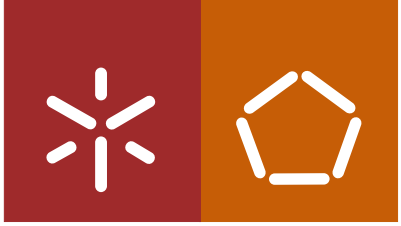
Universidade do Minho
Escola de Engenharia

Miguel Gomes Zenha

Sistema de Monitorização de Transportes Urbanos Utilizando SNMP

julho de 2019

Sistema de Monitorização de Transportes Urbanos Utilizando SNMP
Miguel Gomes Zenha
UMinho | 2019



Universidade do Minho
Escola de Engenharia

Miguel Gomes Zenha

Sistema de Monitorização de Transportes Urbanos Utilizando SNMP

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho efetuado sob a orientação do
Professor Doutor Bruno Alexandre Fernandes Dias

julho de 2019

AGRADECIMENTOS

Agradeço ao meu orientador, Bruno Dias, pelo apoio, dedicação, confiança e disponibilidade durante toda a realização da dissertação.

Aos meus pais e irmãos por me terem acompanhado durante toda a minha vida e pelo esforço em proporcionarem as condições para que eu me encontre hoje aqui.

À Joana, namorada e fiel amiga, por ser um dos pilares da minha vida e por me incentivar a crescer cada vez mais.

Ao Francisco, amigo e eterno mentor, por fazer despertar o meu gosto pela informática e por me proporcionar as bases necessárias ao meu sucesso pessoal e profissional.

Ao Professor José Bernardo Barros, por ter sido professor fora das aulas e por demonstrar uma disponibilidade e atenção incrível para com os seus alunos.

Por fim, agradeço aos professores da Universidade do Minho, nomeadamente do Departamento de Informática, por todas as oportunidades e conhecimentos que proporcionam aos seus alunos

ABSTRACT

Public transportation companies have been searching for new technologies and applications with the goal of improving the service available to its passengers. In this regard, the operators have invested in the increase of the passenger safety and comfort, diversification of available features, the addition of new destinations or stops, and mostly on the efficiency and compliance of timetables, among others. When it comes to information technologies, one of the features which has been given more attention is the information system and the monitoring available to users. While a substantial portion of the information kept by these systems is infrequently updated, which can even be called fixed nature information, the data used in the monitoring systems must be updated as often as possible.

However, enhancing the service quality and delivering real time information to the users usually implies huge costs. When planning and building systems and applications to monitor public transport services at a urban or regional scale, the decision on which infrastructure a company will have to invest in order to obtain and process real time data is crucial. Besides, a wrong choice regarding the associated technologies and systems that assemble the service, combined with the involved implementation costs can seriously compromise the company sustainability in the market.

The present work starts with an in-depth study of the different solutions and technologies that we encounter on today's market, critically analyzing them. From that study, the challenge of conceiving an integrated information management system arose. It provides management tools, optimization and administration of the public transport services based on normalized technologies, open-source and usage costs free.

The designed solution, modular and highly scalable, consists in a system entirely based on the Simple Network Management Protocol (SNMP) which combines applications with the ability to attain real time data from the vehicles and therefore provides differentiated information to the various users (passengers, managers, drivers).

After implementation and testing the complete prototype applications, the proposed architecture has been proven to be effectively functional. Being a modular solution based on a normalized, open source and free of copyright rights technology, it is consequently expected to be also effective in terms of eventual implementation costs and adoption in the real world.

Keywords: SNMP, Automatic Vehicle Localization, Real Time Data, Public Transports.

RESUMO

As empresas do sector dos transportes públicos têm procurado introduzir novas tecnologias e aplicações com o objetivo de melhorar o serviço disponibilizado aos passageiros. Neste sentido, os operadores deste tipo de serviço têm investido no aumento da segurança e conforto do passageiro, na diversificação das funcionalidades disponibilizadas, no acrescento de novos destinos ou paragens e na eficácia do cumprimento de horários planeados, entre outros. Do lado das tecnologias de informação, uma das funcionalidades que mais tem recebido atenção é o sistema de informação e monitorização acessível aos utilizadores. Enquanto uma parte substancial da informação mantida por estes sistemas é atualizada pouco frequentemente, podendo ser, inclusive, denominada de informação de carácter fixo, os dados utilizados no sistema de monitorização devem ser atualizados o mais frequentemente possível. Mas, o objetivo de aumentar a qualidade do serviço de informação prestado melhorando a qualidade da monitorização do sistema, implica, em geral, custos elevados. No planeamento e construção de sistemas e aplicações de informação para monitorização de serviços de transporte de passageiros numa escala urbana ou regional, a escolha do tipo de infra-estrutura que torna possível obter, processar e utilizar os dados em tempo real, ou com frequência funcionalmente útil, é, assim, fulcral. Uma má escolha das tecnologias associadas e dos sistemas que integram o serviço, podem tornar os custos inerentes à sua implementação no terreno insustentáveis e podem comprometer seriamente a sustentabilidade das empresas do ramo.

O presente trabalho começa com um estudo aprofundado das várias soluções e tecnologias já existentes no mercado, analisando-as criticamente. Desse estudo resultou o desafio de conceber uma proposta dum sistema aplicativo integrado de gestão de informação que disponibiliza ferramentas para a gestão, optimização e administração do serviço de transportes públicos com base em tecnologias normalizadas, abertas e de sem custo de utilização.

A solução desenhada, que se pretendeu modular e escalável, consiste num sistema integrado baseado inteiramente no protocolo *Simple Network Management Protocol* (SNMP) e que

combina aplicações informáticas com a capacidade de obter dados em tempo real dos veículos e também disponibilizar informação diferenciada aos diversos intervenientes do sistema (passageiros, gestores, motoristas).

Após a implementação e teste dum sistema completo de aplicações protótipo, fica provada a efetividade funcional da arquitetura proposta. Sendo esta solução modular e baseada numa tecnologia normalizada e aberta e de utilização livre de direitos de autor, espera-se, em consequência, que também seja eficaz no que concerne a eventuais custos de implementação e adoção no mundo real.

Palavras-chave: SNMP, Localização automática de veículos, Informação em tempo real, Transportes de passageiros.

ACRÓNIMOS

ARP	Address Resolution Protocol
AVL	Automatic Vehicle Location
API	Application Programming Interface
DSRC	Dedicated Short-Range Communications
ECU	Engine Control Unit
ETA	Estimated Time of Arrival
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile Communications
IP	Internet Protocol
ITS	Intelligent Transportation System
LTE	Long Term Evolution
MIB	Management Information Base
NFC	Near Field Communication
NMS	Network Management Station
OID	Object Identifier
PDU	Protocol Data Unit
RF	Radio-Frequency
RFID	Radio-Frequency Identification
SQL	Structured Query Language
SAEIP	Sistema de Apoio à Exploração e Informação ao Passageiro
SNMP	Simple Network Management Protocol
TETRA	Terrestrial Turked Radio
UDP	User Datagram Protocol
UHF	Ultra High Frequency
UMTS	Universal Mobile Telecommunications System

ÍNDICE

LISTA DE FIGURAS

---

1. INTRODUÇÃO

Na atualidade, duma forma global, a obtenção, tratamento e gestão de informação em tempo real (ou tempo funcionalmente útil para os vários tipos de utilizadores das aplicações) desempenha cada vez mais um papel crucial no sucesso de uma organização.

Ao mesmo tempo, existe uma crescente necessidade das empresas de transportes públicos fornecerem aos seus clientes uma gama diversificada de serviços e em que a informação esteja disponível em tempo útil e seja de qualidade, de forma a responder às crescentes exigências e necessidades duma sociedade que vive em aglomerados urbanos cada vez maiores.

A falta da informação em tempo útil ou a sua reduzida qualidade funcional pode levar a atrasos nas tomadas de decisão ou que as decisões tomadas estejam baseadas em dados errados ou desatualizados, diminuindo assim o valor funcional das aplicações informáticas utilizadas. No caso do sector dos transportes públicos, o maior custo será o do passageiro deixar de utilizar os transportes devido a falhas ou atrasos nos serviços ou incongruências na informação disponibilizada publicamente. Estes erros podem ser atenuados relevantemente se os utilizadores tiverem acesso a aplicações informáticas que tirem partido dum conhecimento atualizado do estado dos serviços, tanto os planeados como os efectivos.

Do ponto de vista do passageiro, a informação em tempo real é uma ferramenta que aumenta a qualidade do serviço prestado. Com estes sistemas, o passageiro sabe em tempo útil informações como a hora a que chega um veículo, se o serviço está atrasado ou se, por algum motivo previsto ou imprevisto, o serviço não se efetua. Este tipo de informações permitem ao passageiro tomar decisões acerca do meio de transporte a utilizar e decidir com base em informação útil e atempada qual a melhor forma de chegar ao seu destino.

Do ponto de vista do negócio, a informação em tempo real permite a aquisição de dados fundamentais de modo efetuar-se uma gestão mais rigorosa de todo o sistema de transportes, quer ao nível do planeamento quer ao nível da monitorização do mesmo. Alguns dos dados importantes que podem melhorar os serviços prestados são, por exemplo, a localização dos veículos, a quantidade de passageiros que entram e saem de um veículo no espaço e no tempo ou dados técnicos sobre o estado geral do veículo. Por outro lado, a integração de dados de sistemas de bilhética e a implementação dum centro de controlo através duma aplicação informática distribuída são algumas das funcionalidades que permitem mais valias num sistema integrado de transportes públicos.

Atualmente já podemos encontrar vários tipos de sistemas de informação dedicados à apresentação de informação sobre redes de transportes urbanos, incluindo mapas de cobertura, carreiras, paragens, horários, preçários e até a sua monitorização em tempo real onde podem ser apresentados diversos dados relevantes: a posição dos veículos num mapa topográfico ou mapa lógico virtual, estimativas de tempos de passagem, número de passageiros e/ou lugares livres, entre outros.

No entanto, alguns dos sistemas mais conhecidos, como o Google Maps, por exemplo, ou não utilizam tecnologias universais normalizadas para representação da informação e interação com os servidores que prestam o serviço, ou são sistemas comerciais completamente fechados. Estes fatores contribuem para o aumento dos custos para as empresas na implementação e utilização de sistemas integrados de gestão de transportes públicos. Para os passageiros, o principal problema é a falta de homogeneidade das aplicações disponibilizadas pelas várias empresas de transporte num mesmo contexto urbano ou em contextos urbanos distintos (quando, por exemplo, visitam destinos urbanos diferentes).

O principal objetivo do projeto agora apresentado foi o desenvolvimento de um sistema de monitorização e apresentação da informação de serviços de transportes urbanos utilizando tecnologias normalizadas universais e gratuitas, como é o caso do protocolo comunicacional SNMP em que as bases de dados são representadas em *Management Information Bases* (MIBs).

A primeira etapa do projecto passou por investigar extensivamente as soluções informáticas que já tenham sido implementadas num contexto urbano e descobrir as suas funcionalidades e a forma como prestam os serviços (arquitetura funcional, protocolos comunicacionais, tecnologias de interface, linguagens de representação dos dados, etc.) com o objetivo de chegar a conclusões sobre as várias potencialidades e, sobretudo, sobre as suas principais limitações funcionais, técnicas ou de outro tipo.

Seguidamente foram estudados os vários mecanismos de localização de veículos existentes no mercado, incluindo aqueles que são utilizados pelas aplicações já existentes de monitorização de transportes públicos. Além disso, foram analisadas as tecnologias de comunicação sem fios que permitem a possibilidade de adquirir dados e de os transmitir em tempo real para um sistema com poder computacional, onde são processados para originar informação útil a vários utilizadores com diferentes necessidades. Estes estudos ajudaram também no apuramento de quais seriam as vantagens de utilizar, no contexto destas aplicações, o SNMP e as MIBs como tecnologias para a distribuição e gestão das base de dados.

A arquitetura desenvolvida é constituída por três sistemas distintos, mas que trabalham em conjunto: um sistema de administração; um sistema utilizado nos veículos de transporte e uma aplicação utilizada pelos passageiros. Todos estes sistemas utilizam como protocolo de comunicação normalizado o SNMP e toda a informação necessária para caracterizar o sistema (paragens, carreiras, horários, preços, áreas de passes, mapas, etc.) deverá ser representada e integrada através de MIBs.

O sistema de administração permite a integração e alteração de toda a informação necessária para caracterizar um sistema de transportes urbanos, enquanto que o sistema nos veículos de transporte público gere os dados respeitantes à localização em tempo real, entre outros. A aplicação disponibilizada aos passageiros utiliza toda a informação disponível nos sistemas de forma a que estes possam saber o estado atual do serviço, quais as carreiras que melhor o podem servir, dependendo da sua posição atual e qual o destino desejado, informações úteis

sobre duração estimada da viajem, preço do bilhete, etc. Por razões óbvias de maximização da sua funcionalidade, foi decidido que esta aplicação para os passageiros seria implementada num sistema computacional móvel com sistema operativo Android.

De realçar que a arquitetura desenhada não depende de nenhuma instanciação em particular, isto é, não é dependente de nenhum contexto urbano específico. Parte do sistema de administração permite a inclusão da informação necessária para um ou vários contextos urbanos. Ou seja, o sistema pode ser utilizado por vários operadores duma ou de várias cidades simultaneamente.

