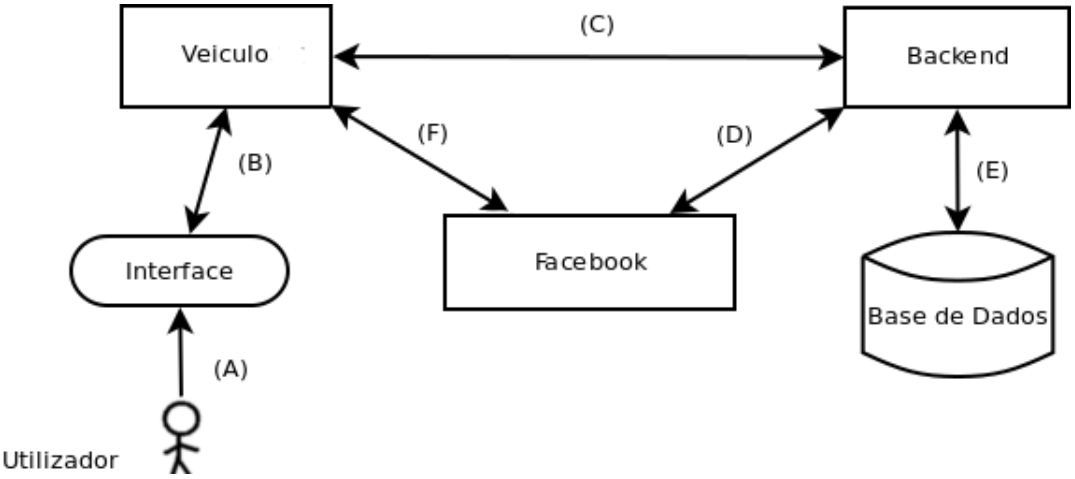

Figura 5.1: Arquitetura da aplicação **VeículoSocial**

A arquitetura definida para o desenvolvimento da aplicação **VeículoSocial** pode ser consultada na figura 5.1. Os elementos constituintes da aplicação são: a Interface, o Veículo, o *Backend*, uma base de dados (pertencente ao *Backend*) e a rede social *Facebook*.

De forma a requisitar um serviço à aplicação, o utilizador acede **(A)** à sua interface, que por sua vez irá requisitar, à aplicação instalada no veículo, as funcionalidades desejadas pelo utilizador **(B)**.

As funcionalidades requeridas pelo utilizador necessitam de dados do *Backend* e do *Facebook* e, por isso, o Veículo necessita de realizar pedidos ao *Backend*, que não é mais que uma aplicação que está instalada num servidor e que efetua ligações a uma base de dados que também se encontra nesse servidor. A autenticação do utilizador, durante o acesso do Veículo ao *Backend*, deverá ser feita com o nome de *login* que o utilizador usa para aceder ao *Facebook* de forma a facilitar o cruzamento de dados no *Backend*. A *password* de autenticação do utilizador no sistema não deve, no entanto, ser a mesma que este utiliza para se autenticar no *Facebook*, por questões de privacidade.

Assim, é necessária uma ligação entre o Veículo e o *Facebook* **(F)**, ligação essa que retornará ao Veículo uma *token* temporária de acesso ao *Facebook* que será passada posteriormente ao *Backend*.

O utilizador poderá necessitar de dados relativos a todos os utilizadores da aplicação acerca de pontos geográficos armazenados no sistema, pelo que neste caso o Veículo apenas necessitá de realizar o pedido ao *Backend* **(C)** que, por sua vez, irá consultar a base de dados **(E)** e devolverá os dados de todos os utilizadores do sistema ao Veículo **(C)**.

Por outro lado, o utilizador poderá necessitar de dados de pontos geográficos relativos a utilizadores do sistema que são seus amigos no *Facebook*, sendo que neste caso o Veículo efetuará esse pedido ao *Backend* **(C)**.

O *Backend*, por sua vez, pedirá os dados relevantes à base de dados **(E)**, dados esses que serão cruzados com a lista de amigos do utilizador no *Facebook*, que será obtida através de uma ligação do *Backend* com o *Facebook* **(D)**.

Uma vez obtidos e cruzados os dados necessários, o *Backend* devolve ao Veículo **(C)** os dados relativos a pontos assinalados pelos amigos do utilizador no *Facebook*.

Assim que o Veículo obtém os pontos geográficos necessários que tem de assinalar no mapa, estes passarão a estar disponíveis à Interface **(B)**, que permitirá a visualização dos mesmos ao utilizador **(A)**.

5.1.3 Tecnologias envolvidas no desenvolvimento da aplicação

A aplicação **VeículoSocial** é uma aplicação que utiliza o modelo cliente-servidor de forma a estabelecer uma comunicação entre um veículo (o cliente) e um servidor. O modelo cliente-servidor é um modelo no qual existem duas aplicações que são executadas em máquinas diferentes: a aplicação servidor e a aplicação cliente. Neste modelo, uma aplicação servidor espera que uma aplicação cliente estabeleça uma comunicação com a aplicação servidor que, por sua vez, irá responder apropriadamente à aplicação cliente [Calvert and Donahoo, 2008]. A aplicação cliente irá apresentar um mapa e alguns botões de forma a interagir com o condutor do veículo. Esta interação entre a aplicação e o condutor será realizada através de um *display touch-screen* e, por isso, foi desenvolvida na plataforma *Android*. A aplicação servidor, que foi desenvolvida na linguagem de programação *Python*, terá de obter dados de uma base de dados, comunicar com a plataforma *Facebook* e comunicar com o veículo. Nesta secção irão ser listadas as tecnologias envolvidas no desenvolvimento da aplicação **VeículoSocial** e será feita uma breve apresentação para cada uma destas tecnologias.

Comunicação por *sockets* TCP

A comunicação efetuada entre o cliente e o servidor da aplicação **VeículoSocial** é uma comunicação que utiliza o protocolo TCP[1]. De modo a estabelecer uma ligação entre a aplicação cliente e a aplicação servidor, ambas as aplicações utilizam um *socket TCP*. Um *socket* é uma abstração através da qual se podem efetuar trocas de dados entre aplicações [Calvert and Donahoo, 2008]. Um *Socket* oferece a uma aplicação a capacidade de se ligar, dentro de uma rede, a outra aplicação e enviar ou receber dados através dessa ligação. Existem vários tipos de *sockets* mas, para o trabalho desenvolvido para esta dissertação, apenas foi utilizado um tipo de *socket*: o *socket* TCP. Um *socket* TCP, que também pode ser denominado de *stream socket*, é um tipo de *socket* que mantém uma ligação constante entre aplicações [Hall, 2009]. Um *socket* TCP utiliza o protocolo TCP/IP[2], que é um protocolo que utiliza o protocolo TCP de modo a garantir a integridade dos dados e utiliza o protocolo IP que, através da utilização de endereços IP, é utilizado para o roteamento de pacotes na rede. Assim, para além de utilizar uma ligação constante entre as aplicações que utilizam *sockets* deste tipo, os *sockets* TCP garantem também que os dados enviados por uma aplicação são recebidos sequencialmente e sem erros por outra aplicação.

JSON

O JSON[3] é um formato para a troca de dados entre aplicações [json.org, 2011]. Trata-se de um formato de texto que é independente de qualquer linguagem de programação, o que o torna ideal para a troca de dados entre aplicações desenvolvidas em diferentes linguagens de programação, como é o caso da troca de dados entre a aplicação cliente e a aplicação servidor do **VeículoSocial**. Uma *string* JSON pode ser composta por dois tipos de elementos: uma coleção de pares chave/valor, denominada de **objeto JSON**; uma lista de valores JSON, denominada de **array JSON**. Um **objeto JSON** é um conjunto de pares chave/valor que não contém uma ordem específica. Um objeto JSON é construído colocando, entre chavetas ({ }), um conjunto de pares chave/valor na forma *chave:valor* e que são separados por vírgulas (,). Um **array JSON** é uma sequência de valores JSON. Um valor JSON pode ser uma *string*, um valor numérico (inteiro, fracionário ou exponencial), um valor booleano (*true* ou *false*), um valor *null* ou um objeto JSON. Um array JSON é construído colocando, entre parênteses retos ([ ]), um conjunto de valores JSON separados por vírgulas (,).

A título de exemplo, considere-se a utilização do JSON na troca de dados entre duas aplicações que pretendem trocar dados acerca das temperaturas mínimas e máximas previstas numa determinada semana nas várias cidades de Portugal. A *string* JSON

[1] TCP - Transmission Control Protocol
[2] TCP/IP - Transmission Control Protocol/Internet Protocol
[3] JavaScript Object Notation

utilizada neste exemplo poderia conter, por exemplo, dois objetos JSON. O primeiro objeto JSON seria constituído pela chave *cidade* e pelo valor (*string*) correspondente ao nome da cidade. O segundo objeto JSON seria composto pela chave *temperaturas* e o respetivo valor seria um array JSON que, por sua vez, seria composto por um conjunto de objetos JSON compostos por três pares chave/valor: a chave *dia*, cujo valor (*string*) corresponde ao dia da semana; a chave *mínima*, cujo valor (numérico) corresponde à temperatura mínima do dia correspondente; a chave *máxima*, cujo valor (numérico) corresponde à temperatura máxima do dia correspondente. Desta forma, a *string* JSON enviada por uma das aplicações poderia ser:

```
{"cidade":"Évora", "temperaturas": [{"dia":"seg", "mínima":6.2,
                                     "máxima":10.8},
                                    {"dia":"ter", "mínima":5,
                                     "máxima":10.2},
                                    ...] }
```

Facebook API

A plataforma *Facebook* possui uma API que permite a comunicação entre a plataforma *Facebook* e uma aplicação. Com a *Facebook* API, uma aplicação pode ler e escrever dados na plataforma *Facebook*, aproveitando assim as ligações sociais estabelecidas pelos diversos utilizadores da plataforma.

De forma a aceder aos elementos disponíveis nos perfis dos seus vários utilizadores, um utilizador que se ligue ao *Facebook* necessita de se autenticar (ou seja, provar a sua identidade) nessa plataforma [facebook.com, 2011b]. A autenticação *Facebook* utiliza o protocolo OAUTH2 [facebook.com, 2011b], que é um protocolo padrão livre que utiliza a troca de *tokens* de acesso entre aplicações de forma a possibilitar a autenticação de utilizadores numa plataforma [Group, 2011a]. A implementação do protocolo OAUTH2 realizado no *Facebook* possui três vertentes: autenticação de utilizadores, autorização de aplicações e autenticação de aplicações [facebook.com, 2011b]. No entanto, apenas a autenticação de utilizadores foi utilizada no trabalho desenvolvido para esta dissertação, pelo que, de entre as três vertentes de autenticação na plataforma *Facebook*, apenas será apresentada a autenticação de utilizadores. Para um utilizador se autenticar no *Facebook* terá obrigatoriamente de ser invocada uma *OAuth Dialog*, que é uma janela que será aberta num *browser* de internet que permite que um utilizador forneça os seus dados de acesso à plataforma *Facebook* (o *login* e a *password*) e, em resposta a esta ação, será devolvida uma *token* de acesso temporário para este utilizador aceder ao *Facebook* [facebook.com, 2011a]. A *token* de acesso devolvida pelo *Facebook* poderá, posteriormente e por um determinado período de tempo, ser utilizada para realizar pedidos à *Graph* API do *Facebook*.

A *Graph* API do *Facebook* é uma API incluída na *Facebook* API que permite a

comunicação com o **grafo social** do *Facebook* [facebook.com, 2011c]. O **grafo social** do *Facebook* é constituído por um conjunto de objetos (utilizadores, imagens, eventos, páginas, etc.) e pelas ligações entre esses objetos. Cada um dos objetos deste grafo contém um número de identificação único que poderá ser utilizado para obter as propriedades do objeto através da realização de um **pedido HTTP**, sendo que a resposta a este pedidos vem na forma de uma *string* JSON. No entanto, de forma a aceder às propriedades dos vários objetos existentes no grafo social do *Facebook* é necessária a inclusão de um *token* de acesso válido no pedido HTTP. Assim, a definição de um pedido HTTP que se liga à plataforma *Facebook* e pede, para um determinado objeto com um número de identificação *ID*, a propriedade *PROP* utilizando o *token* de acesso *TOKEN*, será realizada na forma:

```
https://graph.facebook.com/ID/PROP?access_token=TOKEN
```

Google Maps em *Android*

A aplicação cliente do **VeículoSocial** utiliza um mapa no qual são colocados os pontos geográficos relevantes para os serviços **Pontos de Interesse** e **Procura de Amigos**. De forma a disponibilizar este mapa ao condutor do veículo foi utilizada a biblioteca *Google Maps* para *Android*.

O *Google Maps* é um serviço da *Google* que permite a visualização de mapas, permitindo também a consulta de informação acerca de pontos situados no mapa, a marcação de pontos no mapa e o pedido de direções entre pontos situados no mapa [google.com, 2011]. A aplicação cliente do **VeículoSocial** é implementada em *Android* e, por isso, foram utilizadas as bibliotecas *Google Maps* para *Android* [android.com, 2011]. Estas bibliotecas permitem a utilização, em *Android*, do componente *MapView*, que é um componente que vai permitir a visualização de um mapa *GoogleMaps*. De forma a colocar no mapa os pontos relativos aos serviços **Pontos de Interesse** e **Procura de Amigos**, são utilizados objetos *ItemizedOverlay*, que são objetos nos quais se podem especificar um conjunto de itens a adicionar num componente *MapView*. Os itens colocados num objeto *ItemizedOverlay* pertencem todos a um mesmo tipo, ou seja, cada item possui uma localização geográfica diferente (de modo a que possam ser adicionados na posição adequada na *MapView*), e todos os itens possuem a mesma imagem, que os identificará dos restantes tipos de itens. Assim, para adicionar mais do que um tipo de itens numa *MapView* terão de ser criados mais objetos *ItemizedOverlay*.

PostgreSQL

De forma a implementar a base de dados requerida pela aplicação servidor do **VeículoSocial**, foi utilizado o SGBD[4] **PostgreSQL**. O PostgreSQL é um SGBD relacional de código aberto que utiliza a linguagem SQL[5] para a gestão e consulta de valores em bases de dados [Group, 2011b]. A base de dados contida no servidor do **VeículoSocial** e as respetivas tabelas foram criadas em PostgreSql e, como a aplicação servidor foi desenvolvida em *Python*, foi utilizado o módulo **Psycopg**. O módulo Psycopg é um módulo *Python* que permite a comunicação de uma aplicação com o SGBD PostgreSQL [Psycopg, 2011b].

O módulo Psycopg foi escolhido devido à facilidade que apresenta para efetuar uma ligação a uma base de dados e a facilidade com que permite a execução de expressões SQL sobre essa base de dados [Psycopg, 2011a]. Utilizando o módulo Psycopg, a ligação a uma base de dados de nome *DB*, através do utilizador *USER* com palavra-passe *PASS*, na máquina *HOST* é realizada da forma com recurso ao método *connect()*, na forma:

```
connect("dbname='DB' user='USER' host='HOST' password='PASS'")
```

O método *connect()* efetua uma ligação à base de dados e devolve uma instância de um objeto *connection* que representa a ligação efetuada a essa base de dados. Após estabelecida a ligação à base de dados, pode ser utilizada a instância do objeto *connection* de forma a invocar o método o*cursor()*. O método *cursor()* devolve uma instância de um objeto *cursor* que permitirá a execução de expressões SQL sobre a base de dados através da utilização do método *execute()*. Assim, sendo *lig* uma ligação a uma determinada base de dados obtida pelo método *connect()* e *EXPR* uma expressão SQL, a execução de uma expressão SQL utilizando o módulo Psycopg pode ser feita da forma:

```
cur = lig.cursor()
cur.execute(EXPR)
```

Uma vez executada a expressão SQL, pode ser novamente utilizada a instância do objeto *cursor* de forma a invocar o método *fetchall()* que devolve uma lista com o resultado da avaliação da expressão SQL na base de dados.

5.1.4 Implementação da Aplicação

A implementação do **VeículoSocial** compreende a implementação de duas aplicações: a aplicação cliente e a aplicação servidor. A aplicação cliente foi desenvolvida na

[4]SGBD - Sistema de Gestão de Bases de Dados
[5]Structured Query Language

plataforma *Android*, enquanto que a aplicação servidor foi desenvolvida utilizando a linguagem de programação *Python*. A aplicação cliente será executada num *tablet touch-screen* colocado num veículo e será responsável pela visualização, num mapa, dos pontos geográficos relevantes aos serviços **Pontos de Interesse** e **Procura de Amigos**. A aplicação servidor será executada num servidor remoto e será responsável pela comunicação com a base de dados onde são armazenados os pontos geográficos relevantes aos serviços **Pontos de Interesse** e **Procura de Amigos** e pelo respetivo envio destes pontos à aplicação cliente.

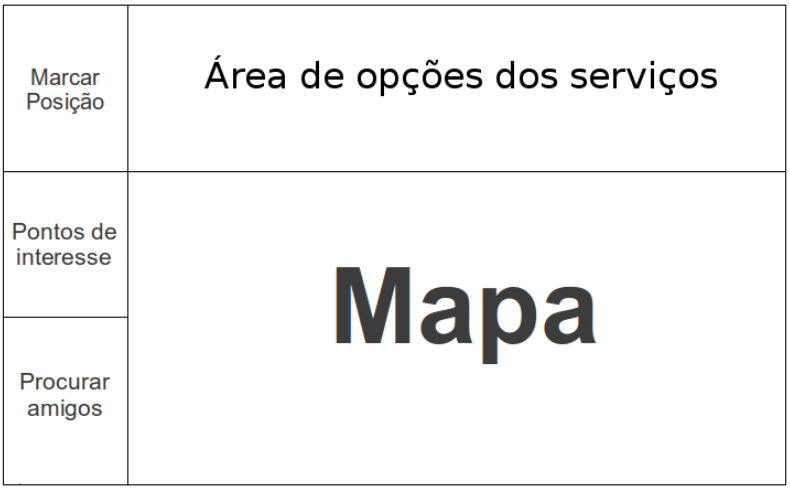

Figura 5.2: Esboço geral da interface da aplicação cliente

Assim, de modo a permitir a visualização de pontos geográficos num mapa e a interação com o *display* de um veículo, foi implementada, em *Android*, a interface da aplicação cliente. Esta interface foi desenhada de forma a incluir um mapa e um conjunto de botões com o quais o utilizador pode alternar entre os serviços disponíveis no **VeículoSocial** (os serviços **Pontos de Interesse** e **Procura de Amigos**) e pode também aceder às opções de cada um dos serviços. O esboço geral da interface definida para a aplicação cliente pode ser consultada na figura 5.2. Como se pode verificar, esta interface é composta por cinco elementos:

1. **Mapa** - é o elemento que ocupa a maior parte da interface de forma a garantir a boa visibilidade do mesmo para o utilizador. Este elemento foi implementado utilizando uma *MapView* que utilizará o serviço *Google Maps* para a visualização e interação com mapas.
2. **Botão "Pontos de Interesse"** - este elemento estará ativo por defeito e indicará que o condutor pretende utilizar o serviço **Pontos de Interesse**. Foi utilizado o componente *Button* do *Android* para a sua implementação na interface.
3. **Botão "Procura de Amigos"** - este elemento estará ativo por defeito e indicará que o condutor pretende utilizar o serviço **Procura de Amigos**.

Foi utilizado o componente *Button* do *Android* para a sua implementação na interface.

4. **Botão "Marcar Posição"** - este elemento tem por objetivo marcar, no mapa, um ponto geográfico relativo a um ponto de interesse do serviço **Pontos de Interesse**. Este elemento apenas estará ativo quando o utilizador escolher o serviço **Pontos de Interesse**, ficando inativo quando o utilizador escolhe o serviço **Procura de Amigos**. Foi utilizado o componente *Button* do *Android* para a sua implementação na interface.

5. **Área de opções dos serviços** - nesta área poderão encontrar-se um conjunto de botões relativos às opções de cada um dos serviços. De forma a alternar entre o conjunto de botões de cada um dos serviços, foi utilizado o componente *ViewFlipper* do *Android*, que é um componente que permite comutar entre diferentes conjuntos de componentes.

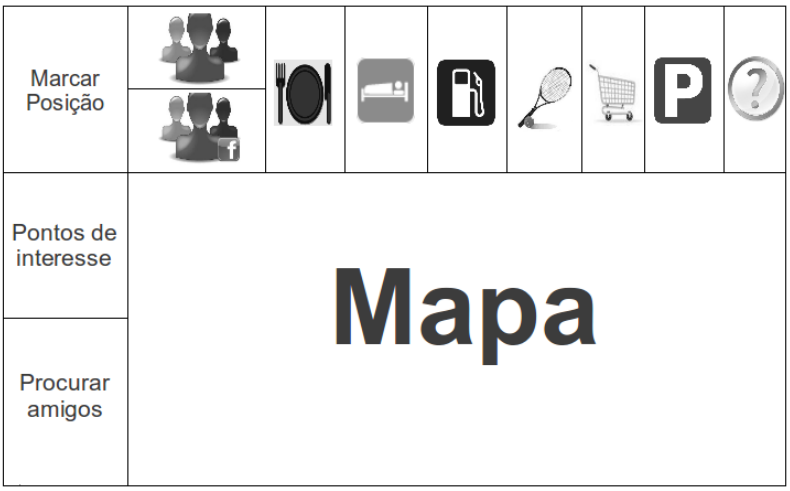

Figura 5.3: Esboço da interface da aplicação cliente para o serviço **Pontos de Interesse**

Como é referido no item 5, a **Área de opções dos serviços** irá alternar entre as opções relativas ao serviço **Pontos de Interesse** e as opções relativas ao serviço **Procura de Amigos**. As opções do serviço **Pontos de Interesse** são o tipo de pontos que se pretende visualizar no **Mapa** (ou seja, de entre as categorias apresentadas em 5.1.1, o condutor pode escolher as categorias de pontos que serão visualizadas no **Mapa**) e a origem dos pontos visualizados no **Mapa** (ou seja, se os pontos apresentados no **Mapa** foram marcados por todos os utilizadores da aplicação ou apenas pelos utilizadores da aplicação que são simultaneamente amigos do condutor no *Facebook*). Como se pode verificar na figura 5.3, todas as opções do serviço **Pontos de Interesse** são representadas por botões que contém imagens de rápida apreensão para o condutor. Os dois botões dispostos verticalmente do lado esquerdo da área

de opções do serviço **Pontos de Interesse** especificam a origem dos pontos apresentados no **Mapa**, podendo o condutor alternar entre a visualização, no **Mapa**, de pontos marcados por todos os utilizadores da aplicação *VeículoSocial*(botão superior) e entre a visualização, no **Mapa**, de pontos marcados por amigos do condutor no *Facebook* que são também utilizadores da aplicação *VeículoSocial*(botão inferior). Os sete botões dispostos horizontalmente na área de opções do serviço **Pontos de Interesse** especificam o tipo de pontos que o condutor pretende visualizar no **Mapa**, sendo que cada um dos botões se inclui, respetivamente, nas categorias Restauração, Alojamento, Postos de abastecimento, Lazer, Espaços Comerciais, Parques de Estacionamento e Outros. O utilizador poderá, assim, escolher a visualização de pontos de uma ou mais categorias. Estes botões serão também utilizados para auxiliar a ação do botão **Marcar Posição**: ao pressionar o botão **Marcar Posição**, a aplicação verificará quais os botões do tipo de pontos da área de opções do serviço **Pontos de Interesse** que estão selecionados de modo a especificar a categoria do ponto que será enviado para o servidor da aplicação. Se apenas um desses botões estiver selecionado, será a categoria do ponto associada a esse botão que será enviada para o servidor. Se estiverem selecionados dois ou mais botões, a aplicação enviará para o servidor a categoria associada ao botão escolhido em último pelo condutor. Todos os botões incluídos na área de opções do serviço **Pontos de Interesse** foram implementados através da utilização do componente *ImageButton* do *Android*.

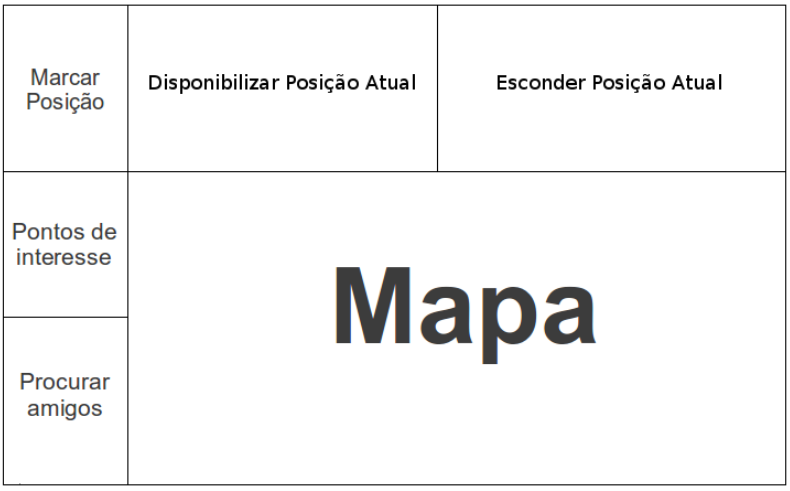

Figura 5.4: Esboço da interface da aplicação cliente para o serviço **Procura de Amigos**

Ao selecionar o **Botão "Procura de Amigos"**, o utilizador pode visualizar no **Mapa** a posição de todos os utilizadores da aplicação *VeículoSocial* que são simultaneamente amigos do condutor no *Facebook*. Ao seleccionar este serviço, as opções da **Área de opções dos serviços** são comutadas para as opções do serviço **Procura de Amigos**: as opções "Disponibilizar Posição Atual" e "Esconder Posição Atual". A primeira opção disponibiliza a posição atual do veículo para todos os utilizadores da aplicação *VeículoSocial* que sejam amigos do condutor no *Facebook*,

enquanto que a segunda opção esconde a posição atual do veículo. Como se pode verificar na figura 5.4, a área de opções dos serviço **Procura de Amigos** contém apenas dois botões, relativos às opções "Disponibilizar Posição Atual" e "Esconder Posição Atual", que foram implementados através da utilização do componente *Button* do *Android*.

Comunicação veículo-servidor

A comunicação efetuada entre a aplicação cliente e a aplicação servidor do **VeículoSocial** foi implementada através da utilização de *sockets* TCP, sendo que as mensagens trocadas pelas aplicações são *strings* JSON.