Por outro lado, sendo que a arquitetura definida é totalmente modular, eventuais problemas de escalabilidade não se colocam, até porque este tipo de sistema não é conhecido por ser um sistema de informação em que a haja quantidades massivas de dados a serem geridos ou uma quantidade massiva de utilizadores em acesso simultâneo. Ainda assim, eventuais problemas no tamanho das bases de dados podem ser resolvidos com tecnologias próprias de bases de dados que já são amplamente conhecidas e incluídas nas implementações das atuais deste tipo de soluções.

Depois da implementação dos protótipos dos três sistemas, foi incorporada toda a informação oficial e atual do sistema de transportes urbanos da cidade de Braga. A utilização desta informação, cedida pela empresa que atualmente gere o sistema de transporte urbanos de Braga, foi oficialmente autorizada pelos serviços próprios da Câmara Municipal de Braga.

Apesar da cidade de Braga não ser um contexto urbano de larga escala, a sua dimensão já será suficiente para que se pudesse testar e validar com sucesso os principais aspetos funcionais da arquitetura definida e dos sistemas implementados.

2. SOLUÇÕES ATUAIS

Hoje em dia, a utilização dos transportes públicos pode tornar-se um desafio arriscado, sobretudo em contextos urbanos das grandes cidades ou em que os serviços sejam caóticos e não disponibilizem informação atualizada, em tempo útil e através de meios facilmente acessíveis ao utilizador comum em qualquer altura.

Existem diversas aplicações de transportes públicos que visam ajudar a desvendar mapas confusos e horários complexos de transportes públicos por todo o mundo (autocarros, metros, etc.). Na presente secção é feita uma análise crítica daquelas que dominam o mercado atual.

2.1 Google Maps

Devido à sua elevada visibilidade e utilização, o Google Maps [1] é uma aplicação que dispensa grandes introduções. Apesar da aplicação não ter sido criada inicialmente para fornecer informações acerca dos transportes públicos, essa funcionalidade foi integrada em 2013, suportando praticamente todas as cidades do mundo. Inclusivamente, em 2017 o Google Maps passou a fornecer informações em tempo real acerca dos horários dos meios de transporte, mas apenas para a Tailândia. A aplicação tem disponível uma versão móvel e uma versão *desktop* e ambas permitem pesquisar informações, caso existam, sobre os serviços de transporte públicos de qualquer cidade do mundo. Ou seja, a grande mais valia desta aplicação é a possibilidade de utilização a uma escala global através dum interface em vários tipos de sistemas informáticos.

Para utilizadores que procuram informações sobre serviços de transporte públicos em várias cidades do mundo, o Google Maps será certamente a melhor aposta, pelo menos a aposta mais segura. Enquanto que as outras aplicações se especializam em cidades particulares, o Google é uma solução genérica. Isto pode ser bom e mau ao mesmo tempo, uma vez que os utilizadores têm muita mais informação na ponta dos dedos, mas, no geral, a qualidade dessa informação pode não ser reduzida.

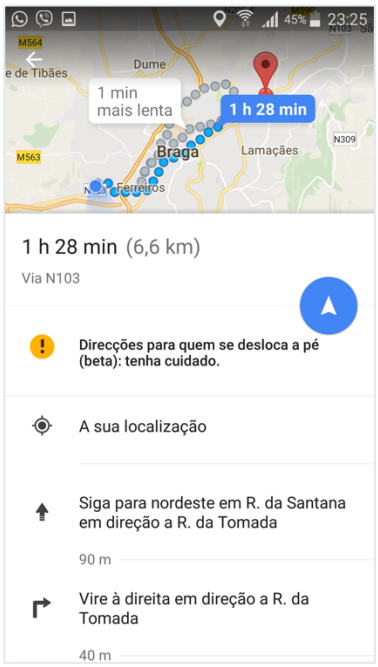


Figura 1 - Exemplo de Funcionalidades do Google Maps

Depois de definir um destino, o utilizador pode escolher entre conduzir, apanhar um autocarro, metro, comboio, bicicleta ou simplesmente ir a pé.

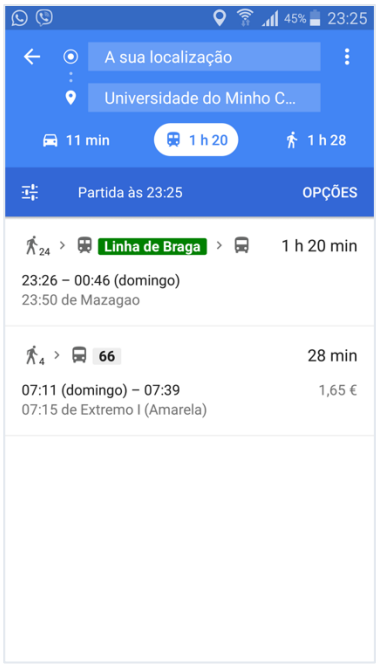


Figura 2 – Exemplo de Funcionalidades do Google Maps

O utilizador pode até configurar o tempo a que quer sair e chegar, para que possa ver as informações mais precisas para essa viagem. No entanto, os horários dos transportes são fixos e não existe informação em tempo real, exceto na Tailândia, mas apresenta o custo de cada transporte, o que nenhuma outra aplicação, das que foram estudadas, faz.

A aplicação oferece também a funcionalidade que permite ao utilizador visualizar onde vão existir atrasos da sua viagem a partir do controlo de tráfego da aplicação Google Traffic. Basicamente, se é versatilidade que os utilizadores procuram, o Google Maps é a aplicação a ser escolhida. Não oferece tantas opções e funcionalidades como as outras aplicações, mas cobre praticamente todas as cidades no mundo, incluindo aquelas que são suportadas pelas restantes aplicações do mesmo ramo.

Tendo em conta que o Google Maps ou o Apple Maps estão atualmente presentes em praticamente todos os telemóveis, surge então a questão: porque não utilizar apenas o Google Maps ao invés das outras aplicações?

Porque existem algumas funcionalidades relevantes que ainda não estão incluídas. Por exemplo, nenhuma das duas aplicações tem a opção de saber quais as "paragens mais próximas" da posição atual do utilizador. Outra razão poderá ser a demora na atualização dos mapas das carreiras, dos horários e dos preçários, sempre que existem modificações oficiais. As aplicações dedicadas exclusivamente ao apoio dos serviços de transporte públicos têm estas e outras funcionalidades incluídas, como por exemplo, a posição em tempo real dos veículos de transporte público que fazem parte da rede e uma estimativa muito mais precisa do tempo de chegada à próxima paragem. Além disso, estas aplicações dedicadas podem integrar também serviços menos clássicos, como por exemplo, o Bikeshare ou o Uber.

2.2 Moovit

A Moovit [2] é a aplicação dedicada de transportes públicos mais utilizada no mundo, com mais de 45 milhões de utilizadores distribuídos por de 1200 cidades, 67 países e 44 linguagens diferentes. É uma aplicação de trânsito pública com um serviço de mapeamento que oferece planeamento de viagens, chegadas e partidas em tempo real, horários atualizados, mapas locais de estações, serviço de alertas e conselhos que podem afetar uma viagem corrente.

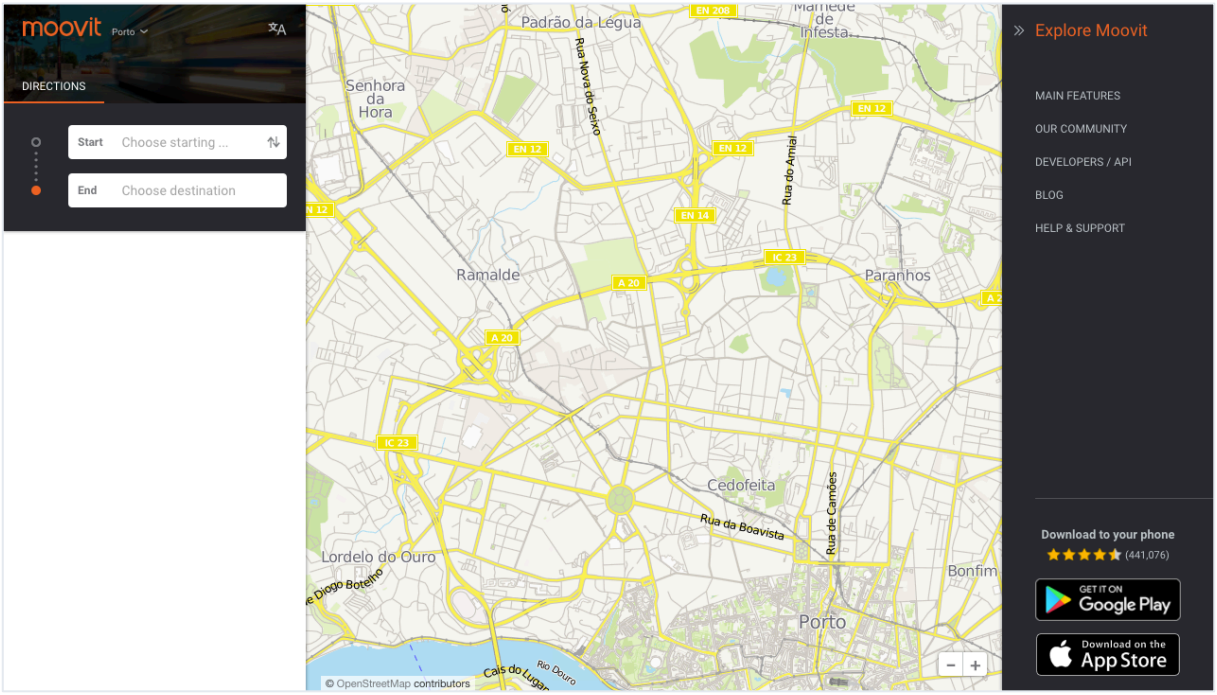


Figura 3 – Versão Desktop da Aplicação Moovit

Esta aplicação suporta os sistemas operativos Android, IOS, dispositivos Windows, tendo também uma versão web. A aplicação oferece informação pública acerca do trânsito em tempo real e navegação nos modos de trânsito, que incluem autocarros, ferry, metros, comboios e elétricos. Os utilizadores podem aceder a um mapa em tempo real e ver as paragens e as estações mais próximas, assim como planearem viagens com dados reais, tendo como base a sua geo-localização atual.

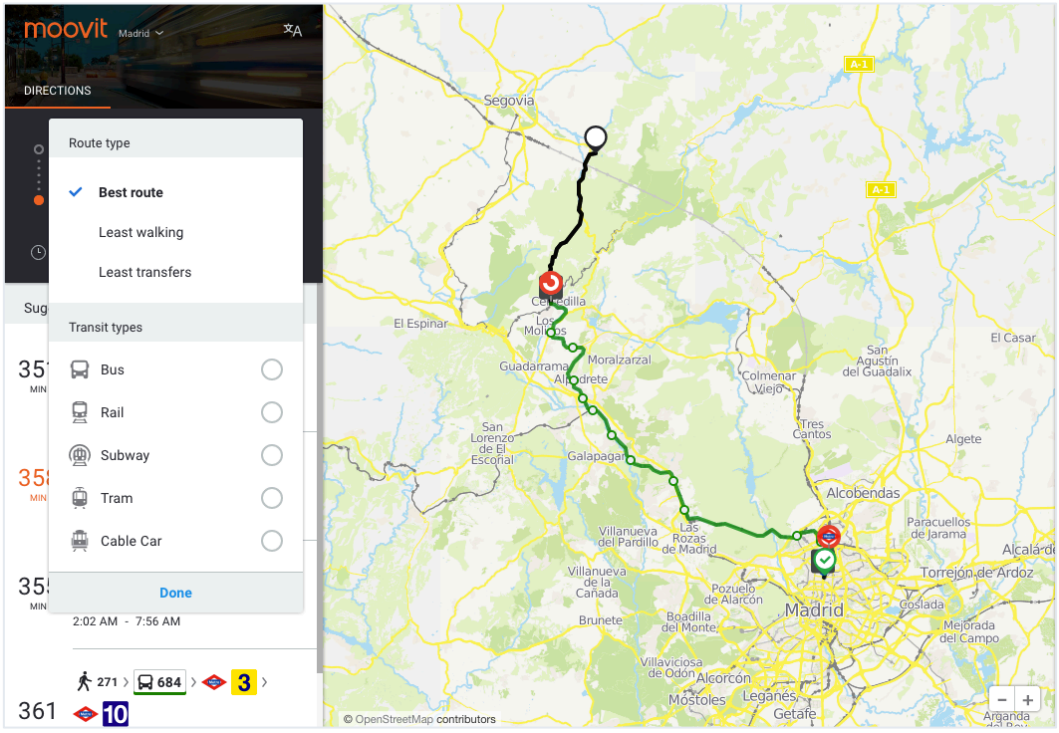


Figura 4 – Opções disponíveis para o Tipo de Rota

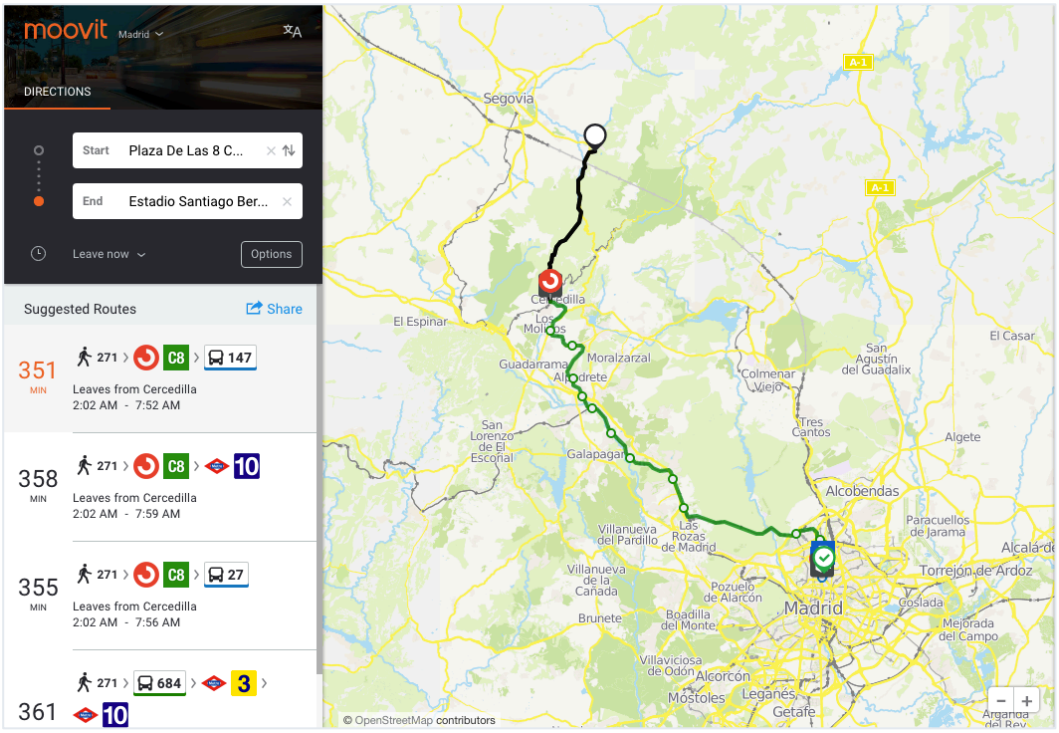


Figura 5 – Apresentação das diferentes rotas no Moovit

Em 2014 a Moovit tornou-se a primeira aplicação a fornecer dados de trânsito para áreas nas quais não existem dados oficialmente disponíveis (ou onde as empresas de transportes públicos não fornecem os dados ao público). Isso é conseguido a partir da criação de uma comunidade de editores, em que contribuintes voluntários criam horários e dados de mapas e os inserem na aplicação.

A aplicação utiliza os mapas do OpenStreetMap, um projecto colaborativo que permite criar mapas do mundo, grátis e editáveis, baseados na *Open Database License*. É a segunda API de mapas mais utilizada neste tipo de aplicações, pertencendo o primeiro lugar à API do Google Maps.

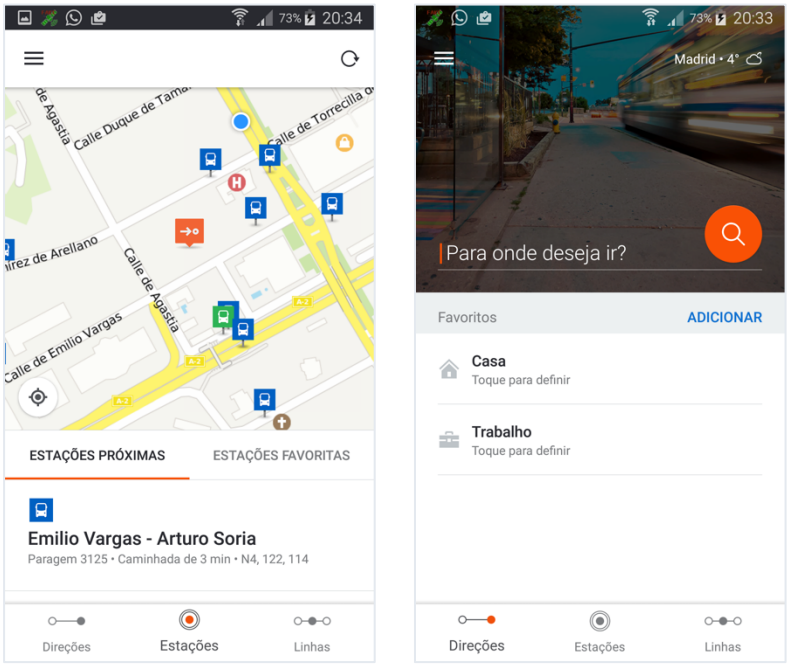


Figura 6 – Exemplos de funcionalidades da Versão Móvel da Moovit

Ao ativar a funcionalidade "*Live Directions*", os utilizadores podem transmitir passiva e anonimamente os seus dados de localização e velocidade para a Moovit. De seguida, a aplicação recolhe esses dados, provenientes de *crowdsourcing* e cruza-os com os cronogramas de trânsito público de forma a melhorar os resultados dos planos de viagem com base as condições atuais. Por fim, compartilha esses dados com a comunidade de utilizadores. Além de partilharem os dados de forma passiva, os utilizadores podem enviar relatórios que podem incluir razões dos atrasos, sobrelotação, níveis de satisfação com o condutor do autocarro onde se encontram, entre outros.

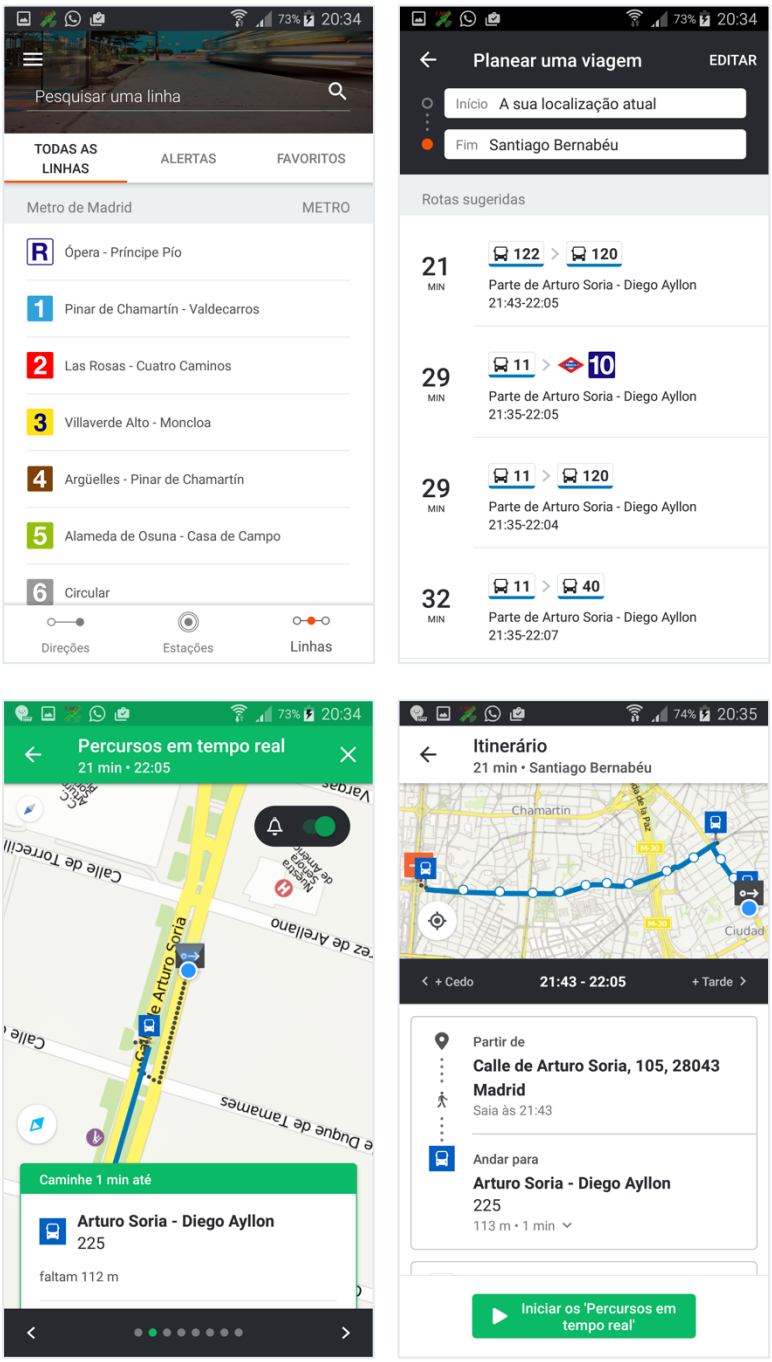


Figura 7 – Exemplos de funcionalidades da Versão Móvel da Moovit

A funcionalidade "*Live Directions*" fornece também ao utilizador um "*companheiro de trânsito*", uma vez que dá instruções passo-a-passo sobre um dado percurso, incluindo notificações "*Saia aqui!*", para que o utilizador saiba onde e quando é que tem que sair. A aplicação fornece também um guia de linhas, que contém informações sobre uma linha específica (comboio, autocarro, etc.), como o estado da linha que o utilizador frequenta regularmente, serviço de alertas e chegadas em tempo real do transporte. Fornece também um guia de estações, onde é apresentado um mapa com as estações mais próximas e as

respectivas linhas activas, assim como os próximos autocarros que vão passar nessa estação e as informações de ETA em tempo real.

Devido ao elevado número de utilizadores e países suportados, esta aplicação destaca-se como uma aplicação de transportes públicos enormemente poderosa. Contudo, por ser a aplicação que suporta mais cidades não significa que seja a melhor.

Tendo em conta que o seu mecanismo é baseado em *crowdsourcing*, que tal como todos os mecanismos tem vantagens e desvantagens, os dados em tempo real da aplicação são bastante fiáveis. No entanto, à medida que se passa de cidades com grande densidade populacional para cidades com menos, a eficácia da aplicação vai diminuindo, uma vez que o número de utilizadores é mais baixo e não são suficientes para manter a base de dados com informação relevante em tempo útil.

2.3 Trafi

A Trafi [3] é uma aplicação de transportes públicos desenhada com o propósito de permitir aos seus utilizadores planear viagens utilizando algoritmos complexos de inteligência artificial. A aplicação encontra-se atualmente disponível em sete países: Turquia, Lituânia, Letónia, Estónia, Rússia, Tailândia e Brasil.

A aplicação utiliza dados em tempo real para desenvolver padrões locais de pesquisa de rotas específicas para uma dada área. Para disponibilizar uma experiência mais correta, parte dos dados utilizados no sistema são também fornecidos pelos próprios utilizadores.

A aplicação permite pesquisar rotas de autocarro, metro e comboios, juntamente com os horários e os dados em tempo real da posição dos autocarros e estimativas do tempo de chegada real. Encontra-se disponível uma versão móvel (*Android* e IOS) e uma versão *desktop*.

A Trafi agrega e apresenta diferentes modos de transporte como metro, comboios, autocarros e Bikeshare, dando sugestões de rotas com base na conveniência e velocidade. De facto, a função em tempo real da aplicação, que é suportada por dados GPS, permite ao utilizador

ver como os diferentes meios de transporte se movem num mapa da cidade. Desta forma, a aplicação serve em segundo plano como um fornecedor de soluções para o setor dos transportes, ajudando agências e as autoridades a melhorarem a visibilidade da sua frota e a aumentarem a eficácia dos transportes públicos.

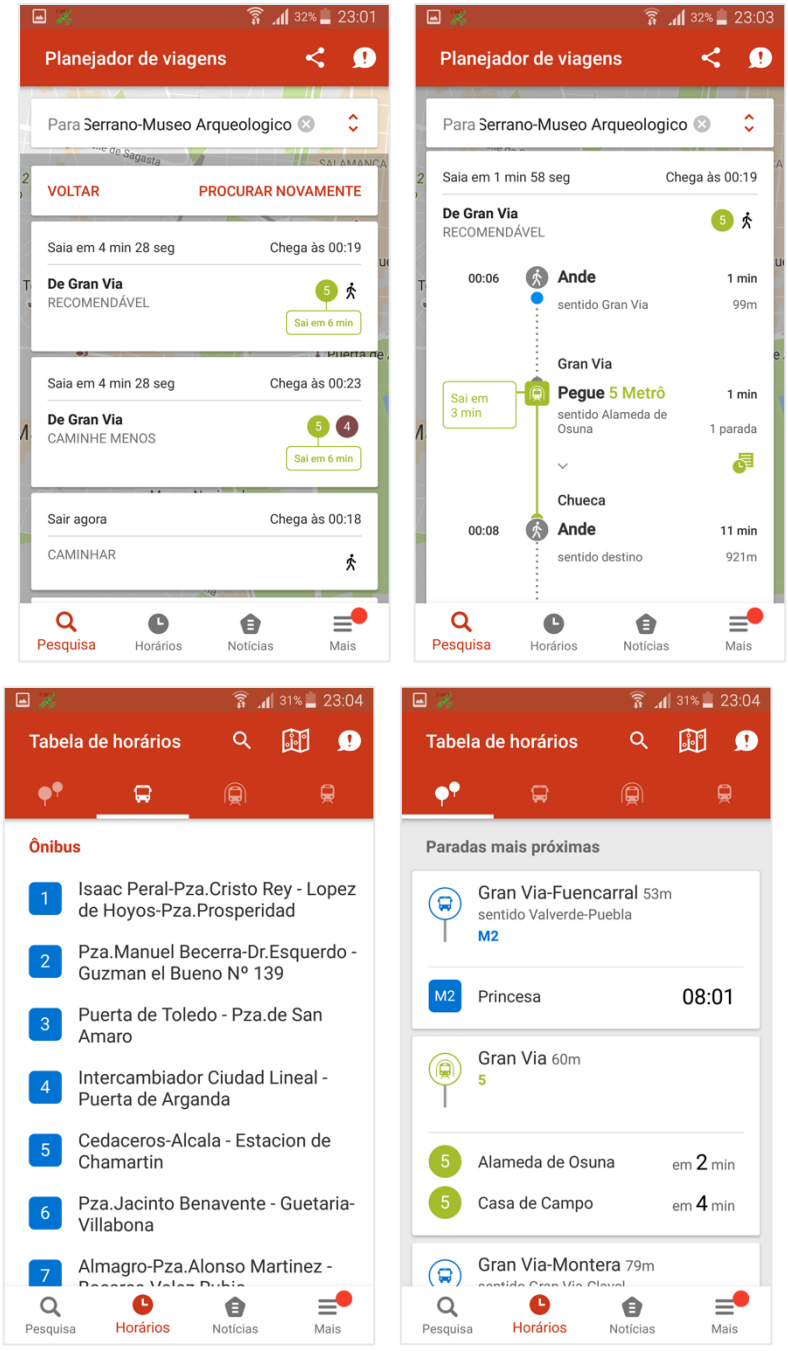


Figura 8 – Exemplos de funcionalidades da Aplicação Trafi

No que diz respeito ao seu funcionamento, a aplicação utiliza uma abordagem mista. Por um lado, utiliza uma combinação de algoritmos, processamento em tempo real de situações de trânsito e relatórios baseados em *crowdsourcing*, de forma a prever durações e tempos de

chegada. Por outro, utiliza tecnologias de *machine learning*, uma vez que compreende padrões locais e fornece resultados diferentes para diferentes cidades e indivíduos. A plataforma permite também aos utilizadores criar conteúdos relacionados com os transportes públicos, que por sua vez são submetidos a filtros, escolhendo assim os aspetos importantes e notificando outros viajantes acerca dos mesmos. A aplicação também está disponível num modo *offline*, permitindo assim planear rotas, mesmo sem rede.