As mensagens enviadas pela aplicação cliente têm como objetivo pedir à aplicação servidor um conjunto de pontos que serão colocados no **Mapa**. Assim, o pedido efetuado pela aplicação cliente deverá incluir o tipo de serviço pretendido (**Pontos de Interesse** ou **Procura de Amigos**) e deverá incluir, para cada um dos serviços, as opções escolhidas pelo condutor relativas a cada um dos serviços. Ao enviar uma mensagem à aplicação servidor, a aplicação cliente deverá esperar pela resposta, que possuirá o conjunto de pontos que a aplicação cliente irá marcar no **Mapa**.

As mensagens enviadas pela aplicação servidor têm como objetivo responder às solicitações da aplicação cliente. A aplicação servidor deverá, assim, estar constantemente à espera de solicitações da aplicação cliente e, ao receber uma mensagem, deve verificar a *string* recebida e agir de acordo com o serviço solicitado pela aplicação cliente. Finalmente, a aplicação servidor enviará uma mensagem à aplicação cliente com o conjunto de pontos solicitado e continuará à espera de solicitações de pontos relativos aos serviços **Pontos de Interesse** ou **Procura de Amigos**.

Os *sockets* utilizados na aplicação servidor são *sockets* do módulo *ServerSocket* do *Python*. Os *sockets ServerSocket* esperam por solicitações de aplicações clientes (através do método *recv()*) e, por cada pedido recebido, lançam uma nova *Thread* (mecanismo de execução concorrente com o qual uma aplicação pode tratar simultaneamente de várias solicitações de outras aplicações [Calvert and Donahoo, 2008]) onde irão tratar do pedido correspondente, de forma a que o servidor possa tratar paralelamente as solicitações de várias aplicações cliente. Para enviar uma resposta para uma aplicação cliente, é utilizado o método *send()* do módulo *ServerSocket*.

Os *sockets* utilizados na aplicação cliente são instâncias da classe *Socket* do *package java.net* do *Java*. Ao iniciar a aplicação cliente, é criado um *socket* e é devidamente estabelecida a ligação TCP à aplicação servidor. De forma a enviar e receber mensagens da aplicação servidor, são associadas ao *socket*, respetivamente, duas instâncias das classes *DataInputStream* e *DataOutputStream*. Para o envio de mensagens à aplicação servidor, é utilizado o método *writeChars()* da classe *DataOutputStream* que irá enviar uma *String*. Para receber mensagens da aplicação servidor, é utilizado

o método *getInputStream()* da classe *DataInputStream* que esperará por uma mensagem enviada pela aplicação servidor. De forma a que o tempo de espera, efetuado pela aplicação cliente, de uma mensagem da aplicação servidor não afete a restante execução da aplicação, é criada uma *Thread* na aplicação cliente e a invocação ao método *getInputStream()* será efetuada somente nesta *Thread*.

Ao receber uma mensagem, a aplicação servidor utiliza o método *load()* do módulo *json* do *Python* de forma a converter a *String* recebida num objeto JSON. Uma vez convertida a mensagem num objeto *JSON*, basta utilizar a notação *obj[PROP]* de forma a obter o valor respetivo à propriedade *PROP* do objeto JSON *obj*.

Similarmente, quando a aplicação cliente recebe uma mensagem, é utilizada a classe *JSONObject* do *package org.json*. A classe *JSONObject* possui o método *getString()* que dada uma propriedade, *PROP*, devolverá o valor respetivo a essa propriedade no objeto JSON *obj*, utilizando a notação *obj.getString(PROP)*.

Autenticação do condutor no *Facebook*

Quando a aplicação cliente é iniciada, é criada uma instância do objeto *Facebook* (do *package com.facebook.android*). Esta instância do objeto *Facebook* permitirá a invocação do método *authorize()* que abrirá uma *OAUTH Dialog* de forma a que o condutor do veículo se autentique na plataforma *Facebook*. A autenticação do condutor do veículo no *Facebook* requer, por parte do condutor, a introdução do seu *login* e palavra-passe do *Facebook*. Este processo pode ser considerado como um processo bastante perigoso para o utilizador, pois a introdução de caracteres enquanto conduz o seu veículo é uma atividade bastante distrativa do ato da condução. No entanto, após o primeiro processo de autenticação na *OAUTH Dialog*, é dada ao utilizador a opção de guardar o seu *login* e a sua palavra-passe no *tablet* presente no veículo. Desta forma, o utilizador apenas terá de obrigatoriamente introduzir os seus dados de autenticação uma vez (ação que deve ser realizada enquanto o condutor não estiver a conduzir o seu veículo).

Uma vez autenticado o condutor no *Facebook*, a aplicação cliente recebe uma *token* de acesso temporário ao *Facebook*. Esta *token* será enviada à aplicação servidor sempre que esta necessite de se ligar ao *Facebook*. No entanto, a *token* de acesso ao *Facebook* só é válida por um determinado período de tempo, pelo que antes do seu envio para o servidor, deverá ser utilizado o método *isSessionValid()* da classe *Facebook* de forma a descobrir se a sessão do condutor no *Facebook* ainda é válida: se a sessão não for válida, deverá ser efetuado novamente o processo de autenticação do utilizador no *Facebook* e, posteriormente, será enviado o *token* de autenticação para a aplicação servidor; se a sessão for válida, o *token* de autenticação do *Facebook* pode ser enviado para a aplicação servidor.

De forma a descobrir o número de identificação do condutor do veículo no *Facebook*

(que será necessário para a marcação de pontos geográficos na aplicação servidor), a aplicação cliente deverá efetuar um pedido HTTP ao *Facebook* (através do método *request* da classe *Facebook*) a solicitar as informações básicas do condutor do veículo no *Facebook*. O resultado deste pedido HTTP será um objeto JSON que contém o parâmetro "id", cujo valor correspondente é o número de identificação do condutor do veículo no *Facebook*.

O mapa da aplicação cliente e a opção "Marcar Posição"

O componente **Mapa** presente na interface da aplicação cliente foi implementado utilizando o componente *MapView* do *Android*.

Ao iniciar a aplicação cliente, é criada uma instância de um objeto do tipo *LocationManager* (do *package android.location*). Esta instância de *LocationManager* permite que a aplicação obtenha as coordenadas GPS do aparelho GPS presente no dispositivo *tablet* presente no veículo. As coordenadas da localização do veículo são obtidas do *LocationManager* ao iniciar a aplicação cliente e quando o veículo muda de posição.

Assim que as coordenadas GPS do veículo estão disponíveis, é criado um *ItimizedOverlay*, constituído por apenas um item, que será adicionado à *MapView* e identificará o veículo no **Mapa**. De forma a facilitar, ao utilizador, a visualização do veículo e da região envolvente ao veículo no **Mapa**, o item do *ItimizedOverlay* correspondente ao veículo é centrado na *MapView*. O nível de *zoom* da *MapView* é colocado com um valor fixo que permite ao condutor a visualização do seu veículo e da região envolvente próximo ao veículo.

O *LocationManager* é também utilizado de forma a obter a localização do veículo para o envio dessa informação de localização para a aplicação servidor quando o utilizador pressiona o botão **Marcar Posição**. Quando a função "Marcar Posição" é acedida pelo utilizador, a aplicação cliente acede ao método *getLastKnownLocation()* do *LocationManager* de forma a obter a latitude e longitude atuais do veículo. Depois de obtida a posição do veículo, será enviada uma mensagem ao servidor com as suas latitude e longitude e também com o tipo de ponto selecionado pelo utilizador. Assim, de forma a que a aplicação cliente envie a longitude (LON) e a latitude (LAT) atuais do veículo do utilizador *USER*, relativas a um ponto do tipo *TYPE*, para a aplicação servidor, deverá ser enviada uma *string* JSON com o formato:

```
{"serviço":"marcar", "utilizador":"USER", "longitude":"LON",
 "latitude":"LAT", "tipo":"TYPE"}
```

Ao receber a mensagem correspondente à opção "Marcar Posição", a aplicação servidor deverá aceder à base de dados e introduzir a informação recebida da aplicação cliente. A base de dados localizada no servidor é constituída por duas tabelas, a

tabela **PontosInteresse**, que contém os campos **idPonto**, **longitude**, **latitude**, **idFB** e **tipo**, e a tabela **PontosAmigos**, que contém os campos **idPonto**, **idFB**, **longitude** e **latitude**. A tabela **PontosInteresse** contém todos os pontos marcados no serviço **Pontos de Interesse** por utilizadores do **VeículoSocial** (pontos que estão representados nos campos **longitude**, **latitude**) e estabelece uma relação entre um ponto e um utilizador que marcou esse ponto (pois o utilizador que marcou o ponto é registado na tabela **PontosInteresse** através do campo **idFB**, que representa o número de identificação de um utilizador no *Facebook*). A tabela **PontosInteresse** contém ainda o campo **tipo** que identifica a categoria do ponto introduzido. A tabela **PontosAmigos** contém todos os pontos marcados no serviço **Procura de Amigos** por utilizadores do **VeículoSocial** (pontos que estão representados nos campos **longitude** e **latitude**) e estabelece uma relação entre um ponto e um utilizador que marcou esse ponto (pois o utilizador que marcou o ponto é registado na tabela **PontosAmigos** através do campo **idFB**, que representa o número de identificação de um utilizador no *Facebook*).

Assim, ao receber da aplicação cliente a mensagem relativa à função "Marcar Ponto", a aplicação servidor irá retirar os dados relativos ao utilizador, longitude e latitude contidos nessa mensagem. De seguida, a aplicação servidor consulta a tabela **PontosInteresse** da base de dados de forma a descobrir se o ponto recebido já se encontra registado: se o ponto já existir na tabela, é enviada à aplicação cliente uma mensagem a reportar essa situação; se o ponto não existir na tabela, o ponto é registado na base de dados e é enviada uma mensagem para a aplicação cliente a confirmar o registo do ponto. De forma a registar um ponto de longitude *LON* e latitude *LAT* enviado para a aplicação servidor pelo utilizador *USER*, relativo a um ponto do tipo *TYPE*, é executada a seguinte expressão SQL:

```
INSERT INTO PontosInteresse(longitude,latitude,idFB,tipo)
       VALUES (LON,LAT,USER,TYPE);
```

Os serviços "Pontos de Interesse" e "Procura de Amigos"

O serviço **Pontos de Interesse** está ativo na aplicação cliente sempre que o botão **Pontos de Interesse** se encontre selecionado. O serviço **Procura de Amigos** está ativo na aplicação cliente sempre que o botão **Procura de Amigos** se encontre selecionado. No entanto, quando um dos serviços está selecionado, a interface da aplicação cliente não permite que o outro serviço se encontre simultaneamente selecionado.

Quando a aplicação cliente é iniciada, quando um dos botões **Pontos de Interesse** ou **Procura de Amigos** é pressionado, quando um dos botões da **Área de opções dos serviços** for pressionado ou quando o veículo mudar de posição, a aplicação cliente deverá solicitar à aplicação servidor os pontos geográficos que deverão ser

introduzidos no **Mapa**, relativos aos serviços **Pontos de Interesse** e **Procura de Amigos**. Se os pontos requeridos pela aplicação cliente estiverem relacionados com o serviço **Pontos de Interesse**, então o conjunto de pontos devolvido pela aplicação servidor será o conjunto de pontos de interesse introduzidos pelos utilizadores do **VeículoSocial**. Se os pontos requeridos pela aplicação cliente estiverem relacionados com o serviço **Procura de Amigos**, o conjunto de pontos devolvido pela aplicação servidor será o conjunto de pontos relativo à posição atual dos amigos do condutor no *Facebook* que são também utilizadores do **VeículoSocial**.

A execução do serviço **Pontos de Interesse** inicia-se pela obtenção dos tipos e origem dos pontos pretendidos pelo condutor. O conjunto do tipo dos pontos é obtido através da verificação do estado dos **botões de tipo de pontos** da área de botões do serviço **Pontos de Interesse**, enquanto que a origem dos pontos é obtida através da verificação do estado dos **botões de origem dos pontos** presentes na área de botões do serviço **Pontos de Interesse**. A execução do serviço **Procura de Amigos** inicia-se com a obtenção da vontade de partilha da posição atual do veículo por parte do condutor. A vontade de partilha da localização atual do veículo será então obtida através da verificação do estado dos botões **Disponibilizar Posição Atual** e **Esconder Posição Atual** da área de botões do serviço **Procura de Amigos**.

Após verificação dos parâmetros do serviço selecionado pelo condutor do veículo, a aplicação cliente deve enviar para a aplicação servidor uma mensagem JSON que solicite o conjunto de pontos que a aplicação cliente irá adicionar ao **Mapa**.

Para o serviço **Pontos de Interesse** a mensagem JSON enviada à aplicação servidor deverá ter o formato:

```
{"serviço":"pontos", "utilizador":"USER", "token":"TOKEN",
 "origem":ORIG, "tipos":["TIPO1","TIPO2",...]}
```

A chave **serviço** terá o respetivo valor **pontos** de modo a identificar o serviço **Pontos de Interesse**. Os valores USER, TOKEN e ORIG dizem respeito ao número de identificação do condutor no *Facebook*, ao *token* de autenticação do condutor no *Facebook* e à origem dos pontos (ou seja, se os pontos devem ou não ser restritos aos pontos colocados pelos amigos do condutor do *Facebook*). A chave **tipos** diz respeito aos tipos de pontos solicitados pelo condutor do veículo, pelo que o seu valor correspondente será um *array* JSON de *strings*.

Para o serviço **Procura de Amigos** a mensagem JSON enviada à aplicação servidor deverá ter o formato:

```
{"serviço":"amigos", "utilizador":"USER", "token":"TOKEN",
 "longitude":LON, "latitude":LAT, "partilha":PART}
```

A chave **serviço** terá o respetivo valor **amigos** de modo a identificar o serviço **Procura de Amigos**. Os valores USER, TOKEN, LON e LAT dizem respeito ao número de identificação do utilizador no *Facebook*, ao *token* de autenticação do condutor no *Facebook* e à longitude e latitude atuais do veículo. O valor PART, relativo à chave **partilha**, é um valor booleano e indica se o condutor pretende ou não partilhar a sua posição no **VeículoSocial**.

Quando a aplicação servidor recebe uma mensagem JSON de uma aplicação cliente, irá primeiramente verificar na mensagem JSON recebida a chave **serviço** de modo a identificar o serviço solicitado pelo condutor do veículo.

Se o serviço solicitado pelo condutor for o serviço **Pontos de Interesse**, a aplicação servidor irá verificar se o condutor pretende receber apenas os pontos marcados pelos seus amigos do *Facebook*. Se o condutor pretender obter apenas os pontos marcados pelos seus amigos do *Facebook*, a aplicação servidor utilizará o *token* de acesso do condutor e fará um pedido HTTP da forma:

```
https://graph.facebook.com/USER/friends?acess_token=TOKEN
```

A resposta da plataforma *Facebook* a este pedido HTTP será um objeto JSON que contém a lista de amigos do condutor no *Facebook*. De seguida, a aplicação servidor deve aceder à tabela **PontosInteresse** da base de dados e, com o auxílio da lista de amigos do condutor, obter os pontos marcados pelos amigos do condutor do *Facebook*. Se o condutor pretender obter os pontos marcados por todos os utilizador do **VeículoSocial**, a aplicação servidor deve aceder somente à tabela **PontosInteresse** da base de dados e obter todos os pontos registados nessa tabela.

Se o serviço solicitado pelo condutor for o serviço **Procura de Amigos**, a aplicação irá aceder ao *Facebook* e obter a lista de amigos do condutor do *Facebook* (utilizando o mesmo processo descrito para o serviço **Pontos de Interesse**). Uma vez obtida a lista de amigos do condutor, a aplicação servidor deve aceder à tabela **ProcuraAmigos** da base de dados e, com o auxílio da lista de amigos do condutor do *Facebook*, obter todos os pontos relativos à posição atual dos utilizadores amigos do condutor que partilharam a sua posição. De seguida, a aplicação servidor verificará se o condutor pretende partilhar a sua posição atual: se sim, a aplicação servidor acede à tabela **ProcuraAmigos** da base de dados e insere um novo ponto relativo à posição atual do condutor; se não, a aplicação servidor acede à tabela **ProcuraAmigos** da base de dados e verifica se existe algum registo com a posição atual do condutor. Se este registo existir, a aplicação servidor deve removê-lo da tabela **ProcuraAmigos** de forma a que a posição atual do condutor não esteja disponível para os utilizadores do **VeículoSocial**.

Depois de obtidos, por parte da aplicação servidor, os pontos solicitados pelo condutor do veículo, estes terão que ser enviados à aplicação cliente. Para o serviço **Pontos de Interesse** a mensagem enviada à aplicação cliente deverá ser uma *string*

JSON que contém um *array* de objetos JSON, em que cada objeto terá três pares chave/valor relativos ao tipo, longitude e latitude de cada um dos pontos. Assim, a mensagem deverá ter a forma:

```
{"pontos":[{"tipo":"TIPO1", "longitude":LON1, "latitude": LAT1},
  {"tipo":"TIPO2", "longitude":LON2, "latitude": LAT2},
   ...]}
```

Para o serviço **Procura de Amigos** a mensagem enviada à aplicação cliente deverá ser uma *string* JSON que contém um *array* de objetos JSON, em que cada objeto terá dois pares chave/valor relativos à longitude e latitude de cada um dos pontos. Assim, a mensagem deverá ter a forma:

```
{"pontos":[{"longitude":LON1, "latitude": LAT1},
  {"longitude":LON2, "latitude": LAT2},
   ...]}
```

Ao receber a lista de pontos que deverá adicionar no mapa, a aplicação cliente deverá "popular" os *ItimizedOverlays* que serão adicionados à *MapView*. No caso do serviço **Pontos de Interesse**, existirá um *ItimizedOverlay* por cada tipo de pontos de interesse. Assim, a aplicação cliente deverá percorrer a lista de pontos obtidos da aplicação servidor e, através do tipo do ponto, criar um item para cada ponto no *ItimizedOverlay* respetivo. No caso do serviço **Procura de Amigos** só existirá um *ItimizedOverlay*, respetivo aos amigos do condutor no *Facebook* que partilharam a sua posição atual. Assim, a aplicação cliente deverá percorrer a lista de pontos obtidos da aplicação servidor e criar um item para cada ponto no *ItimizedOverlay* respetivo aos amigos do condutor no *Facebook*.

Finalmente, quando os *ItimizedOverlays* respetivos a cada um dos serviços se encontrarem preenchidos, estes serão adicionados à *MapView* de forma a ser permitida a visualização, no **Mapa**, dos pontos solicitados pelo condutor do veículo.

Capítulo 6

Interface do veículo elétrico Gecko Merula

Neste capítulo será apresentada a interface definida nesta dissertação para o veículo elétrico **Gecko Merula**. O veículo **Gecko Merula** é um protótipo de um veículo elétrico produzido em Évora com apoio de estudantes da Universidade de Évora. A interface do *Gecko Merula*, que será acedida num *tablet touch-screen* com sistema *Android*, será constituída por um painel de instrumentos que apresentará componentes que habitualmente se encontram em veículos elétricos e que permitirá a execução da aplicação VeículoSocial.

Serão apresentados os componentes cuja presença é necessária no painel de instrumentos do **Gecko Merula**, e será também apresentada a interface do painel de instrumentos escolhida para o veículo, sendo depois realizada uma análise de performance ao ambiente de desenvolvimento de painéis de instrumentos para veículos definido nesta dissertação. De seguida, será apresentada a interface da aplicação VeículoSocial.

De forma avaliar a apresentação de informação da interface do painel de instrumentos do **GeckoMerula** e da interface da aplicação VeículoSocial, as mesmas serão estudadas com base no capítulo 3, de forma a compreender se as interfaces definidas não causam uma distração excessiva ao condutor na tarefa da condução do veículo.

6.1 Componentes do painel de instrumentos

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

1. **1 velocímetro** - componente que indica a velocidade instantânea do veículo
2. **1 termómetro** - componente que indica a temperatura do líquido de refrigeração do veículo
3. **1 odómetro** - componente que indica distância percorrida pelo veículo ao longo da sua vida
4. **1 taquímetro** - componente que indica o número de rotações do motor do veículo por minuto
5. **1 indicador do nível de bateria do veículo** - componente que indica o nível atua de bateria do veículo
6. **1 indicador de travão de mão** - componente que indica se o veículo está ou não travado com o travão de mão
7. **1 indicador de aviso de bateria** - componente que indica se a bateria do veículo está a receber carga
8. **1 indicador de luz interior** - componente que indica se a luz interior do veículo está ligada
9. **1 indicador de luzes de presença** - componente que indica se a luz de presença do veículo ("mínimos") está ligada
10. **1 indicador de luzes de cruzamento** - componente que indica se a luz de cruzamento do veículo ("médios") está ligada
11. **1 indicador de luzes de estrada** - componente que indica se a luz interior de estrada ("máximos") está ligada
12. **1 indicador de luzes de nevoeiro** - componente que indica se a luz interior de nevoeiro está ligada
13. **2 indicadores de luzes indicadoras de direção** - componentes que indicam a mudança de direção do veículo

6.2 A interface do painel de instrumentos

A interface do painel de instrumentos do veículo **Gecko Merula** foi definida através da utilização do ambiente de desenvolvimento de painéis de instrumentos apresen-

tado no capítulo 4. Esta interface foi desenhada para visualização num *display touch-screen* com resolução de 1280x798 píxeis que se encontrará incluído num *tablet* cujo sistema operativo é o *Android*. O documento XML de definição de componentes do painel de instrumentos utilizado para a definição do painel de instrumentos do **Gecko Merula** pode ser consultado no anexo A.

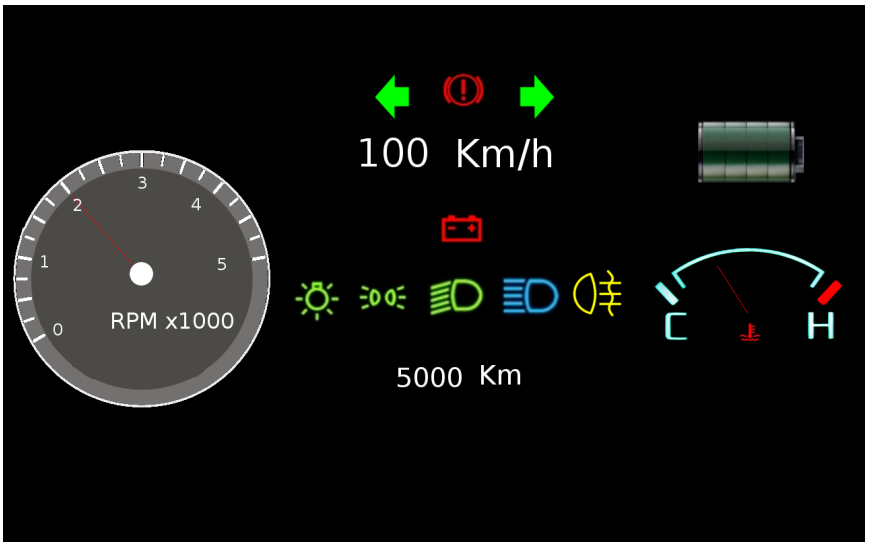

Figura 6.1: Painel de instrumentos do veículo elétrico **Gecko Merula**

A interface definida para o painel de instrumentos do **Gecko Merula** pode ser visualizada na figura 6.1.

Os componentes, do ambiente de desenvolvimento de painéis de instrumentos de veículos definido nesta dissertação, utilizados na definição da interface são:

1. **velocímetro** - são utilizados dois componentes **label**. O componente **label** permite a apresentação de texto no ecrã do veículo que pode ou ser atualizado durante o ciclo de execução do painel de instrumentos. De forma a colocar no *display* a velocidade atual do veículo, é utilizado um **label** atualizável (*refreshable*), enquanto que para a indicação da escala de velocidade (*km/h*) é utilizado um **label** não atualizável (não *refreshable*).

2. **termómetro** - é utilizado um componente **pointerGauge**. O componente **pointerGauge** permite a definição de um elemento que indica um valor através de um ponteiro. Assim, o ponteiro do componente **pointerGauge** utilizado para a implementação do **termómetro** (cujos valores variam entre 50 e 150) aponta para o valor atual da temperatura do líquido de refrigeração do veículo.

3. **odómetro** - são utilizados dois componentes **label**. O primeiro é um **label** atualizável (*refreshable*), e é utilizado para a indicação da distância percorrida pelo veículo durante o seu tempo de vida, enquanto que para a indicação da

escala de distância respetiva ao **odómetro** (*km*) é utilizado um **label** não atualizável (não *refreshable*).

4. **taquímetro** - é utilizado um componente **pointerGauge**. O ponteiro componente **pointerGauge** utilizado para a implementação do **taquímetro** (cujos valores variam entre 0 e 100) aponta para o valor atual do número de rotações por minuto do veículo.

5. **indicador do nível de bateria do veículo** - é utilizado um componente **levelGauge**. O componente **levelGauge** permite a visualização de um componente cujo estado é indicado por vários "níveis". Assim, o número de níveis apresentado no componente **levelGauge** utilizado na implementação do **indicador do nível de bateria do veículo** indica a percentagem de bateria atual do veículo.

6. **restantes indicadores** - são utilizados componentes **indicator**. Os vários **indicadores** presentes no veículo (indicadores de luzes, de travão de mão e de aviso de bateria) possuem dois estados: o estado ativo e o estado inativo. Assim, de forma a representar os estados ativo e inativo de cada um dos **indicadores**, são utilizados componentes **indicator** que permitem a apresentação de uma imagem para o componente quando este está ativo e uma outra imagem para o componente quando este está inativo.

Teste de desempenho do ambiente de desenvolvimento

De modo a testar o desempenho do ambiente de desenvolvimento de painéis de instrumentos definido nesta dissertação, foram realizados testes relativos ao tempo médio de desenho da totalidade dos componentes do painel no ecrã. Estes testes foram realizados numa **plataforma de teste** composta por um simulador, disponibilizado pela *Google*, do sistema *Android*, que foi executado num computador *HP Pavilion dv5* com 4 Gb de memória RAM e com um processador *Duo Core* de 2.40 GHz.

Nos testes efetuados ao ambiente de desenvolvimento de painéis de instrumentos foram incluídos componentes dos tipos *pointerGauge*, *levelGauge*, *indicator*, *staticImage*, *label* e *button*. Cada teste é composto por uma interface que contém, respetivamente, 1, 10, 20, 40, 60, 80 e 100 componentes de cada um destes tipos, pelo que o número mínimo de componentes testados foi de 7 componentes (1 componente de cada um dos 7 tipos de componentes testados) e o número máximo de componentes testados foi de 700 componentes (100 componentes de cada um dos 7 tipos de componentes testados).