Aquando da utilização da aplicação, o utilizador pode ver a posição onde se encontra e todas as paragens da cidade, assim como a distância a que se encontra das paragens mais próximas e quais os próximos serviços nessas paragens e o horário fixo dos mesmos, assim como o tempo que falta para ocorrerem. Tal como acontece com outras aplicações, é possível adicionar favoritos para que a aplicação aplique as suas técnicas de *machine learning* e o uso desta seja mais rápido. Pesquisando um local, a aplicação fornece os vários meios de transporte e recomenda os melhores e mais rápidos, indicando ao utilizador a paragem a que se deve dirigir e quanto tempo o meio de transporte escolhido demora a chegar a essa paragem.

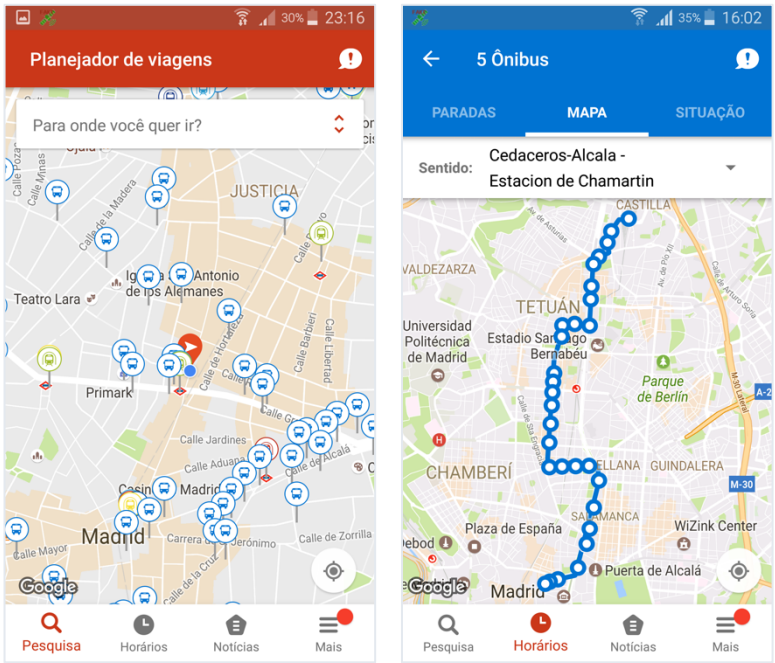


Figura 9 – Exemplos de funcionalidades da Aplicação Trafi

O utilizador tem acesso ao horário completo de todos os meios de transporte, assim como a opção de visualizar cada percurso em grelha ou no mapa da cidade, mostrando todas as paragens do percurso. No caso de uma linha estar interrompida ou a posição do veículo estar

errada na aplicação, o utilizador tem a opção de reportar um problema, assim como adicionar fotos do mesmo, e esse problema é atualizado na aplicação, isto após ser submetido a algoritmos de filtragem de informação. No modo *offline*, a aplicação tem na mesma acesso às coordenadas GPS do telemóvel e diz quais são as paragens mais próximas. Funciona da mesma maneira mas não tem acesso à localização em tempo real dos transportes.

Para a Trafi, o maior desafio encontrado até agora é a disponibilidade dos dados, especialmente em mercados emergentes em países como, por exemplo, a Índia. Muitas cidades no mundo ainda não possuem horários no formato digital e os criadores da Trafi estão a tentar ultrapassar essa dificuldade ao fornecerem as suas soluções às respetivas autoridades de transporte, para que estas se possam transformar rapidamente em *Smart Cities*, pelo menos no que toca aos transportes públicos. Além disso, o facto de a aplicação ser baseada em mecanismos de *crowdsourcing* faz com que só funcione bem caso tenha muitos utilizadores uma vez que a posição dos transportes públicos é fornecida, manual ou automaticamente, pelos próprios utilizadores. Há também o risco de sabotagem, uma vez que os utilizadores podem reportar informações falsas (e.g. atrasos, trânsito, interrupções de percursos), e poluir desta forma o bom funcionamento da aplicação.

2.4 Citymapper

Eleita pela Apple como aplicação do ano em 2013 e 2014, o Citymapper [4] é uma aplicação britânica de transporte urbano criada por um ex-funcionário paquistanês da Google, Amat Yusuf. Fundada em 2011, nasceu inicialmente direcionada apenas para autocarros – chamava-se, na altura, Busmapper – mas expandiu-se desde então com o objetivo de se tornar o guia para qualquer citadino preocupado em saber como se deslocar numa cidade. A aplicação cobre 29 cidades, estando presente tanto na Europa como na Ásia e Américas.

A aplicação funciona com base nos dados públicos provenientes das empresas de transporte incluindo opções para rotas de transporte públicos clássicos, ciclismo e transportes Uber. Além de permitir inserir destinos como ruas, códigos postais ou monumentos, a aplicação utiliza a API Foursquare que permite procurar qualquer tipo de local, desde restaurantes, bares, discotecas, etc.

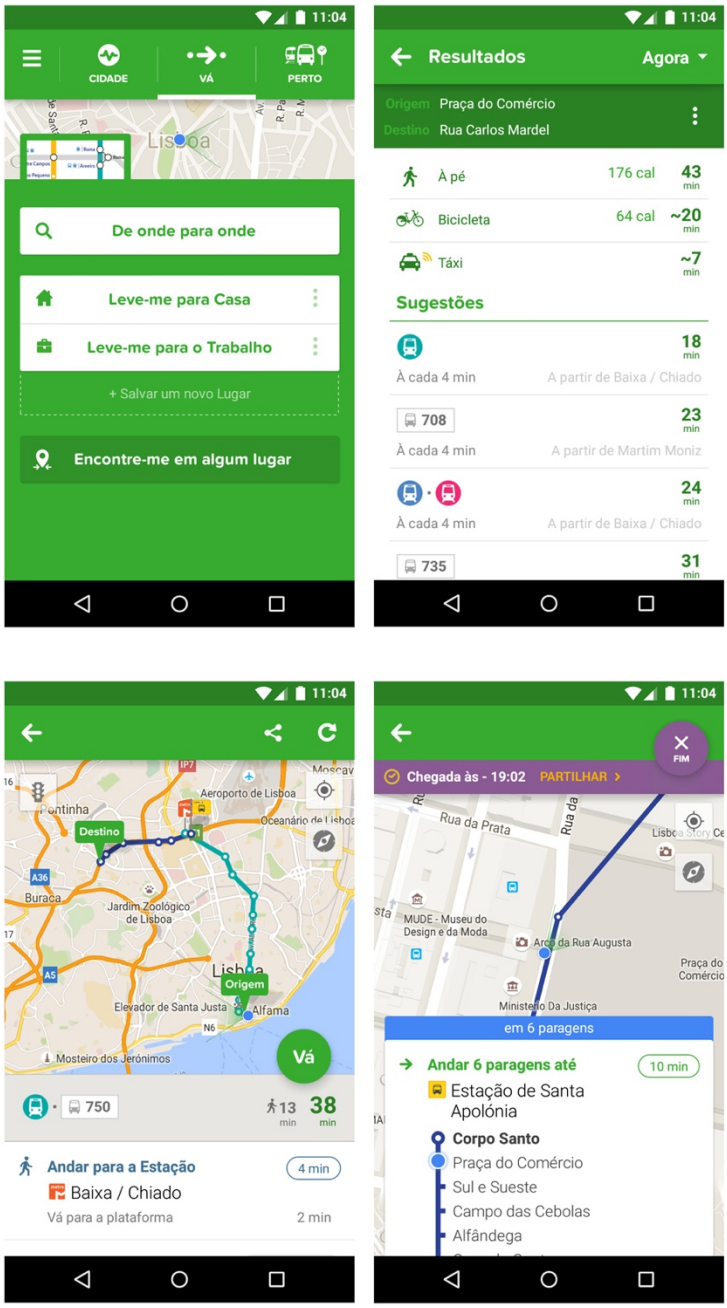


Figura 10 – Exemplos de funcionalidades da Versão Móvel do Citymapper

Disponível em várias cidades, desde São Francisco a Tóquio, a aplicação chegou recentemente à região de Lisboa com itinerários de metro (inclui não só o metro de Lisboa, como o metro Sul do Tejo), autocarro, elétrico, comboios urbanos (CP e Fertagus) e barcos. O Citymapper reúne os dados dos vários operadores de transporte para fornecer os melhores percursos envolvendo vários tipos de transporte para ir se ir dum determinado ponto de origem até um determinado ponto de destino.

O Citymapper disponibiliza todas as paragens que se encontram à volta da localização atual do utilizador.

Esta aplicação tem também a opção de utilização em modo *offline*, possibilitando a consulta dos horários e mapas de metro.

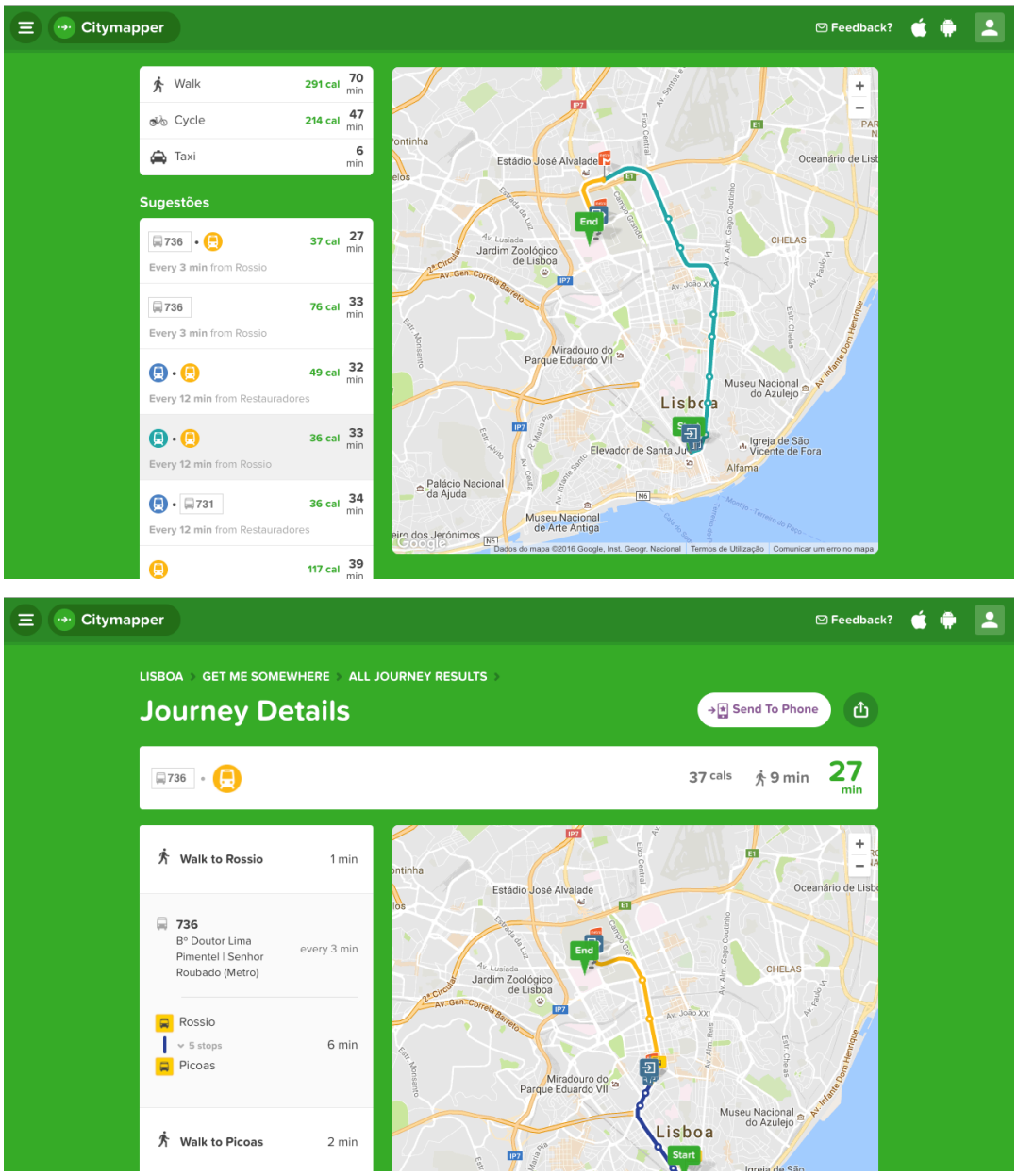


Figura 11 – Exemplos de funcionalidades da Versão Desktop do Citymapper

Em complemento, para além de percursos de transporte público, o Citymapper também recomenda percursos a pé, de bicicleta ou de carro. Neste último caso, não fornece direcções, mas diz quanto tempo demora a fazer o percurso de táxi ou de Uber. Não é possível chamar

um táxi usando a aplicação, mas é possível pedir um Uber, sendo que é apresentada uma estimativa do custo da viagem bem como o tempo de espera.