As medições realizadas estão relacionadas com o tempo médio (expresso em milisegundos) necessário para desenhar todos os componentes da interface numa iteração do método *onDraw()* da plataforma *Android*. De forma a obter o tempo médio de

Nº de componentes de cada tipo	**Número total de elementos**	**Média de tempo de desenho (ms)**	**Desvio padrão do tempo de desenho (ms)**
1	7	100,73	17,1
10	70	243,73	73,77
20	140	377,46	383,2
40	280	659,49	476,1
60	420	917,96	717,26
80	560	1228,35	757,85
100	700	1393,32	920,29

s

Tabela 6.1: Valores dos testes de desempenho do desenho de componentes no ecrã da **plataforma de teste**

desenho no ecrã de todos os componentes testados, foi realizada, para cada uma das diferentes quantidades de componentes desenhados, a média do tempo de desenho ao fim de 100 iterações do método *onDraw()*. Os resultados destas medições podem ser consultados na tabela 6.1 e no gráfico da figura 6.2.

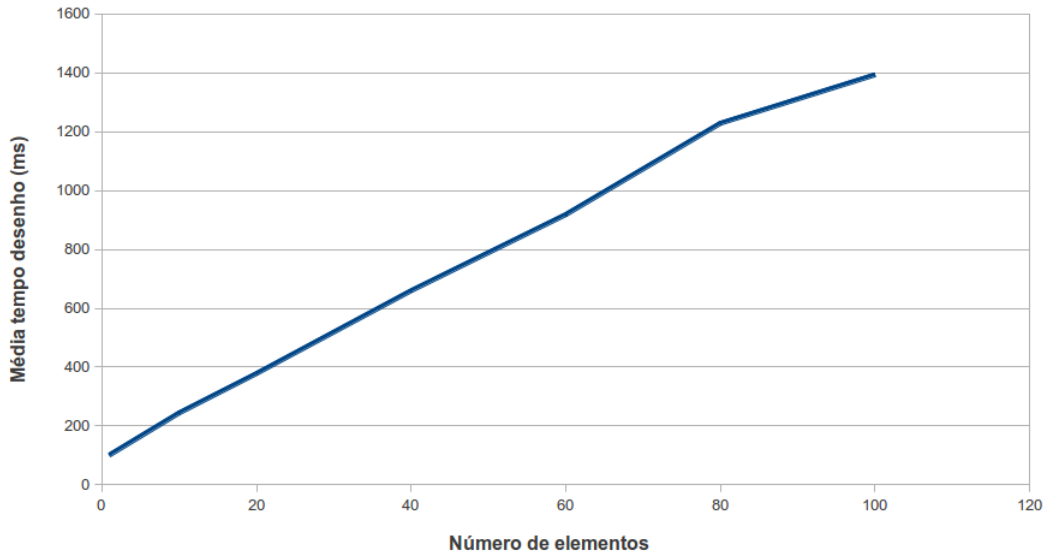

Figura 6.2: Gráfico de tempo médio de desenho dos componentes no ecrã da **plataforma de teste**

Os resultados apresentados na **plataforma de teste** indicam que o tempo médio de desenho dos componentes no ecrã cresce de forma aproximadamente linear com o número de componentes desenhados. Similarmente, a variabilidade do tempo de desenho dos componentes no ecrã da **plataforma de teste** cresce também de forma aproximadamente linear com o número de componentes desenhados.

Estes resultados parecem indicar que é necessária precaução com o número de componentes definidos ao utilizar o ambiente de desenvolvimento, pois à medida que o

número de componentes aumenta, o tempo de desenho dos mesmos no ecrã também aumenta, implicando que o tempo de resposta do painel de instrumentos ao *input* de utilizador diminui. A título de exemplo, considere-se, de acordo com a **plataforma de teste**, a diferença entre um painel de instrumentos composto por 20 componentes, *painel1*, e um painel de instrumentos composto por 60 componentes, *painel2*. O tempo médio de desenho dos componentes no ecrã do *painel1* é de 243,73 milisegundos, o que significa que em 1 segundo o *painel1* é atualizado cerca de 4 vezes na **plataforma de teste**. O tempo médio de desenho dos componentes no ecrã do *painel2* é de 917,96 milisegundos, o que significa que em 1 segundo o *painel2* é atualizado apenas 1 vez na **plataforma de teste**. Apesar de não existir um tempo de resposta padrão para painéis de instrumentos, pode considerar-se que não é exequível que um painel demore aproximadamente 1 segundo a atualizar o seu conteúdo, como é o caso do painel *painel2*, visto que as informações críticas ao ato da condução, como a velocidade instantânea do veículo, requerem uma maior precisão de apresentação ao condutor. No entanto, pode considerar-se que é perfeitamente aceitável que um painel de instrumentos atualize 4 vezes o seu conteúdo em 1 segundo, como é o caso do painel *painel1*. Assim, pode assumir-se que o número de componentes definido num painel de instrumentos, através da utilização do ambiente de desenvolvimento definido nesta dissertação e em relação à **plataforma de teste**, deve ser limitado, pois pode afetar a precisão com que um condutor recebe informações críticas durante o manuseio do seu veículo.

É necessário, no entanto, realizar um maior número de testes ao tempo de desenho de componentes utilizando o ambiente de desenvolvimento. De forma a alcançar um conjunto de resultados relativos à precisão de atualização do ambiente de desenvolvimento devem ser realizados, idealmente, vários testes em situação real de condução e em *hardware* dedicado (contrariamente aos testes realizados nesta dissertação, que foram feitos num simulador *Android*). Não foi possível, porém, realizar estes testes no âmbito do trabalho desenvolvido para esta dissertação, pelo que se assumirá que, para a **plataforma de teste**, o número de componentes desenhados no ecrã deve ser limitado.

6.3 A interface da aplicação VeículoSocial

Ao iniciar o VeículoSocial, é feita a autenticação do condutor na plataforma *Facebook*. Como esta tarefa requer ações relativamente complexas por parte do condutor enquanto este conduz o seu veículo, esta tarefa deve ser realizada enquanto o veículo está parado. Depois de feita a autenticação, o condutor poderá guardar os seus dados de autenticação do *Facebook* no dispositivo *Android* instalado no *Gecko Merula*.

O condutor pode efetuar o *login* no *Facebook* através de uma *OAuth Dialog* que será disponibilizada pela aplicação VeículoSocial, na qual o condutor terá que colocar o seu nome de acesso e a palavra-passe de acesso ao *Facebook*, como se pode ver na

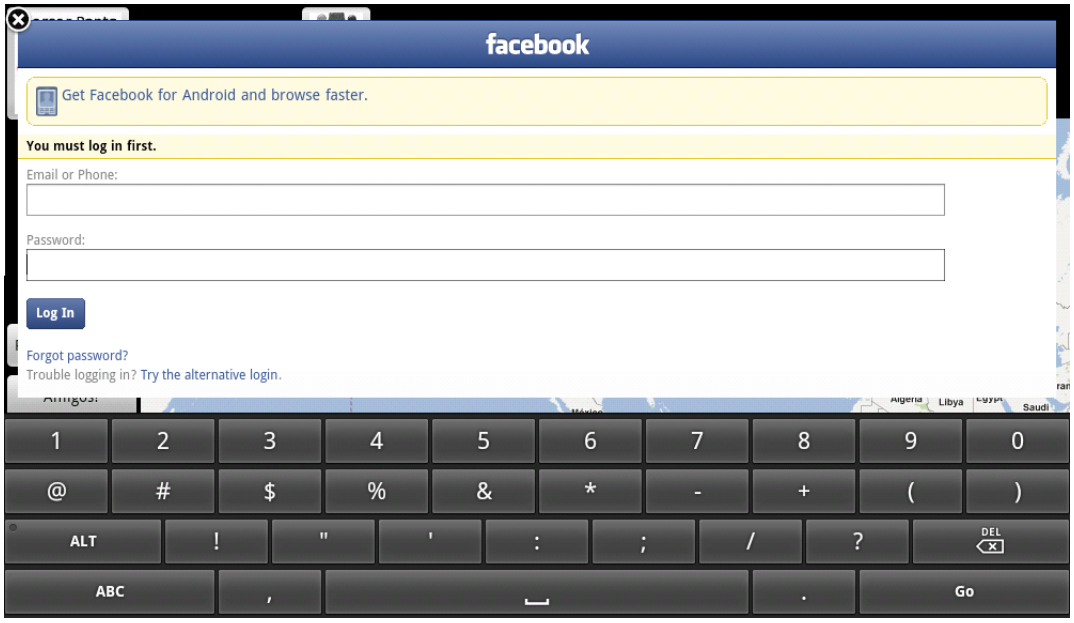

Figura 6.3: Autenticação *Facebook* na aplicação VeículoSocial

figura 6.3.

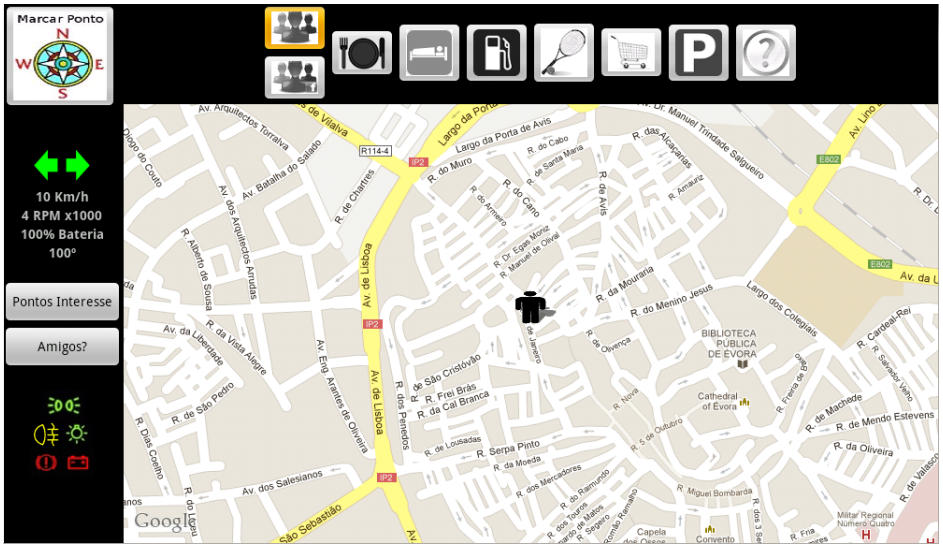

Figura 6.4: Ecrã de mapa da aplicação VeículoSocial

Assim que o utilizador seja autenticado no *Facebook*, será apresentado o ecrã do mapa da aplicação (figura 6.4).

O ecrã do mapa da aplicação VeículoSocial é constituídos pelo mapa, pelos indicadores de estado atual do veículo e pelos diversos botões de interação do utilizador com a aplicação. Inicialmente, a aplicação executa o serviço "Pontos de Interesse", podendo o utilizador escolher o serviço que pretende utilizar nos botões centrais es-

querdos da interface. Os indicadores presentes no lado esquerdo da aplicação apresentam diversas indicações acerca do estado atual do veículo (indicações de luzes, de velocidade atual, de número de rotações por minuto, de percentagem de bateria restante, de temperatura atual do veículo e indicadores de travão de emergência e de nível de bateria). No mapa pode ser visualizado um símbolo respetivo à posição atual do condutor, sendo que a posição deste símbolo no mapa será atualizada à medida que o veículo se movimenta. O botão de "Marcar Posição" encontra-se na posição superior esquerda da interface, enquanto que os botões de tipo de pontos, referentes ao serviço "Pontos de Interesse", se encontram na posição superior central da interface. Os botões de origem dos pontos do mapa encontram-se à esquerda dos botões de tipo de pontos.

Figura 6.5: Ecrã de mapa da aplicação VeículoSocial com pontos de origem geral do serviço "Pontos de Interesse"

Inicialmente, os botões do tipo de pontos encontram-se inativos, sendo que quando um dos botões é pressionado pelo utilizador, o seu estado muda de ativo para inativo e poderão ser visualizados, no mapa, os símbolos relativos aos pontos das categorias escolhidas pelo condutor. Na figura 6.5 poderá ser visualizada a interface da aplicação em que o mapa apresenta pontos de origem geral pertencentes às categorias Restauração e Abastecimento.

Se a origem dos pontos for o *Facebook*, apenas serão apresentados, no mapa, os pontos das categorias escolhidas pelo condutor que foram marcados na aplicação pelos amigos do condutor no *Facebook*. Na figura 6.6 poderá ser visualizada a interface da aplicação em que o mapa apresenta pontos de origem do *Facebook* pertencentes à categoria Alojamento.

No caso do condutor escolher o serviço "Procura de Amigos", os botões de topo da interface mudarão para os botões de disponibilização da posição atual do condutor

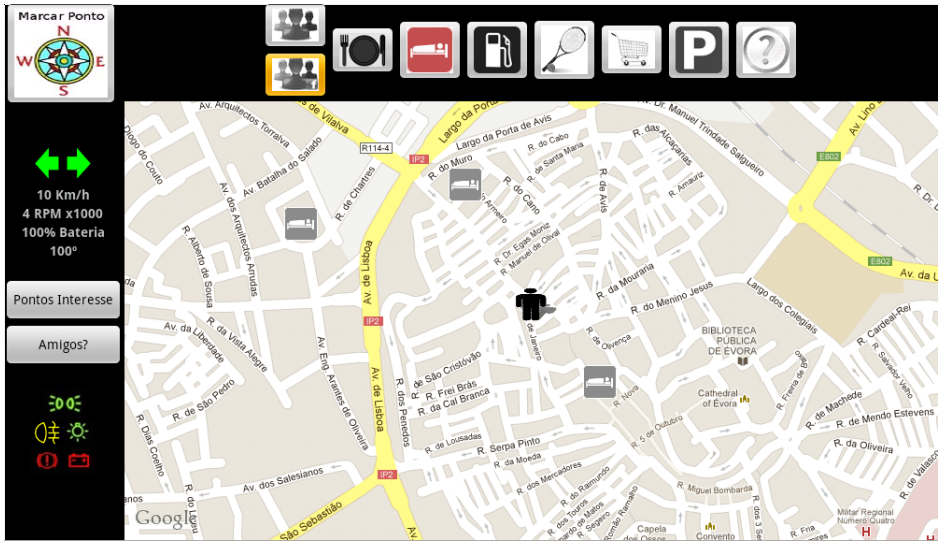

Figura 6.6: Ecrã de mapa da aplicação VeículoSocial com pontos de origem do *Facebook* do serviço "Procura de Amigos"

aos seus amigos do *Facebook*. Os elementos apresentados no mapa serão símbolos respetivos à posição dos amigos do condutor. Na figura 6.7 poderá ser visualizada a interface da aplicação em que o mapa apresenta pontos do serviço "Procura de Amigos".

6.4 Apreciação das interfaces definidas

Com o objetivo de avaliar as interfaces definidas nesta dissertação, as mesmas são comparadas, nesta secção, com os aspetos considerados relevantes dos princípios de apresentação de informação em veículos apresentados no capítulo 3. Para cada um dos princípios apresentados, são avaliadas as interfaces do painel de instrumentos e a interface da aplicação VeículoSocial, sendo que cada princípio será considerado *verificado* ou *não verificado* para as duas interfaces.

1. **A Interface deve ser desenhada de modo a que as tarefas que permite desempenhar sejam feitas com breves *glances* que não afetem gravemente a tarefa de condução**

   A verificação deste princípio pretende concluir se uma interface minimiza o quanto possível a distração do condutor, requerida pelo *display*, enquanto conduz o seu veículo. De forma a verificar este princípio, poderá ser estudado o tempo de *glance* exigido pela interface ao condutor, ou pode ainda definir-se um conjunto de tarefas de referência possíveis de efetuar na interface e comparar-se o número de *lane exceedances* e a variação de *following headway* do condutor, enquanto realiza as tarefas de referência, em duas situações: condução sem

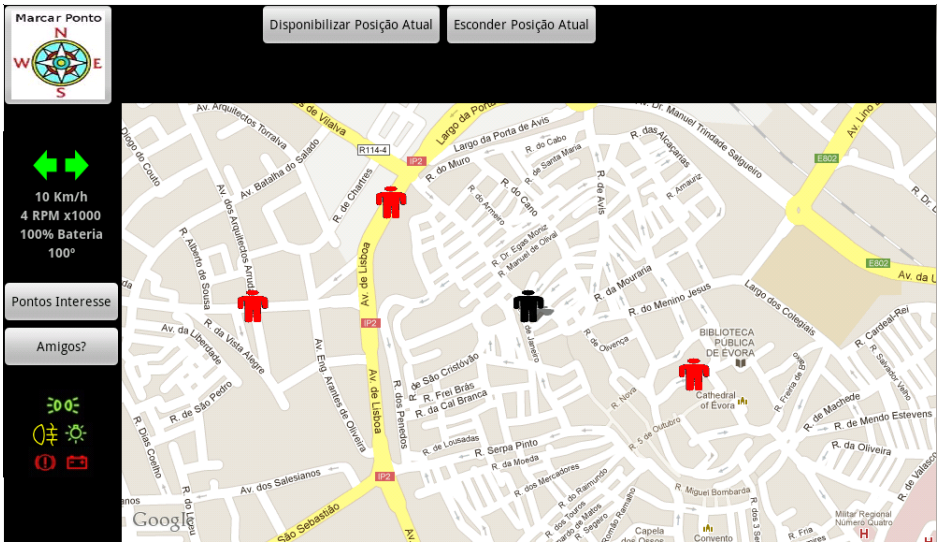

Figura 6.7: Ecrã de mapa da aplicação VeículoSocial com pontos do serviço "Procura de Amigos"

utilização do *display* e condução com ocasional utilização do *display*.

A verificação deste princípio implica a realização de testes em cenários reais de condução ou em simuladores de cenários reais de condução. No entanto, nenhum destes testes foi realizado no trabalho desenvolvido para esta dissertação, pelo que este princípio se considera **não verificado** para ambas as interfaces.

2. **Devem ser usados *standards* de legibilidade, ícones, símbolos, palavras, acrónimos ou abreviações. Se tais *standards* não existirem, devem ser usadas diretrizes de *design* ou dados empíricos**

   A verificação deste princípio pretende concluir se a informação visual fornecida por uma interface é facilmente compreendida e rapidamente apreendida por parte do condutor. A fácil compreensão e rápida apreensão de uma interface de um veículo por parte do condutor auxiliam a tarefa de minimização de distração do mesmo. Para que este princípio se possa considerar verificado, uma interface deve utilizar padrões de ícones, símbolos, palavras, acrónimos e abreviações sempre que possível. Se a utilização destes padrões não for possível, a interface deve ser composta de elementos de fácil compreensão e rápida apreensão para o condutor.

   A interface do **painel de instrumentos** utiliza um conjunto de símbolos para a apresentação de informação relevante à tarefa de condução. Todos os símbolos utilizados no **painel de instrumentos** estão incluídos na ISO 2575, que é um padrão que regulariza os símbolos que devem ser apresentados num veículo, e a posição destes elementos no ecrã foi feita de forma a permitir uma boa visualização dos mesmos por parte do condutor. As abreviações utilizadas nesta interface (*km/h* para o velocímetro, *km* para o odómetro e *RPM x1000*

para o taquímetro) estão incluídas no padrão FMVSS 101, que é um padrão que regulariza a apresentação de abreviações em veículos. Assim, este princípio considera-se **verificado** para o **painel de instrumentos** do **Gecko Merula**.

As categorias dos símbolos e palavras utilizados no **VeículoSocial** não estão definidas nos padrões de apresentação de informação em veículos (com a exceção dos símbolos utilizados para representação do estado atual do veículo). Porém, considera-se que a interface da aplicação **VeículoSocial** utiliza ícones que são facilmente reconhecidos e apreendidos por parte de um condutor (como é o exemplo dos símbolos utilizados para distinguir as categorias de pontos apresentadas no mapa). A categoria da informação textual apresentada nesta aplicação também não se encontra definida nos padrões de apresentação de informação em veículos. No entanto, a informação textual apresentada no **VeículoSocial** é constituída por breves descrições textuais, com um tamanho de fonte de letra que se considera facilmente legível para o condutor. Assim, este princípio considera-se **verificado** para a aplicação **VeículoSocial**.

3. **A informação relevante à tarefa de condução deve ser precisa e atempada**

   A verificação deste princípio tem como objetivo minimizar as tomadas de decisões erróneas por parte do condutor. Para minimizar os erros cometidos pelo condutor, uma interface deve apresentar as informações atempadas e adequadas às diversas situações de condução (ou seja, deve apresentar as informações relevantes para o condutor enquanto manuseia o seu veículo). De modo a verificar este princípio, uma interface não deve apresentar uma quantidade de informação exagerada, não deve apresentar informação errada nem deve apresentar informação pouco relevante ao condutor.

   Os elementos presentes na interface do **painel de instrumentos** são os elementos considerados indispensáveis na tarefa de condução de um veículo, pelo que as informações apresentadas nesta interface são, por isso, relevantes à tarefa da condução de um veículo. As informações apresentadas nesta interface dizem respeito ao estado atual do veículo, pelo que, no geral, se pode considerar que as informações apresentadas estarão corretas e que não é apresentada uma quantidade exagerada de informação ao condutor. Assim, este princípio considera-se **verificado** para o **painel de instrumentos** do **Gecko Merula**.

   As informações apresentadas no **VeículoSocial** dizem respeito a elementos considerados indispensáveis na tarefa de condução de um veículo e a pontos geográficos de interesse para o condutor. Em relação aos primeiros elementos, e tal como acontece na interface do **painel de instrumentos**, as informações apresentadas dizem respeito ao estado atual do veículo, pelo que se considera que as informações estão corretas e não surgem em quantidade exagerada. Em relação aos segundos elementos, a quantidade de informação apresentada será sujeita a uma filtragem por parte do condutor, pelo que o número de

informações disponíveis ao mesmo será controlado pelo próprio condutor. Não é possível, no entanto, verificar a veracidade das informações disponíveis no **VeículoSocial**, visto que os vários pontos colocados no mapa são disponibilizados pelos vários utilizadores da aplicação. No entanto, pode considerar-se que a filtragem dos conteúdos com recurso aos amigos de um condutor no *Facebook* realizada pela aplicação minimiza o número de informações erradas que estarão disponíveis a um condutor. Assim, este princípio considera-se **verificado** para a aplicação **VeículoSocial**.

4. **O sistema não deve requerer que o condutor retire mais do que uma mão do volante**

   Este princípio tem como objetivo garantir que o condutor mantém no mínimo uma mão no volante de modo a não colocar em risco a sua segurança, bem como a segurança dos restantes possíveis ocupantes do veículo. A verificação deste princípio é feita garantindo que a interface definida permita que o condutor utilize apenas uma mão para fornecer os *inputs* de informação necessários à mesma.

   O **painel de instrumentos** não possui nenhum componente que necessite de *input* do condutor através do *touch-screen* do veículo, pelo que o condutor não necessita de usar as mãos para interagir com o veículo. Assim, este princípio considera-se **verificado** para o **painel de instrumentos** do **Gecko Merula**.

   Os pedidos de *input* realizados pela aplicação **VeículoSocial** (à excepção do *login* do condutor no *Facebook*, que não deve ser realizado enquando o veículo se encontra em movimento) são feitos através de botões que necessitam de apenas um toque para fornecer o *input* necessário, garantindo assim que o condutor mantenha sempre pelo menos uma mão no volante. Assim, este princípio considera-se **verificado** para a aplicação **VeículoSocial**.

5. **O sistema não deve requerer sequências ininterruptas de interações manuais ou visíveis**

   Este princípio tem como objetivo garantir que o condutor de um veículo não se distraia da tarefa da condução devido a uma sequência ininterruptas de *inputs* realizados pelo mesmo na interface. Este princípio será verificado se as sequências de *inputs* requeridas na execução de uma tarefa pela interface possa ser interrompida, de modo a que o condutor possa voltar facilmente ao mesmo ponto lógico de execução da tarefa sem perder os *inputs* anteriores.

   O **painel de instrumentos** não apresenta nenhuma sequência lógica de *inputs* para a realização de uma tarefa. Assim, este princípio considera-se **verificado** para o **painel de instrumentos** do **Gecko Merula**.

   A aplicação **VeículoSocial** necessita apenas de um tipo de sequência de *inputs* para completar uma tarefa: para o serviço **Pontos de Interesse**, de modo a obter os pontos de interesse pretendidos, o condutor necessita de escolher

a origem dos dados e os tipos de pontos que pretende obter. A interação do condutor com este serviço foi efetuada com recurso a botões que apenas mantém o seu estado quando são pressionados pelo condutor, pelo que esta interação pode ser interrompida sem que o condutor perca a sequência lógica de execução da tarefa e sem que o condutor perca os *inputs* anteriores. Assim, este princípio considera-se **verificado** para a aplicação **VeículoSocial**.

6. **O condutor deve poder controlar, salvo algumas exceções, o ritmo de interação com o sistema. O sistema não deve requerer que o condutor forneça *inputs* urgentes**

   Este princípio tem por objetivo evitar que a interface do veículo distraia o condutor da tarefa primária da condução através de pedidos de *input* urgentes e que não podem ser ignorados pelo condutor. De modo a verificar este princípio, basta que a interface do veículo não necessite de *inputs* urgentes ou que no caso do condutor ignorar um pedido de *input*, a interface regresse para um estado apropriado para a tarefa iniciada pelo condutor.

   O **painel de instrumentos** não apresenta nenhuma tarefa que necessite de *inputs* urgentes por parte do condutor, pelo que este princípio se considera **verificado** para o **painel de instrumentos** do **Gecko Merula**.

   Os serviços disponíveis na aplicação **VeículoSocial** não necessitam que o condutor forneça *inputs* urgentes, pois o condutor apenas terá de fornecer *inputs* à aplicação quando este pretender obter informações de localização diferentes das que escolheu anteriormente. Assim, este princípio considera-se **verificado** para a aplicação **VeículoSocial**.

7. **A resposta do sistema a um *input* do condutor deve ser atempada e claramente percetível**

   Este princípio considera-se verificado se for percetível ao condutor que a interface recebeu o seu pedido de *input* e se a consequência de tal *input* for óbvia para o condutor. Para se considerar que a resposta a um pedido de *input* do condutor por parte da interface é percetível, a interface deve estar de acordo com a ISO 15005, que indica que a resposta de uma interface a um pedido de *input* não deve exceder os 250 milisegundos e que, no caso do tempo de resposta da interface exceder os 2 segundos, a interface deve apresentar uma informação a indicar que o pedido de *input* do utilizador está a ser processado.

   Este princípio não se aplica a painéis de instrumentos "tradicionais", visto que a ISO 15005 se refere especificamente a sistemas de informação e comunicação que necessitem de *input* do condutor. Como a interface do **painel de instrumentos** do **Gecko Merula** não necessita de *input* do condutor, este princípio considera-se **verificado**.