O Citymapper é extremamente vocacionado para quem se desloca em cidade sem usar o carro particular, um cuidado que se nota em toda a interface, toda ela cuidadosamente desenhada e organizada. Pode-se adicionar a localização da residência e local de trabalho do utilizador, bem como definir outros locais como favoritos para rápido acesso.

A aplicação fornece também um guia da cidade onde o utilizador se encontra, como é o exemplo de Lisboa, onde é possível aceder a mapas e horários do Metro de Lisboa, do Metro Sul do Tejo, dos comboios urbanos CP e Fertagus, dos autocarros e eléctricos da Carris e dos barcos Transtejo & Soflusa.

Como grande desvantagem, o Citymapper desconhece a posição em tempo real dos transportes públicos, remetendo-se à informação dos tempos planeados para as diferentes carreiras. Desta forma, a aplicação só funciona em cidades e horários onde geralmente os tempos não sofrem grandes atrasos. Apesar disto, o Citymapper é reconhecida por muitos como sendo a melhor aplicação de transportes urbanos, pelo menos para as cidades que a aplicação suporta.

2.5 Transit

O Transit [5] é uma aplicação de transportes urbanos em tempo real, que oferece suporte em mais de 120 cidades por todo o mundo. Permite ao utilizador seguir de forma fácil direções passo-a-passo em torno de uma nova cidade, com a ajuda das previsões em tempo real dos tempos de chegada dos transportes. Sendo uma aplicação bastante poderosa, o mapa embutido na aplicação mostra aos seus utilizadores onde, por exemplo, um dado comboio ou autocarro pretendido pelo utilizador se encontra. Além disso, lança notificações ao utilizador quando este deve sair e em que estação.

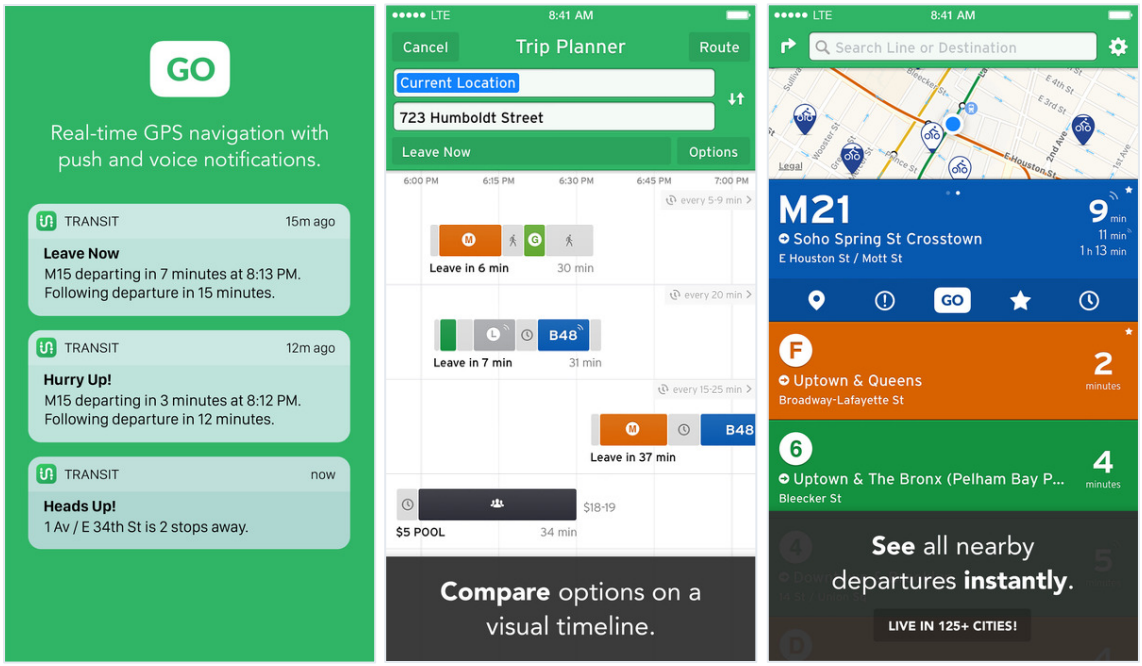


Figura 12 – Exemplos de funcionalidades do Transit

Quer o utilizador tenha conexão à internet ou não, este é capaz de visualizar os horários das paragens mais próximas, verificar os ETA dos autocarros e até mesmo solicitar um Uber, caso nenhum outro método de transporte atenda às necessidades do utilizador. A aplicação suporta praticamente todos os meios de transporte, tais como ferry, autocarro, comboio, metro, bicicleta, entre outros, dependendo da cidade onde o utilizador se encontra.

O Transit é uma ótima alternativa ao Citymapper e um excelente complemento ao Google Maps, própria para aqueles momentos em que o utilizador necessita de informações mais detalhadas sobre como se movimentar numa nova cidade. Em comparação direta com o Citymapper, o Transit oferece suporte para muitas mais cidades, mas o seu conjunto de funcionalidades não é tão extensivo como o do Citymapper.

2.6 Transit vs Moovit vs Citymapper

Nesta secção é feita uma comparação das três aplicações dedicadas mais utilizadas no mundo. De notar que eficiência e qualidade relativas da informação disponibilizada por cada uma das três aplicações, depende das cidades em que o utilizador estiver.

Tendo em conta o caso de estudo da cidade de Nova Iorque [6], registou-se o seguinte:

- O Citymapper parece conseguir sistematicamente as melhores rotas. Dá informação acerca da secção do comboio em que se deve entrar para obter as trocas mais fáceis e ainda oferece rotas "*rain safe*" que são subterrâneas, tanto quanto possível. Tinha todas as opções de trânsito que o Transit oferece na área metropolitana e que não estão presentes em Nova Iorque. A opção Bikeshare mostra quantas bicicletas estão disponíveis e onde deixá-las. Apesar de só estar disponível em 29 cidades, demonstrou ser o melhor e mais claro mecanismo de encaminhamento para a maioria das rotas ponto-a-ponto.

- A interface do Transit, sobretudo no que diz respeito à disponibilização dos horários das chegadas de autocarro em paragens próximas é a mais completa e apelativa, uma das razões que faz com que alguns utilizadores prefiram esta aplicação ao Citymapper. Também está disponível em mais cidades. No entanto, a Transit tem uma menor qualidade nas rotas escolhidas e nos transportes aconselhados, chegando mesmo a cometer alguns erros, não disponibilizando a opção de comboio nalgumas partes da cidade. Por outro lado, a Transit tem uma melhor interface com o serviço da Bikeshare.

- O Moovit conseguiu os piores itinerários em Nova Iorque e não tinha conhecimento de alguns comboios a partir dos aeroportos. Por outro lado, deu os itinerários mais rápidos e flexíveis de Chicago. A verdadeira força desta aplicação é que ela cobre mais de 700 cidades - muito mais do que as 80 da Transit e as 30 do Citymapper.

Através da análise deste estudo [6] pode concluir-se que a aplicação que cada utilizador escolher vai depender da cidade onde esse utilizador se encontra. O Citymapper parece ser a mais escolhida pelos utilizadores visto que tem o melhor equilíbrio entre opções de rotas e apresentação clara das viagens. No entanto, só se encontra disponível em 30 países.

2.7 Outras aplicações

Em Portugal estão presentes outras aplicações, embora não tão evoluídas como as que já foram apresentadas:

- **Sapo Transportes** [7] – Permite em qualquer lugar e instante informar os utilizadores acerca de itinerários possíveis para qualquer ponto (origem/destino) da área metropolitana de Lisboa e consultar os horários e percursos das diferentes carreiras/linhas, organizados por modo de transporte. No entanto, esta aplicação desconhece a posição dos veículos de transporte e, portanto, os seus tempos de chegada são fixos e dependem do bom funcionamento dos transportes.

- **MOVE-ME** [8] – Esta aplicação permite o acesso a informação diversificada sobre transportes públicos, incluindo a consulta de informação de diversos operadores e que permite planear uma rota com transbordo entre operadores. Ainda permite encontrar tempos de partida e chegada de veículos e, com a funcionalidade complementar de mapeamento, permite, numa dada localização, definir paragens mais próximas, pontos de interesse mais próximos, próximo veículo a passar na paragem, entre outros. Mais uma vez, a localização em tempo real da aplicação não é precisa, dependendo fortemente de horários fixos.

- **SMS/E-Mail ao Minuto** [9] – A Carris de Lisboa fornece um sistema de mensagens SMS, ou de correio eletrónico, ao minuto que tem como principal objetivo proporcionar informação - via telemóvel - sobre os horários reais de passagem dos veículos nas paragens. A pedido do cliente, o serviço proporciona a consulta de estimativas relativamente fiáveis sobre a passagem dos veículos nas próximas paragens. Ainda que a localização pareça ser em tempo real, as tecnologias utilizadas e o modo interno de funcionamento do sistema não é claro.

- **TUB Mobile** [10] – A empresa de Transportes Urbanos de Braga (TUB) lançou em 2013 uma aplicação que permite ao utilizador visualizar os horários dos autocarros e as paragens de cada percurso. Embora possuam um sistema de monitorização que

pretende ser em tempo real, devido a problemas técnicos de implementação, os tempos previstos de chegada não são fiáveis e o utilizador depende sempre dos horários fixos. Quando contactada no sentido de se obter informação técnica adicional, a empresa não conseguiu indicar com detalhes relevantes o tipo de tecnologia utilizada e a origem destes problemas. Da informação obtida conclui-se que a aplicação apresenta o tempo de passagem do autocarro nas paragens com base na média do tempo de passagem registado nos últimos três meses. Quer isto dizer que os autocarros registam a sua posição em tempo real e que de alguma forma (em tempo real ou não) essa informação é adicionada à base de dados central para depois se obter uma estimativa dos tempos de passagem.

2.8 Mecanismo de *Crowdsourcing*

No geral, as aplicações de transportes urbanos utilizam um ou dois métodos distintos para obter informações relevantes das rotas, carreiras, horários, etc.: os dados são inseridos manualmente pelas administrações das aplicações e dos operadores ou os dados são obtidos a partir de *crowdsourcing* [22].

No primeiro caso, o método é eficaz e permite obter resultados precisos apenas em contextos onde os transportes mantenham os horários planeados, independentemente da altura do dia.

No segundo método, o passageiro contribui ativamente para os dados da aplicação ao enviar em tempo real a sua posição e velocidade de deslocamento para o servidor central. Quanto mais utilizadores contribuirem, mais precisas e fiáveis serão as estimativas calculadas pela aplicação. Este mecanismo, normalmente implícito, apresenta as vantagens adicionais de custos reduzidos (cada utilizador é um voluntário gratuito), qualidade das estimativas (independentemente da altura do dia) e escalabilidade (aliás, a sua utilização em larga escala é desejável para a obtenção de resultados com maior qualidade).

De notar que o mecanismo *crowdsourcing* poder ser sensível a contribuições de baixa qualidade (conscientemente ou não) e a algoritmos de inteligência artificial menos eficazes. Normalmente este método tem a necessidade de implementação de algoritmos, mais ou

menos complexos, de filtragem da informação. Além disso, há sempre a probabilidade de uma aplicação baseada em *crowdsourcing* ser pouco fiável devido à falta de motivação ou ao número reduzido de utilizadores não aumentar ao longo do tempo. Ou seja, a qualidade da informação das aplicações que utilizam este mecanismo depende, em grande parte, da qualidade e quantidade das contribuições dos próprios utilizadores.

2.9 Análise crítica

Das várias aplicações descritas anteriormente, pode-se inferir com clareza dois tipos de serviços distintos, eventualmente com arquiteturas de implementação também distintas:

- Um tipo de aplicação genérica, construída sobre um serviço web, em que a informação é gerida num sistema interno de bases de dados ligado diretamente ao servidor da aplicação e a forma como a informação é representada não é conhecida.

  Neste modelo, os utilizadores únicos da aplicação são os passageiros e usam um componente de interface que funciona como uma aplicação cliente. Estes componentes podem estar disponíveis para várias plataformas mas a sua construção e disponibilização são sempre da responsabilidade dos próprios serviços.

  Como informação base estes serviços obtêm os dados fixos (rotas, percursos, carreiras, horários, preçários, etc.) diretamente dos operadores de transportes públicos de cada cidade/região. Essa informação é depois importada para as bases de dados internas e mantida atualizada com a ajuda desses mesmos operadores. A informação não estática do sistema que é utilizada para monitorização em tempo real dos serviços de transporte é obtida, quase exclusivamente, por mecanismos de *crowdsourcing*, uma vez que não existe forma das eventuais aplicações informáticas dos operadores, caso disponham de mecanismos próprios de monitorização do sistema, interagirem duma maneira normalizada com os servidores destes serviços genéricos.