   As tarefas executadas na aplicação **VeículoSocial** necessitam de uma ligação a um servidor, o que implica que o tempo de resposta a um *input* de utilizador

pode exceder os 250 milisegundos. No caso deste tempo de resposta exceder os 2 segundos, o **VeículoSocial** não apresenta nenhuma mensagem a informar o condutor que o seu pedido de *input* está a ser processado, pelo que se considera que este princípio é **não verificado** para a aplicação **VeículoSocial**.

8. **A informação visual que não está relacionada com a prática da condução e que poderá distrair significativamente o condutor deverá estar inacessível ao condutor durante a condução do veículo**

   Este princípio tem como objetivo garantir que a informação visual fornecida pela interface ao condutor apresente somente informações que estão relacionadas com a prática da condução.

   As informações apresentadas ao condutor pelo **painel de instrumentos** dizem respeito às informações essenciais ao condutor para o ato da condução, pelo que este princípio se considera **verificado** para o **painel de instrumentos**.

   As informações apresentadas ao condutor na aplicação **VeículoSocial** dizem respeito a informações essenciais ao condutor e a localizações geográficas de interesse para o condutor quando este conduz o seu veículo, pelo que este princípio se considera **verificado** para a aplicação **VeículoSocial**.

9. **O sistema não deve permitir que o condutor tenha acesso a tarefas que não sejam dirigidas ao mesmo enquanto o veículo se encontra em movimento**

   Este princípio tem como objetivo separar as tarefas acessíveis das inacessíveis ao condutor na interface de um veículo, de forma a que, durante o ato da condução, apenas estejam disponíveis ao condutor as tarefas que lhe sejam relevantes quando o veículo se encontra em movimento.

   Todas as tarefas incluídas nas interfaces do **painel de instrumentos** e na aplicação **VeículoSocial** são relevantes à tarefa da condução, pelo que este princípio se encontra **verificado** para ambas as interfaces.

10. **Informações acerca do estado do veículo que possam ter impacto na segurança dos seus ocupantes devem ser apresentadas ao condutor**

    Este princípio tem o objetivo de garantir que o condutor possui informações suficientes acerca do estado do seu veículo para tomar decisões responsáveis durante o ato da condução. De forma a verificar este princípio, uma interface deve apresentar as respetivas indicações sempre que o veículo apresentar um problema.

    A interface do **painel de instrumentos** possui um conjunto de indicadores cujo objetivo é o de informar o condutor acerca do estado do seu veículo. Estes indicadores da interface do **painel de instrumentos** mantém-se inativos se o veículo não apresentar nenhum problema e apresentam-se ativos se for detetado

algum problema no veículo. Assim, este princípio considera-se **verificado** para o **painel de instrumentos** do **Gecko Merula**.

Para além de permitir a visualização de pontos geográficos de interesse para o condutor do veículo, a interface da aplicação **VeículoSocial** permite ainda a visualização de indicadores que informam o condutor acerca do estado atual do veículo. Tal como na interface do **painel de instrumentos**, estes indicadores mantém-se inativos se o veículo não apresentar nenhum problema e apresentam-se ativos se for detetado algum problema no veículo. Assim, este princípio considera-se **verificado** para a aplicação **VeículoSocial**.

Capítulo 7

Conclusões

Nesta dissertação foi discutido o tema de definição de painéis de instrumentos para veículos automóveis e da integração de redes sociais nos mesmos.

Com vista a definir uma interface para o veículo elétrico *Gecko Merula*, foi referido um conjunto de princípios de apresentação de informação em veículos automóveis. Neste conjunto de princípios foi referida a importância fulcral que o painel de instrumentos de um veículo deve ter na redução de atenção do condutor à tarefa primária da condução. Considera-se que destes princípios se retira fundamentalmente que:

1. Os componentes do painel de instrumentos de um veículo devem ser desenhados de forma a que o condutor ganhe rapidamente um alto nível de familiaridade com os mesmos;

2. As informações dadas pelo painel de instrumentos de um veículo ao condutor devem ser corretas e relevantes para a situação atual de condução para o condutor, sendo que o painel de instrumentos não deve apresentar informações nem necessitar de interações que coloquem em risco a vida do condutor.

Nesta dissertação foi definido um ambiente de desenvolvimento para painéis de instrumentos de veículos. O ambiente de desenvolvimento definido separa os elementos que fornecem informações sobre o estado do veículo dos elementos que definem a interface gráfica do painel de instrumentos. O ambiente de desenvolvimento definido nesta dissertação efetua também a separação entre a aparência de um painel de instrumentos e o comportamento e apresentação dos componentes do painel de instrumentos no *display*, garantindo assim uma interdisciplinaridade no grupo de trabalho responsável pela definição da interface de um veículo. A aparência do pai-

nel de instrumentos é definida num documento XML de definição de componentes no qual se colocam os elementos e as respetivas propriedades que se pretendem incluir no painel. O comportamento e a apresentação do painel de instrumentos pode ser definida em qualquer linguagem de programação que tenha capacidade de realizar *parsing* de um documento XML.

Na dissertação foi apresentada uma proposta de implementação, em *Java*, do comportamento e apresentação dos componentes no *display* de um veículo. Para tal, foram criadas classes *Java* que auxiliam a tarefa de desenho no ecrã do veículo e na obtenção das propriedades dos componentes definidos no documento XML de definição de componentes de um painel de instrumentos.

Foram também apresentadas interfaces *Java* que facilitam a portabilidade do código *Java* para outra plataformas que utilizam essa linguagem de programação. Usando estas interfaces, foi descrita a implementação do ambiente de desenvolvimento em *Android*.

Na presente dissertação foi definida uma aplicação que utiliza a ligação a uma rede social num veículo automóvel. Foram estudadas, primeiramente, as vantagens que a ligação a uma rede social pode apresentar durante o ato de condução de um veículo. Concluiu-se que podem ser aproveitadas as ligações sociais previamente estabelecidas pelos utilizadores das redes sociais para ser realizado um filtro de conteúdos relevantes a um determinado indivíduo, em função das suas relações numa rede social.

Assim, foi apresentada a aplicação VeículoSocial que é uma aplicação *Android* que utiliza o modelo cliente-servidor para comunicação entre um veículo e um servidor. A aplicação VeículoSocial utiliza essa ligação entre o veículo e o servidor de forma a que o condutor de um veículo possa partilhar, para os outros utilizadores da aplicação, um ponto de interesse incluído numa de sete categorias possíveis. O VeículoSocial faz uso de uma outra ligação, à rede social *Facebook*, de forma a filtrar conteúdos de localização. Esta filtragem permite que um condutor obtenha pontos de interesse que são relevantes para si e para os seus amigos do *Facebook*. Esta filtragem permite também que um condutor partilhe a sua posição atual com utilizadores do VeículoSocial que são simultaneamente seus amigos no *Facebook*. A visualização dos pontos de interesse e da posição dos amigos do condutor é feita num mapa, da *Google Maps*, incluído no ecrã do veículo. A interface do VeículoSocial foi também posteriormente apresentada.

Nesta dissertação foi definida a interface do veículo *Gecko Merula*. Esta interface foi sujeita a uma avaliação com base nos princípios apresentados na secção 3. Essa avaliação permitiu descobrir se esse princípios eram totalmente verificados na interface definida. Desta avaliação, concluiu-se que apenas um dos princípios não foi verificado. Trata-se do princípio no qual uma interface deve ser desenhada de modo a que o condutor não necessite de mais do que breves *glances* para apreender

as informações necessárias ao ato da condução. A não verificação deste princípio deveu-se ao facto de que nesta dissertação não foram realizados os testes necessários à verificação do mesmo.

A interface da aplicação VeículoSocial foi também avaliada com base nos princípios apresentados na secção 3. Dos 10 princípios avaliados, 2 não foram verificados pelo VeículoSocial:

1. O princípio que indica que uma interface de um veículo deve ser desenhada de modo a que as tarefas que pretende desempenhar sejam feitas com breves *glances* não foi verificado pela mesma razão que no caso do painel de instrumentos, ou seja, não foram efetuados nesta dissertação os testes necessários à sua verificação;

2. O princípio que indica que a resposta do sistema a um *input* de utilizador deve ser atempada e claramente percetível não foi verificado pois o VeículoSocial não garante um tempo de resposta ao utilizador que não exceda os 250 milisegundos, nem apresenta uma mensagem de aviso quando esta resposta demora mais de 2 segundos;

Foram também realizados testes de desempenho ao ambiente de desenvolvimento definido nesta dissertação. Estes testes foram realizados numa plataforma de teste, que corria um simulador *Android*, e foi medido o tempo médio de desenho no ecrã de um conjunto de interfaces que incluíram um número diferente de componentes entre si. A partir dos resultados obtidos por estes testes, considerou-se que, para a plataforma de teste, o tempo médio de desenho de componentes de uma interface, definida no ambiente de desenvolvimento apresentado nesta dissertação, cresce de forma aproximadamente linear com o número de componentes que compõem a interface. Estes resultados parecem indicar que o número de componentes incluídos na interface definida no ambiente de desenvolvimento não deve ser demasiado elevado pois poderá afetar o tempo e a precisão de resposta da interface ao condutor.

7.1 Principais contribuições

Considera-se que as principais contribuições dadas por esta dissertação são:

1. A apresentação de um conjunto de princípios que podem ser utilizados no desenvolvimento de um painel de instrumentos;

2. A definição de um ambiente de desenvolvimento de painéis de instrumentos que: permite a interdisciplinaridade do grupo de trabalho; permite a independência em relação à escolha da linguagem de programação a utilizar; auxilia a portabilidade do código para plataformas que utilizem a linguagem de programação *Java*;

3. A definição de uma aplicação para *Android* que faz uso da rede social que possui mais utilizadores na atualidade, o *Facebook*, de forma a filtrar conteúdos de localização.

7.2 Trabalho Futuro

O trabalho futuro que pode ser realizado para as contribuições dadas por esta dissertação é:

1. A definição de um conjunto de tarefas de referência das interfaces do painel de instrumentos *Gecko Merula* e da aplicação VeículoSocial e a comparação, em duas situações diferentes (condução sem recurso ao *display* e condução com ocasional uso do *display*), do número de *lane exceedances* e da variação de *following headway*. Estas contagens serão efetuadas de forma a verificar o princípio no qual as interfaces dos veículos devem ser desenhadas de modo a que as tarefas que permitem desempenhar sejam feitas com breves *glances*. Se para cada uma das interfaces o número de *lane exceedances*, realizadas por um condutor, não for maior, com o uso do *display*, que o número de *lane exceedances* realizadas sem o uso do *display*, e se a variação de *following headway* com uso do *display* com uso do *display*, enquanto o condutor realiza uma tarefa de referência da interface, não for menor que a variação de *following headway* sem o uso do *display*, então o princípio considera-se verificado para a interface do *Gecko Merula* e para a interface do VeículoSocial;

2. Definir, para a aplicação VeículoSocial, uma mensagem de aviso para o condutor para quando a resposta a um pedido de *input* do condutor exceder os 2 segundos. A definição desta mensagem permite que o princípio que indica que a resposta do sistema a um *input* de utilizador deve ser atempada e claramente percetível seja verificada para a aplicação VeículoSocial;

3. Modificar a visualização dos pontos relativos aos amigos do condutor no serviço "Procura de Amigos" da aplicação VeículoSocial, de forma a que seja possível distinguir os vários utilizadores representados no mapa (atualmente todos os amigos de um condutor são representados pelo mesmo ícone);

4. Deve ser realizada uma extensão à aplicação VeículoSocial de forma a que esta se possa ligar a mais redes sociais para além do *Facebook*. Ao permitir a ligação entre a mais redes sociais, o VeículoSocial ganha a capacidade de ser utilizado por mais indivíduos e ganha a capacidade de utilizar mais ligações sociais entre indivíduos como filtro de conteúdos de localização;

5. Melhorar a aplicação servidor para que não necessite de enviar todos os pontos da base de dados para o veículo, reduzindo assim a carga das aplicações cliente

e servidor. Este melhoramento pode ser efetuado através de uma filtragem, realizada pelo servidor, dos pontos que se encontram mais próximos do veículo;

6. Testar a interface *Gecko Merula* num cenário de condução real no veículo *Gecko Merula*;

7. Realização dos testes de desempenho do ambiente de desenvolvimento em mais máquinas, preferencialmente em *tablets*, utilizando *hardware* dedicado, ao invés da utilização do simulador *Android*, de forma a obter resultados mais fidedignos.

Assim, após os melhoramentos descritos nesta secção, espera-se que as aplicações apresentadas nesta dissertação chamem a atenção de eventuais colaboradores de desenvolvimento de forma a que possam ser aplicadas em cenários de condução real.

Bibliografia

[AAM, 2006] AAM (2006). Statement of principles, criteria and verification procedures on driver interactions with advanced in-vehicle information and communication systems. Technical report, Alliance of Automobile Manufacturers.

[Alpern and Minardo, 2003] Alpern, M. and Minardo, K. (2003). Developing a car gesture interface for use as a secondary task. In *CHI '03 extended abstracts on Human factors in computing systems*, CHI EA '03, pages 932–933, New York, NY, USA. ACM.

[android.com, 2011] android.com (2011). Hello, mapview. http://developer.android.com/guide/tutorials/views/hello-mapview.html. Acedido em 19/12/2011.