  Assim, este modelo não é vantajoso para os próprios operadores de transporte público porque não podem controlar a forma como os utilizadores acedem às informações dos seus serviços, têm um controlo limitado sob a forma como a informação cedida aos servidores vai ser utilizada e não conseguem integrar, caso os

possuam, dos seus próprios sistemas de monitorização em tempo real. Ou seja, os operadores são vistos como agentes cooperativos passivos e não como um tipo especial de cliente dessas aplicações. Por outro lado, este modelo facilita uma integração dos vários operadores numa única aplicação e, em geral, é um tipo de serviço que não acarreta custos relevantes para os operadores.

- O outro tipo de aplicação disponível é específica para cada operador de transportes (ou até para um pequeno grupo de operadores sob a alçada do mesmo grupo empresarial ou organização estatal).

  Em termos de arquitetura técnica, o modelo é um pouco diferente do anterior, no sentido em que costuma disponibilizar-se pelo menos mais um componente informático de interface, próprio para a administração do sistema pelos próprios operadores. Além disso, o componente de monitorização do estado do sistema é da responsabilidade direta dos próprios operadores, ainda que, nalguns casos, possam incluir também mecanismos de *crowdsourcing*. De notar que, em termos de representação da informação nas bases de dados internas e da comunicação entre os componentes do sistema, este modelo é similar ao anterior, usando-se soluções tecnológicas próprias, escolhidas internamente durante o período de desenvolvimento dos projetos.

  Neste tipo de aplicação, o interface disponível para os passageiros é específico de operador para operador.

  As maiores limitações deste modelo são a dificuldade maior de integração com outros operadores de transportes e a limitação geográfica da sua utilização (normalmente circunscrita a uma região ou cidade). Ou seja, dificultam a vida aos passageiros que utilizem regularmente mais do que um operador de transporte ou que viajem com regularidade para fora da sua região ou cidade.

Tendo em consideração algumas das limitações das duas abordagens anteriores, surgiu a oportunidade de estudar a possibilidade de definir o modelo dum serviço de informação e monitorização para operadores de transporte público que utilizasse um esquema intermédio,

fosse o mais modular possível e utilizasse protocolos normalizados para a representação da informação das bases de dados e para a comunicação entre os diversos componentes.

As mais valias subjacentes a tal modelo estão ligadas à hipótese de construção dum serviço que, sendo genérico e escalável, portanto passível de poder ser utilizado por vários operadores de transporte em contextos geográficos de qualquer parte do mundo, também permitisse que os operadores tivessem disponíveis componentes de administração e de controlo direto das suas bases de dados, permitindo até que a sua informação não tivesse que ser importada para bases de dados de terceiros. Mais ainda, um modelo que permitisse que a construção dos vários componentes de interface para os operadores e para os passageiros pudessem ser construídos por entidades/empresas diversas que, em concorrência, disponibilizassem várias alternativas de aplicações.

Os operadores continuariam a poder integrar nas suas bases de dados a sua informação de monitorização em tempo real para que esta pudesse ser utilizada diretamente nas componentes de interface para os utilizadores. Finalmente, esta arquitetura permitiria que mecanismos complementares de monitorização baseados em *crowdsourcing* pudessem também ser adicionados como componentes do sistema.

De notar que seria possível que distintas entidades/empresas pudessem construir e oferecer em concorrência, não só os componentes de interface para os utilizadores/passageiros mas também os componentes que implementem, em separado, os mecanismos de monitorização em tempo real dos serviços de transporte públicos baseados em *crowdsourcing*. Já os mecanismos de monitorização mais precisos serão, em princípio, apenas da responsabilidade dos operadores uma vez que é necessária tecnologia implementada nos próprios veículos de transporte e/ou paragens ou zonas de passagem pré-definidas.

3. TECNOLOGIAS RELACIONADAS

Conforme se pode concluir das seções anteriores, um aspeto tecnológico importante e diferenciador das aplicações de informação de transportes urbanos é o componente de monitorização do sistema em tempo real. Normalmente este componente pode ser implementado por *software* através de mecanismos que envolvem os próprios utilizadores/passageiros ou através de tecnologias mistas (*software* e *hardware*) e que são da responsabilidade dos próprios operadores. Neste último caso é possível obter resultados muito mais precisos e fiáveis mas os custos de implementação e manutenção, para os operadores, são relevantes, enquanto no primeiro caso, os custos para os operadores são praticamente inexistentes mas a precisão e fiabilidade dos resultados são menores.

3.1 AVL

Atendendo à qualidade dos resultados e à sua utilização alargada pelos operadores de transportes (tanto de cargas como de passageiros), foi feito um estudo das principais tecnologias atuais utilizadas na implementação dos mecanismos de localização automática de veículos em tempo real, ou *Automatic Vehicle Location Systems* (AVL) [11], até para perceber quais seriam as necessidades de representação da informação de monitorização nas bases de dados do sistema. Do estudo, apresentado no Anexo I – Tecnologias AVL, resulta que os sistemas de monitorização em tempo real da posição dos veículos de transporte mais utilizados estão baseados em equipamentos com tecnologia *Global Positioning System* (GPS) [12].

Independente das tecnologia base das soluções disponíveis, o fator comum a todas elas é a necessidade dum mecanismo de comunicação dos dados de monitorização entre os equipamentos base que calculam a posição dos veículos e o sistema de base de dados central.

3.2 SNMP

Nas aplicações em que foi possível apurar a tecnologia que as aplicações utilizam para guardar e ter acesso aos dados dos transportes públicos (e.g. horários, posição, etc.),

verificou-se que esses dados são guardados em bases de dados utilizando variadas tecnologias de bases de dados. A mais usada está baseada na norma *Structured Query Language* (SQL) [13]. Esta tecnologia é amplamente usada em aplicações de bases de dados em que um desenvolvedor de software controla todos os componentes do sistema. No entanto, no sistema que se pretende desenvolver, os componentes serão módulos independentes que podem ser definidos e construídos por entidade/empresas distintas pelo que o paradigma da normal SQL poderá não ser a mais adequada.

Assim, a representação da informação de serviços de transportes urbanos, e o transporte comunicacional necessário para gerir e transmitir esta informação entre componentes, irá ser feita através de uma tecnologia normalizada, universal gratuita, definida no contexto do protocolo SNMP [14]. Ou seja, toda a informação necessária para caracterizar o sistema (e.g. paragens, carreiras, horários, mapas, etc.) irá ser representada e integrada através duma MIB criada para o efeito. Informação esta que, posteriormente, deverá ser acedida e atualizada através do protocolo SNMP. Os módulos do sistema que acedem a MIB através do SNMP poderão utilizar a informação acedida de maneiras diversas na implementação dos vários componentes de interface. A integração/inserção e manutenção/gestão de toda a informação respeitante a um serviço de transporte público poderá ser feita através duma aplicação gestora SNMP, um componente de interface que será controlado pela entidade que administra o sistema, provavelmente um operador de transporte. Além duma superior adequação do SNMP para a implementação de componentes do sistema desenvolvidos por entidades independentes, a representação da informação em MIBs e o protocolo de comunicação são universais, independentes da plataforma utilizada e utilizam menos recursos computacionais e de comunicação do que mecanismos clássicos de implementação de bases de dados relacionais como o SQL.

A utilização do SNMP, no contexto deste projeto, será explicada com mais detalhe em seções posteriores onde o modelo da solução completa é descrita.

3.2.1 SNMP: Uma breve introdução

O SNMP é um protocolo padrão da Internet utilizado originalmente para gerir informação de monitorização e controlo de dispositivos em redes IP. Modificando essa informação é uma forma de mudar o comportamento dos dispositivos. Hoje em dia, praticamente todos os dispositivos como *routers*, *switches*, *servers*, *workstations*, impressoras, entre outros, possuem suporte para SNMP. Foi desenhado de forma a ser uma ferramenta básica de gestão de redes e de implementação fácil e que consome poucos recursos. Uma das chaves do seu sucesso e aceitação generalizada é a sua simplicidade relativa.

O SNMP é amplamente usado em sistemas de gestão de redes para monitorizar dispositivos conetados à rede para condições que exigem atenção administrativa. Expõe os dados de gestão sob a forma de variáveis nos sistemas de gestão, que descrevem a configuração do sistema. Estas variáveis podem então ser consultadas (e por vezes alteradas) por aplicações (ou clientes) de gestão.

O SNMP é uma componente do conjunto de protocolos definido pela *Internet Engineering Task Force* (IETF) [15] e consiste num grupo de normas para a gestão de redes e dispositivos, implementadas na camada de aplicação, incluindo um esquema de representação abstrata duma base de dados que descreve univocamente um conjunto de objetos. Estes objetos são abstrações de parâmetros de funcionamento dos dispositivos a gerir ou de dados genéricos de aplicações internet.

Na maioria dos contextos, o software de gestão de redes não segue o modelo cliente-servidor convencional, uma vez que para as operações GET e SET da estação de gestão comporta-se como um cliente e o dispositivo de rede a ser analisado ou monitorizado assume o papel de servidor. Na operação TRAP/NOTIFICATION ocorre exatamente o oposto, uma vez que no envio de alarmes é o dispositivo gerido que toma a iniciativa da comunicação. Os sistemas de gestão de redes evitam os termos “cliente” e “servidor” e optam antes por usar "gestor" para a aplicação que corre na estação de gestão, e "agente de gestão" para a aplicação que corre no dispositivo de rede (ou aplicação a ser gerida).

Implementar a gestão dum dispositivo de rede através do SNMP é muito mais direto do que a maioria das abordagens alternativas para gestão de redes. Apesar disso, o desenvolvimento de aplicações SNMP não é tão simples como se esperaria que fosse, uma vez que exige um esforço significativo para desenvolver aplicações de gestão para acolher uma variedade de dispositivos de rede. Em qualquer dos casos, o SNMP tornou-se no sistema normalizado de gestão de redes dominante nos dias de hoje.

3.2.2 Arquitetura SNMP

No contexto de uso do SNMP, um ou mais computadores administrativos, denominados de gestores, têm a tarefa de monitorizar ou gerir um grupo de *hosts*, dispositivos ou aplicações numa rede de computadores. Cada sistema gerido executa, a toda a hora, uma componente de software denominada de agente que reporta informação via SNMP para o manager, a pedido do último ou através de notificações não solicitadas. Existem várias formas de implementar o protocolo SNMP, mas geralmente uma rede gerida com base neste protocolo é formada por dois componentes essenciais: um agente SNMP que implementa, pelo menos, uma MIB, e que reside no dispositivo/aplicação a ser gerida (no caso mais genérico, um agente pode ser composto por um agente master que controla e divide o fluxo de informação por vários sub-agentes que implementam uma parte específica duma MIB ou MIBs distintas); um gestor SNMP que comunica com os agentes SNMP e que implementa uma aplicação de gestão para monitorizar e controlar os dispositivos ou aplicações na rede.

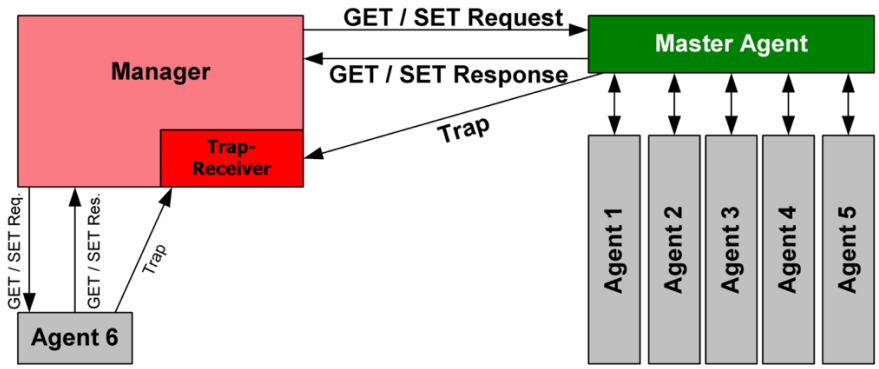


Figura 13 – Princípios da Comunicação SNMP [16]

3.2.3 MIBs

Os agentes SNMP expõem dados de gestão na forma de variáveis, ou objetos de gestão, nos sistemas geridos. O protocolo permite também tarefas de gestão ativa, tais como modificação e aplicação de novas configurações através da modificação remota destas variáveis ou objetos. As variáveis acessíveis via SNMP estão organizadas em hierarquias, e estas hierarquias e outros metadados (como tipo e descrição da variável) estão descritos em bases de dados de objetos de gestão, ou MIBs.

O próprio SNMP não define qual a informação ou quais as variáveis que um sistema gerido deve oferecer. Ao invés disso, o protocolo SNMP utiliza um desenho extensível onde a informação disponível é definida por MIBs. Estas descrevem a estrutura de dados geridos, utilizando nomeação hierárquica utilizando identificadores de objeto, ou *Object Identifier*. (OID). Cada um destes OIDs identifica uma variável que pode ser lida ou alterada via SNMP. As MIBs usam a notação definida na norma *Structure of Management Information – Version 2* [17].