[BrightHub, 2011] BrightHub (2011). How to create screen size independent android applications. http://www.brighthub.com/mobile/google-android/articles/61895.aspx. Acedido em 24/11/2011.

[Calvert and Donahoo, 2008] Calvert, K. L. and Donahoo, M. J. (2008). *TCP/IP Sockets in Java*.

[Chang et al., 2009] Chang, J. C., Lien, A., Lathrop, B., and Hees, H. (2009). Usability evaluation of a volkswagen group in-vehicle speech system. In *Proceedings of the 1st International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, AutomotiveUI '09, New York, NY, USA. ACM.

[de Oliveira Pereira, 2009] de Oliveira Pereira, M. S. (2009). *In-Vehicle Information Systems-related multiple task performance and driver behaviour: Comparison between different age groups*. PhD in human motricity, FACULDADE DE MOTRICIDADE HUMANA – Universidade Técnica de Lisboa.

[DiMarzio, 2008] DiMarzio, J. (2008). *Android a programmer's guide*.

[Ellison and Ellison, 2007] Ellison, D. M. B. and Ellison, N. B. (2007). Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*.

[et al., 2005] et al., P. S. (2005). *Fundamentals of Computer Graphics*.

[facebook.com, 2011a] facebook.com (2011a). Facebook advanced concepts: Dialogs: Oauth dialog. http://developers.facebook.com/docs/reference/dialogs/oauth/. Acedido em 18/12/2011.

[facebook.com, 2011b] facebook.com (2011b). Facebook core concepts: Authentication. http://developers.facebook.com/docs/authentication/. Acedido em 18/12/2011.

[facebook.com, 2011c] facebook.com (2011c). Facebook core concepts: Graph api. http://developers.facebook.com/docs/reference/api/. Acedido em 18/12/2011.

[Galitz, 2002] Galitz, W. O. (2002). *The Essential Guide to User Interface Design*.

[GigaOm, 2010] GigaOm (2010). Social networked cars: The future of connected vehicles? http://gigaom.com/cleantech/social-networked-cars-the-future-of-connected-vehicles/. Acedido em 12/11/2011.

[Google, 2011a] Google (2011a). Android graphics: Canvas and drawable. http://developer.android.com/guide/topics/graphics/2d-graphics.html. Acedido em 17/12/2011.

[Google, 2011b] Google (2011b). Android touch mode. http://developer.android.com/resources/articles/touch-mode.html. Acedido em 17/12/2011.

[Google, 2011c] Google (2011c). What is android? http://developer.android.com/guide/basics/what-is-android.html. Acedido em 17/12/2011.

[google.com, 2011] google.com (2011). Bem-vindo ao google maps. http://support.google.com/maps/bin/answer.py?hl=pt&topic=1687350&answer=144352. Acedido em 19/12/2011.

[Green, 1999] Green, P. (1999). Visual task demands of driver information systems. Technical report, University of Michigan.

[Group, 2011a] Group, N. W. (2011a). Especificação oauth2. Technical report, Network Working Group.

[Group, 2011b] Group, T. P. G. D. (2011b). *PostgreSQL 9.1.2 Documentation*.

[Hall, 2009] Hall, B. (2009). *Beej's Guide to Network Programming Using Internet Sockets*.

[Hua and Ng, 2010] Hua, Z. and Ng, W. L. (2010). Speech recognition interface design for in-vehicle system. In *Proceedings of the Second International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, AutomotiveUI 2010.

[json.org, 2011] json.org (2011). Introducing json. http://www.json.org/. Acedido em 18/12/2011.

[Kern and Schmidt, 2009] Kern, D. and Schmidt, A. (2009). Design space for driver-based automotive user interfaces. In *Proceedings of the 1st International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, AutomotiveUI '09, pages 3–10, New York, NY, USA. ACM.

[Knudsen, 1999] Knudsen, J. (1999). *Java 2D Graphics*.

[Kumar and Kim, 2005] Kumar, M. and Kim, T. (2005). Dynamic speedometer: dashboard redesign to discourage drivers from speeding. In *Speeding, Conference on Human Factors in Computing Systems CHI '05 extended abstracts on Human factors in computing system*, pages 1573–1576. ACM Press.

[Liang, 2006] Liang, P. A. (2006). Social networking in vehicles. Master's thesis, Massachusetts lnstitute of Technology.

[Martins, 2009] Martins, F. M. (2009). *JAVA6 e Programação Orientada pelos Objectos*.

[McLaughlin, 2000] McLaughlin, B. (2000). *Java and XML*.

[Meschtscherjakov et al., 2009] Meschtscherjakov, A., Wilfinger, D., Scherndl, T., and Tscheligi, M. (2009). Acceptance of future persuasive in-car interfaces towards a more economic driving behaviour. In *Proceedings of the 1st International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, AutomotiveUI '09, pages 81–88, New York, NY, USA. ACM.

[Murphy, 2010] Murphy, M. L. (2010). *Beginning Android 2*.

[Norman, 2002] Norman, D. A. (2002). *The Design of everyday things*.

[Oracle, 2011a] Oracle (2011a). The java tutorials: A closer look at the paint mechanism. http://docs.oracle.com/javase/tutorial/uiswing/painting/closer.html. Acedido em 15/12/2011.

[Oracle, 2011b] Oracle (2011b). The java tutorials: About the java technology. http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html. Acedido em 15/12/2011.

[Oracle, 2011c] Oracle (2011c). The java tutorials: How to make frames (main windows). http://docs.oracle.com/javase/tutorial/uiswing/components/frame.html. Acedido em 15/12/2011.

[Oracle, 2011d] Oracle (2011d). The java tutorials: How to use panels. http://docs.oracle.com/javase/tutorial/uiswing/components/panel.html. Acedido em 15/12/2011.

[Oracle, 2011e] Oracle (2011e). The java tutorials: Performing custom painting. http://docs.oracle.com/javase/tutorial/uiswing/painting/step2.html. Acedido em 15/12/2011.

[Oracle, 2011f] Oracle (2011f). The java tutorials: What is swing? http://docs.oracle.com/javase/tutorial/ui/overview/intro.html. Acedido em 15/12/2011.

[Oracle, 2011g] Oracle (2011g). Painting in awt and swing. http://www.oracle.com/technetwork/java/painting-140037.html#awt. Acedido em 15/12/2011.

[Psycopg, 2011a] Psycopg (2011a). Psycopg - basic module usage. http://initd.org/psycopg/docs/usage.html. Acedido em 19/12/2011.

[Psycopg, 2011b] Psycopg (2011b). Psycopg - postgresql database adapter for python. http://initd.org/psycopg/docs/. Acedido em 19/12/2011.

[Ralph and Wand, 2009] Ralph, P. and Wand, Y. (2009). A proposal for a formal definition of the design concept. In Lyytinen, K., Loucopoulos, P., Mylopoulos, J., Robinson, B., Aalst, W., Mylopoulos, J., Rosemann, M., Shaw, M. J., and Szyperski, C., editors, *Design Requirements Engineering: A Ten-Year Perspective*, volume 14 of *Lecture Notes in Business Information Processing*, pages 103–136. Springer Berlin Heidelberg.

[Report, 2011] Report, A. S. M. (2011). Facebook usage: Factors and analysis. Acedido em 08/12/2011.

[Reports, 2011] Reports, C. (2011). Toyota to create a social network for its cars and drivers. http://news.consumerreports.org/cars/2011/05/toyota-to-create-a-social-network-for-its-cars-and-drivers.html. Acedido em 12/11/2011.

[Rockwell, 1988] Rockwell, T. H. (1988). Spare visual capacity in driving-revisited: New empirical results for an old idea. In *Vision in vehicles II. Proceedings of the second international conference on vision in vehicles*, pages 317–324. Elsevier.

[Salvucci, 2005] Salvucci, D. D. (2005). Distract-r: Rapid prototyping and evaluation of in-vehicle interfaces. In *In Human Factors in Computing Systems: CHI 2005 Conference Proceedings*, pages 581–589. ACM Press.

[Schmidt et al., 2010] Schmidt, A., Dey, A. K., Kun, A. L., and Spiessl, W. (2010). Automotive user interfaces: human computer interaction in the car. In *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*, CHI EA '10, pages 3177–3180, New York, NY, USA. ACM.

[Stats, 2011] Stats, I. W. (2011). Facebook users in the world. http://www.internetworldstats.com/facebook.htm. Acedido em 08/12/2011.

[Sudarshan et al., 2010] Sudarshan, P. K., R., B. I., and Ray, A. (2010). Joy of use in automotive touch screen ui for the driver. In *Proceedings of the Second International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, AutomotiveUI 2010.

[Tecmundo, 2009] Tecmundo (2009). O que é o xml? http://www.tecmundo.com.br/1762-o-que-e-xml-.htm. Acedido em 14/12/2011.

[Times, 2010] Times, T. N. Y. (2010). Social networking for cars. http://wheels.blogs.nytimes.com/2010/07/20/social-networking-for-cars/. Acedido em 12/11/2011.

[W3C, 2008] W3C (2008). Extensible markup language (xml) 1.0 (fifth edition). Technical report, W3C.

[webopedia, 2011] webopedia (2011). markup language. http://www.webopedia.com/TERM/M/markup_language.html. Acedido em 14/12/2011.

Anexos

Anexo A

Documento XML do painel do veículo Gecko Merula

```
<panel>
        <name>gecko</name>
                <screen>
                        <width>1280</width>
                        <height>798</height>
                </screen>

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

                <pointerGauge>
                        <id>taquimetro</id>
                        <file>contarotacoes.png</file>
                        <x>11</x>
                        <y>214</y>

                        <pointer>
                                <id>ponteiroodometro</id>
                                <color>red</color>
                                <p0x>205</p0x>
                                <p0y>398</p0y>
```

```
                <p1x>64</p1x>
                <p1y>491</p1y>
                <p2x>320</p2x>
                <p2y>379</p2y>
                <vmin>0</vmin>
                <vmax>5</vmax>
        </pointer>
</pointerGauge>

<levelGauge>
        <id>nivelbateria</id>
        <file>bateria2.png</file>
        <x>1026</x>
        <y>170</y>

        <levelGaugeElement>
                <id>elementobateria</id>
                <elements>4</elements>
                <file>barra_bateria.png</file>
                <x>1040</x>
                <y>173</y>
                <length>33</length>
                <height>90</height>
        </levelGaugeElement>
</levelGauge>

<pointerGauge>
        <id>temperatura</id>
        <file>temperature.jpg</file>
        <x>933</x>
        <y>328</y>

        <pointer>
                <id>ponteirotemperatura</id>
                <color>red</color>
                <p0x>1109</p0x>
                <p0y>469</p0y>
                <p1x>994</p1x>
                <p1y>446</p1y>
                <p2x>1203</p2x>
                <p2y>420</p2y>
                <vmin>50</vmin>
                <vmax>150</vmax>
```

```
        </pointer>
</pointerGauge>

<indicator>
        <id>indicadortravao</id>
        <imageActive>travao.jpg</imageActive>
        <imageInactive>travaoinact.jpg</imageInactive>
        <x>634</x>
        <y>80</y>
</indicator>

<label>
        <id>velocimetro</id>
        <refreshable>true</refreshable>
        <x>525</x>
        <y>187</y>
        <color>white</color>
        <fontype>Serif</fontype>
        <fontsize>57</fontsize>
        <default>velocidade</default>
</label>

<label>
        <id>kmh</id>
        <refreshable>false</refreshable>
        <x>571</x>
        <y>188</y>
        <color>white</color>
        <fontype>Serif</fontype>
        <fontsize>57</fontsize>
        <default>Km/h</default>
</label>

<indicator>
        <id>indicadorbateria</id>
        <imageActive>bateria.jpg</imageActive>
        <imageInactive>bateriainact.jpg</imageInactive>
        <x>575</x>
        <y>286</y>
</indicator>

<indicator>
        <id>indicadorluzinterior</id>
```

```
        <imageActive>interior.jpg</imageActive>
        <imageInactive>interiorinact.jpg</imageInactive>
        <x>422</x>
        <y>405</y>
</indicator>

<indicator>
        <id>indicadorluzpresenca</id>
        <imageActive>minimos.jpg</imageActive>
        <imageInactive>minimosinact.jpg</imageInactive>
        <x>514</x>
        <y>401</y>
</indicator>

<indicator>
        <id>indicadorluzcruzamento</id>
        <imageActive>mediosm.jpg</imageActive>
        <imageInactive>mediosminact.jpg</imageInactive>
        <x>624</x>
        <y>392</y>
</indicator>

<indicator>
        <id>indicadorestrada</id>
        <imageActive>maximosm.jpg</imageActive>
        <imageInactive>maximosminact.jpg</imageInactive>
        <x>735</x>
        <y>398</y>
</indicator>

<indicator>
        <id>indicadorluznevoeiro</id>
        <imageActive>nevoeiro.jpg</imageActive>
        <imageInactive>nevoeiroinact.jpg</imageInactive>
        <x>832</x>
        <y>392</y>
</indicator>

<label>
        <id>odometro</id>
        <refreshable>true</refreshable>
        <x>583</x>
        <y>530</y>
```

```
                <color>white</color>
                <fontype>Serif</fontype>
                <fontsize>40</fontsize>
                <default>quilometros</default>
        </label>

        <label>
                <id>km</id>
                <refreshable>false</refreshable>
                <x>706</x>
                <y>526</y>
                <color>white</color>
                <fontype>Serif</fontype>
                <fontsize>40</fontsize>
                <default>Km</default>
        </label>
</panel>
```