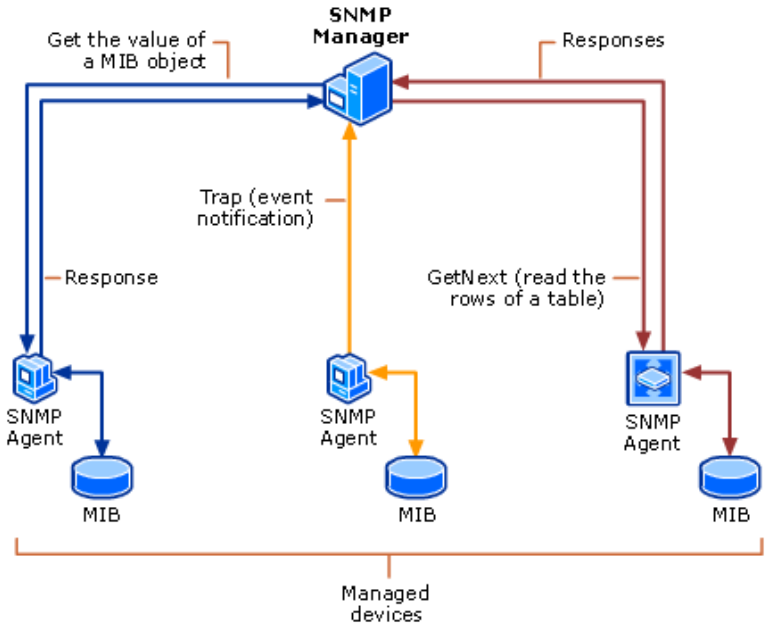


Figura 14 - Exemplo de funcionamento do SNMP [18]

A definição dos objetos das MIBs é feita em ASN.1 [19], o qual atribui a cada objeto uma sequência longa que garante a unicidade do identificador completo, sendo que a cada nome da sequência é atribuído um número inteiro. Uma vez que a definição de objetos é independente do protocolo de comunicação, o SNMP permite criar novos conjuntos de

variáveis MIB para novos dispositivos, novos protocolos, novas aplicações, etc., todos pendurados na mesma árvore de nomeação.

Na figura podemos verificar a base da árvore comum a todos os objetos de todas as MIBs definidas. Cada OID completo é definido como o caminho da raiz até à folha respetiva do objeto.

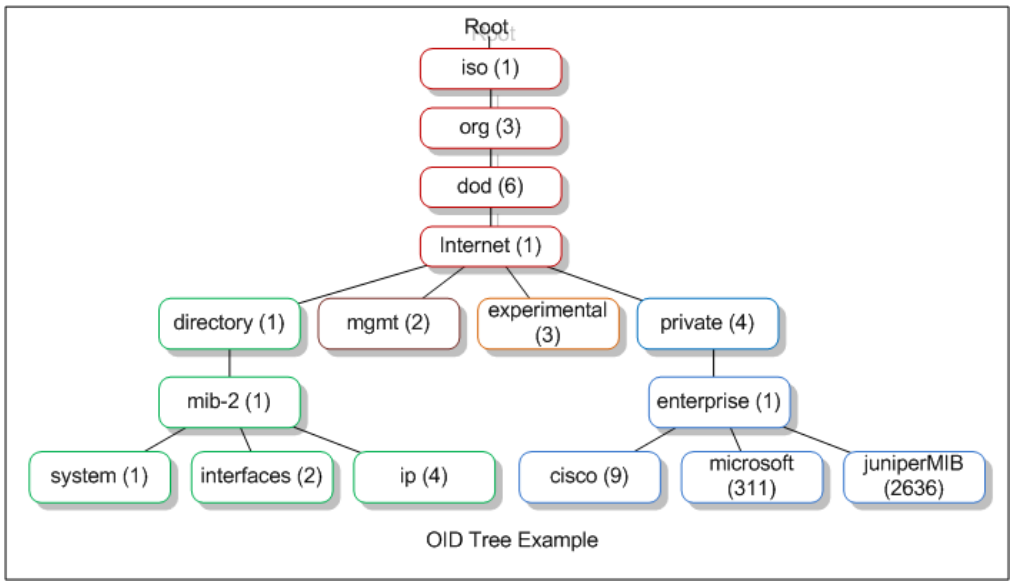


Figura 15 - Árvore base de todos os OIDs de MIBs [20]

3.2.4 Comunicação protocolar no SNMP

O protocolo SNMP é, na maior parte dos casos, encapsulado no protocolo de transporte UDP, ou seja, opera na camada de aplicação da pilha TCP/IP - camada 7 do modelo OSI – e, por defeito, os agentes recebem pedidos na porta UDP 161 e geram notificações a partir de qualquer porta UDP disponível no sistema. Um gestor SNMP envia e recebe pedidos a partir de uma qualquer porta UDP de origem disponível no sistema mas recebe notificações obrigatoriamente na porta UDP 162.

Relativamente a aspectos de segurança, até à versão 2 do SNMP, não era suportado qualquer tipo mecanismo de autenticação e confidencialidade. Isto tornava o protocolo mais vulnerável a uma série de ameaças de segurança, como acesso e modificação não autorizadas de dados nas MIBs dos dispositivos geridos. Esta vulnerabilidade do protocolo fez com que

diversos fabricantes não implementassem a operação *SET*, reduzindo assim o SNMP a uma ferramenta de monitorização.

A SNMPv2 é uma evolução do protocolo inicial, oferecendo uma quantidade de melhorias muito relevantes relativamente à primeira versão. Estas incluem melhorias no desempenho dos mecanismos de *polling*, segurança e controlo de acesso e comunicação entre gestores. O único avanço significativo que o SNMPv3 veio trazer tem a ver com a obrigatoriedade clara de implementação dos mecanismos de segurança e controlo de acesso, que na versão 2 do protocolo não era cumprida.

A versão 1 do protocolo especificava cinco tipos de unidades protocolares de dados: *GetRequest, GetResponse, GetNextRequest, SetRequest, SetResponse* e *Trap*. Mais dois foram adicionados na segunda versão: *GetBulkRequest* e *InformRequest*. Estas unidades protocolares de dados, ou *Protocol Data Units* (PDUs), permitem a definição das seguintes primitivas: GET, GETNEXT e GETBULK (utilizadas pelos gestores para consultarem, através do método de *polling*, a informação contida nas MIBs implementadas pelos agentes), SET (utilizada pelos gestores para modificarem a informação mantida nas MIBs implementadas pelos agentes), RESPONSE (utilizada pelos agentes para responderem às primitivas anteriores enviadas pelos gestores), TRAP e NOTIFICATION (utilizada pelos agentes para enviarem alertas não solicitados explicitamente pelos gestores) e INFORM (utilizada pelos gestores para trocarem informações entre si).

No contexto deste projeto, o SNMP é utilizado para monitorizar e controlar a informação aplicacional dum sistema de transportes público, que é representada através duma MIB de objetos criada para o efeito.

4. SOLUÇÃO PROPOSTA

Após o estudado das principais soluções existentes no mercado e das suas tecnologias associadas, foi possível verificar que, atualmente, existem poucos sistemas que tenham a informação integrada com as empresas de transportes públicos e em que essas empresas controlem/administrem diretamente a informação incluída nas bases de dados, como aliás já foi discutido em secções anteriores. Essas soluções, normalmente mais genéricas, também têm métodos de monitorização em tempo real do sistema baseadas exclusivamente em mecanismos de *crowdsourcing*.

Maioritariamente, apenas nas soluções específicas para uma dada região/cidade, e que são da responsabilidade dos próprios operadores de transporte, é que existe a possibilidade de administração direta da informação das bases de dados por parte desses operadores e em que pode ser possível implementar mecanismos de monitorização em tempo real baseados em tecnologias AVL, em geral mais precisas e fiáveis do que as soluções de *crowdsourcing*, ainda que, nalguns casos, por limitações dos processos AVL realmente implementados, a informação recolhida ainda tenha que ser processada e tratada estatisticamente antes de poder utilizada com fiabilidade, o que reduz a sua natureza de "tempo real"…

Durante o estudo, nos casos onde foi possível obter informações técnicas sobre a implementação interna das aplicações, foi descoberto que a tecnologia primordial de implementação das bases de dados para gestão da informação do sistema é baseada em SQL, que não é muito adequada para arquiteturas de aplicações distribuídas com módulos aplicacionais de desenvolvimento independente entre si.

Na busca duma solução que pudesse ser construída por componentes totalmente independentes entre si (desenvolvidos por entidades diversas) foi escolhido o SNMP como tecnologia base para representar e abstrair a informação das bases de dados e também para ser utilizado como protocolo de comunicação entre os componentes. Além disso, comparativamente, é um protocolo que consome poucos recursos computacionais e pouca

largura de banda, sendo escalável e, em caso de necessidade, já tem incluídos mecanismos de segurança para autenticação, confidencialidade e verificação de alteração de conteúdo.

De salientar que não foi encontrado qualquer sistema atual que se baseie no protocolo SNMP, pelo que a solução proposta vem trazer também uma perspectiva diferente no que toca à gestão e integração dos dados utilizando normas universais e amplamente utilizadas e validadas, ainda que noutros contextos.

Assim, o principal objetivo, ao propor um novo modelo de aplicação informática para a gestão da informação e monitorização do estado dum serviço de transportes público, foi a possibilidade de definir uma arquitetura com uma natureza intermédia em relação aos dois tipos de aplicações já encontradas, tal como já apresentado anteriormente. Ou seja, este novo modelo permitiria, por um lado, o controlo direto de toda a informação disponibilizada pelos próprios operadores, incluindo informação necessária para a implementação dum sistema de monitorização em tempo real sem ser baseado em mecanismos de crowdsourcing, além de permitir também a possibilidade de agrupar e integrar vários operadores num único sistema.

Mais concretamente, como principais objetivos para o modelo da solução proposta, podemos enumerar:

- Definir uma arquitetura dum serviço de informação e monitorização para operadores de transporte público que utilize um esquema o mais modular possível;

- Que no sistema se utilize protocolos normalizados para a representação da informação das bases de dados e para a comunicação entre os diversos componentes;

- Permitir que o sistema seja o mais universal e escalável possível, ou seja, que permita a implementação de diversos tipos de soluções, desde as mais genéricas às mais específicas e em contextos geográficos completamente distintos;

- Permitir que no sistema a construção dos vários componentes de interface para os operadores e para os passageiros se faça por entidades/empresas diversas e independentes, que, em concorrência, disponibilizem várias alternativas de aplicações, sobretudo no que diz respeito aos interfaces para os utilizadores/passageiros.

- Permitir que os operadores possam integrar nas suas bases de dados a sua informação de monitorização em tempo real, independentemente dos processos ou tecnologias utilizadas para as obter.

Sendo a solução proposta baseada no protocolo SNMP e que a implementação das bases de dados de informação é feita à custa de MIBs, é desejável que a fiabilidade da comunicação entre os componentes do sistema seja aumentada com a implementação dum processo de retransmissão de pedidos e respostas ao nível aplicacional, isto é, implementado pelos próprios componentes do sistema.

4.1 Limitações à solução proposta

Apesar dos principais objetivos já propostos para a nova solução, tendo em conta a limitação temporal na execução deste projeto, a sua especificação terá que ter também em consideração a capacidade de desenvolver e testar um protótipo dum sistema o mais completo possível, com a inclusão dos componentes essenciais e as suas principais funcionalidades. Nesse sentido, por exemplo, foi decidido que, no contexto deste projeto, não fosse incluído um componente de monitorização em tempo real que incluísse mecanismos AVL clássicos mas apenas um método a implementar nos veículos de transporte e que depende do manuseamento manual duma aplicação informática por parte do motorista ou alguém responsável pela sua execução em tempo real. Além disso, na especificação do modelo foi apenas incluído explicitamente um único operador de transporte público de autocarros, ainda que a arquitetura possa conter mais do que um operador de transportes de qualquer tipo. Por fim, também foi considerada apenas a implementação dum tipo de componente de interface para os passageiros, ainda que a arquitetura possa integrar outros componentes deste tipo.

4.2 Arquitetura do modelo proposto

Após a definição dos objetivos maiores para o modelo proposto e depois de vários esboços preliminares, foi finalmente definida uma arquitectura para a solução a implementar. Esta arquitetura é constituída por quatro tipos de subsistemas distintos e independentes, ainda que concorram para a implementação dum único sistema global. De referir, que, no mínimo, é necessária a implementação dum subsistema de cada tipo.

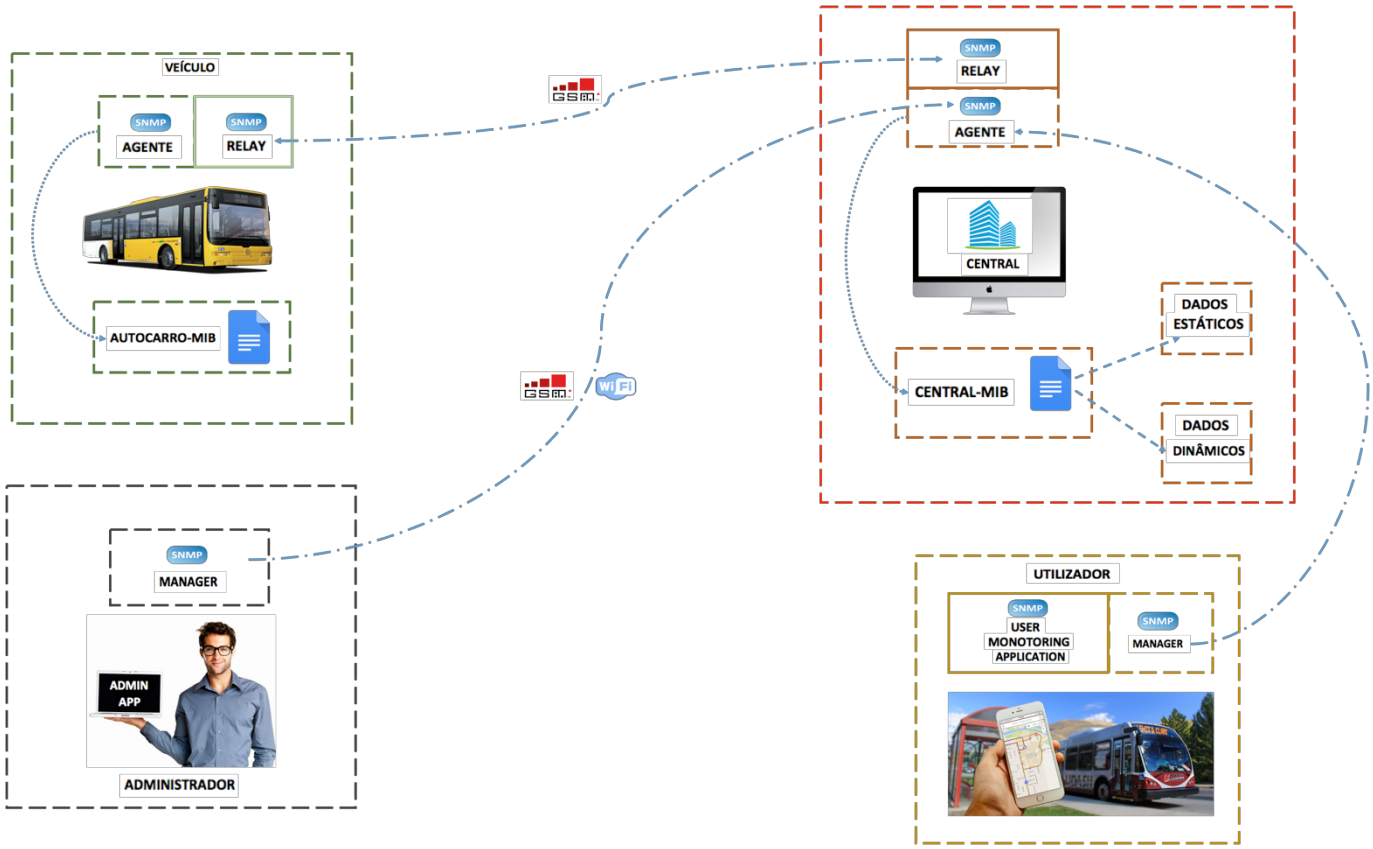
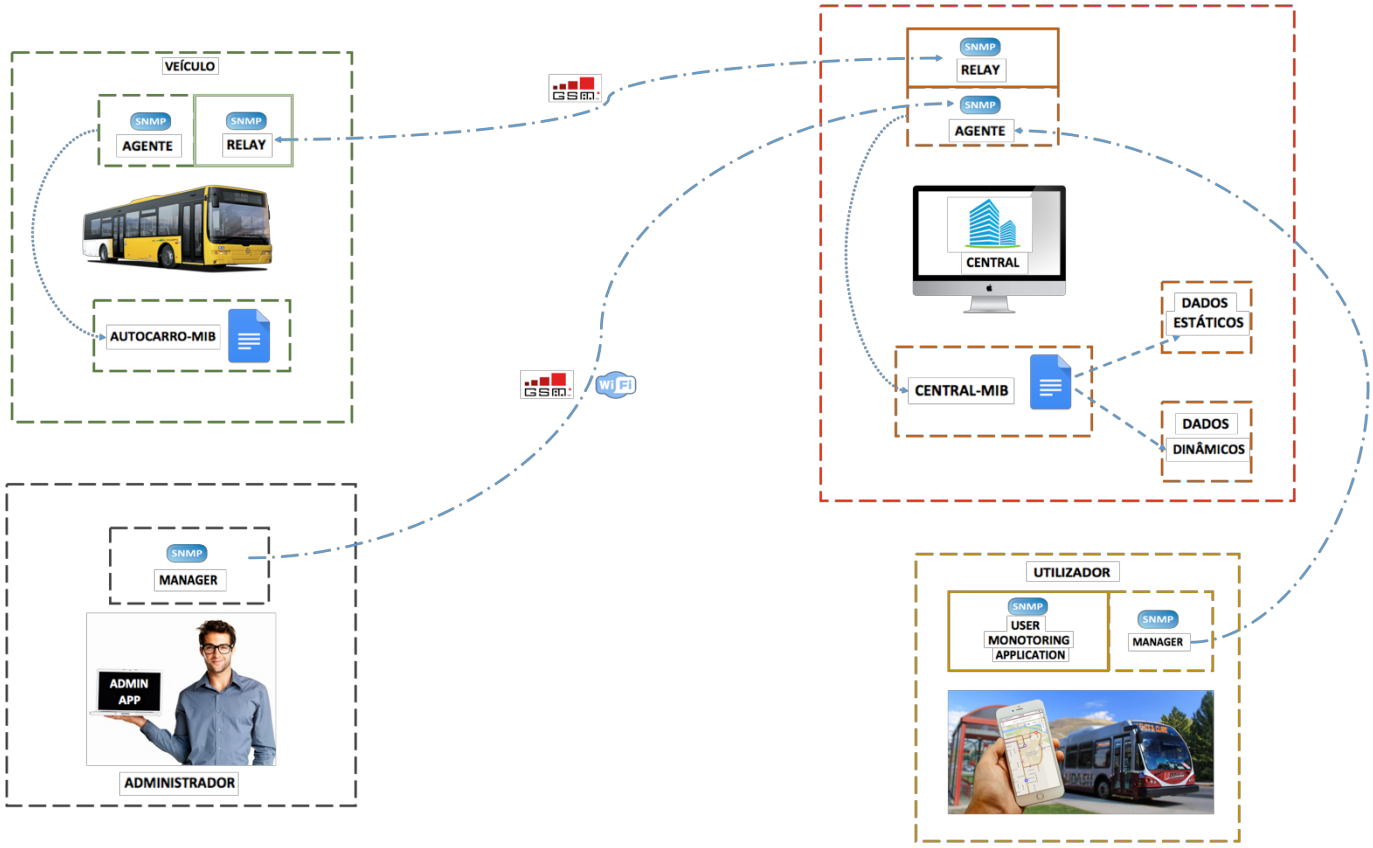


Figura 16 - Arquitetura da solução proposta

Assim, o sistema é então constituído por quatro partes, que serão detalhadas nas secções seguintes e que é genericamente representada pela Figura 16.



:

- A **central** ou **servidor central** onde reside o agente SNMP que implementa a MIB principal e permite o controlo direto da informação do sistema através duma aplicação de gestão utilizado pela entidade administradora. Na especificação agora definida, é referido apenas um servidor central para um operador de transporte mas é possível que haja mais do que servidor central, dependendo do operador, ou até que mais do que operador estejam pendurados no mesmo servidor central.

- O subsistema presente no **autocarro**, que pode ser controlado manualmente pelo motorista (também é possível que este subsistema implemente uma aplicação informática AVL ou um mecanismo de *crowdsourcing*, ou ambos).

- O sistema de **administração** ou gestão de dados, que é manuseado pela entidade administradora (provavelmente da responsabilidade do operador de transporte ou grupo de operadores de transporte representados pela mesma organização).

- O **passageiro** (ou utilizador) que usa o serviço através dum componente de interface, normalmente uma aplicação num dispositivo móvel.

Central

A **central** ou **servidor central** pode ser vista como a base de dados central de todo o sistema de informação de cada operador de transporte. É a central que alimenta todos os outros intervenientes com os dados essenciais ao funcionamento desses subsistemas. Possui um agente SNMP à escuta de pedidos para modificação (alteração, remoção ou criação) ou monitorização de dados. Geralmente, os dados a serem alterados são os dados dinâmicos (e.g. posição de um dado autocarro, entre outros) e os dados a serem monitorizados são os estáticos (e.g. informação de itinerários, horários, etc.).

De forma a receber e responder a pedidos, para além do agente a central possui um *SNMP Relay*. Um *SNMP Relay* é necessário porque existe troca de informações nos dois sentidos, ou seja, na realidade, acoplado ao agente SNMP existe também um gestor/cliente SNMP para se poder monitorizar as MIBs dos componentes nos autocarros e integrar automaticamente a informação nas MIBs do servidor central. Por exemplo, o autocarro quando inicia uma viagem procede à transferência de dados acerca do itinerário que vai fazer, dados esses que são enviados a partir da central após a última receber os vários pedidos (ou pacotes) *GET* por parte do gestor existente no autocarro. No entanto, à medida que o autocarro vai passando pelas diversas paragens do percurso, a sua posição é enviada para a central (caso exista sinal) através de sucessivos pedidos *SET*. Isso impede que a central tenha que estar sempre a "*perguntar*" ao autocarro se já alterou a sua posição. Ao invés disso, o *relay* na central apenas "*escuta*" e processa os pacotes que lhe são enviados. De notar que, por motivos de segurança, apenas clientes SNMP credenciados têm a possibilidade de alterar os dados presentes nas MIBs da central.

O componente de interface de administração envia pedidos *GET* e *SET* para a central sempre que deseja obter ou alterar informações. O componente de monitorização do autocarro também envia e recebe dados. Sempre que inicia um novo trajeto obtém todos os dados do

itinerário da central. Depois, os papéis invertem-se e passa a enviar para a central os dados acerca da posição de um dado autocarro, o itinerário que está a fazer, etc.

Por fim, o subsistema do passageiro apenas faz pedidos à central acerca de dados dos itinerários e da posição dos autocarros. Neste último caso, a troca de informação é unidirecional não havendo qualquer tipo de informação enviada para a central por parte do passageiro, todo o processamento da informação tendo em conta os dados específicos do próprio utilizador (posição geográfica, por exemplo), é feito localmente no próprio componente ou aplicação.

Quanto à MIB implementada no agente SNMP da central é constituída por duas partes distintas:

- **Dados Dinâmicos**: é a informação que está em constante mudança. Esta informação muda consoante a informação recolhida dos veículos (e.g. posição de um dado autocarro). Por exemplo, o gestor SNMP no lado componente do passageiro consulta esta parte da MIB, pois contém os dados fulcrais para estimar o tempo de chegada dos veículos.

- **Dados Estáticos**: é a informação (relativamente) estática da aplicação. Isto é, a informação que muda com pouca frequência como dados sobre os horários, os percursos das carreiras, as paragens, os motoristas, os autocarros, etc. Esta informação é alterada somente pelos administradores e são da responsabilidade direta do operador.

Administrador

O componente de administração tem como principal função adicionar ou remover a informação acerca de horários, itinerários, paragens, motoristas, autocarros, entre outros dados da aplicação referentes a um operador (ou grupo de operadores) a que refere a informação contida nas MIBs implementadas pelo agente SNMP.

No programa de administração existe um gestor que estabelece ligação com o agente presente na central. Embora não seja claro aos olhos do administrador, no decorrer das suas operações sempre que este deseja gerir alguma informação, pedidos *GET* ou pedidos *SET* com a especificação da informação pretendida são enviados para a central, sendo as respostas a esse pedidos processados no próprio componente de administração. Sempre que as operações de administração impliquem a alteração ou remoção de informação da MIB, um ou mais pedidos *SET* são enviados para o agente central que, por sua vez, valida e actualiza em conformidade a informação.

Veículo

O subsistema residente no veículo (neste caso, um autocarro, mas poderia ser outro tipo de veículo ou parte de veículo) possui um agente SNMP para implementar uma MIB própria para gravar as informações necessárias para implementar um sistema de monitorização local (circunscrito ao veículo). A MIB mantida por este agente local contém informação acerca da posição do autocarro, da identificação do motorista que o conduz num dado momento, entre outros dados. Esta informação é guardada localmente, é pré-processada localmente e depois enviada, a intervalos regulares, através do SNMP *relay*, para o sistema central do operador respetivo. Em caso de perdas de ligação existe sempre a garantia de que a informação está guardada localmente e é enviada para a central assim que a ligação é restabelecida.

Utilizador

Por fim, o componente de interface com o passageiro que implementa uma aplicação informativa (ajuda à utilização do serviço de transportes) e de monitorização em tempo real do estado do serviço tendo em conta o itinerário sugerido/associado ao utilizador num dado momento. À medida que esta aplicação vai verificando se existem alterações de posição do veículo, o sistema residente no autocarro vai atualizando a sua posição (que é gravada a um ritmo de frequência superior localmente e, a um ritmo inferior, é depois enviada para o sistema central do operador respetivo através do SNMP *relay*).

Além dos dados de monitorização em tempo real, o gestor SNMP obtém informação da MIB implementada no agente SNMP do sistema central quando o utilizador requisita informação

diversa sobre os serviços, como por exemplo, sobre itinerários, horários, preços, pontos de interesse turístico, etc. Para tornar a utilização da aplicação mais agradável, com tempos de espera irrelevantes, a informação estática do sistema do operador associado à localização do utilizador (ou, na falta dela, por escolha explícita do utilizador) é obtida quando a aplicação é instalada pela primeira vez no telemóvel do utilizador ou quando os dados presentes no telemóvel estão desatualizados. A versão das bases de dados são comparadas e quando uma nova versão é verificada no sistema central, a informação a ser atualizada é outra vez recolhida do sistema central. Esta estratégia permite ao utilizador usar a aplicação ainda que não haja uma ligação de rede ao sistema central, ainda que não esteja disponível a funcionalidade de monitorização em tempo real.

4.3 MIBs

Após definir quais os dados necessários para cada elemento do sistema, procedeu-se à definição das bases de dados de objetos a gerir no sistema utilizando o paradigma SNMP, ou seja, definiram-se as MIBs.

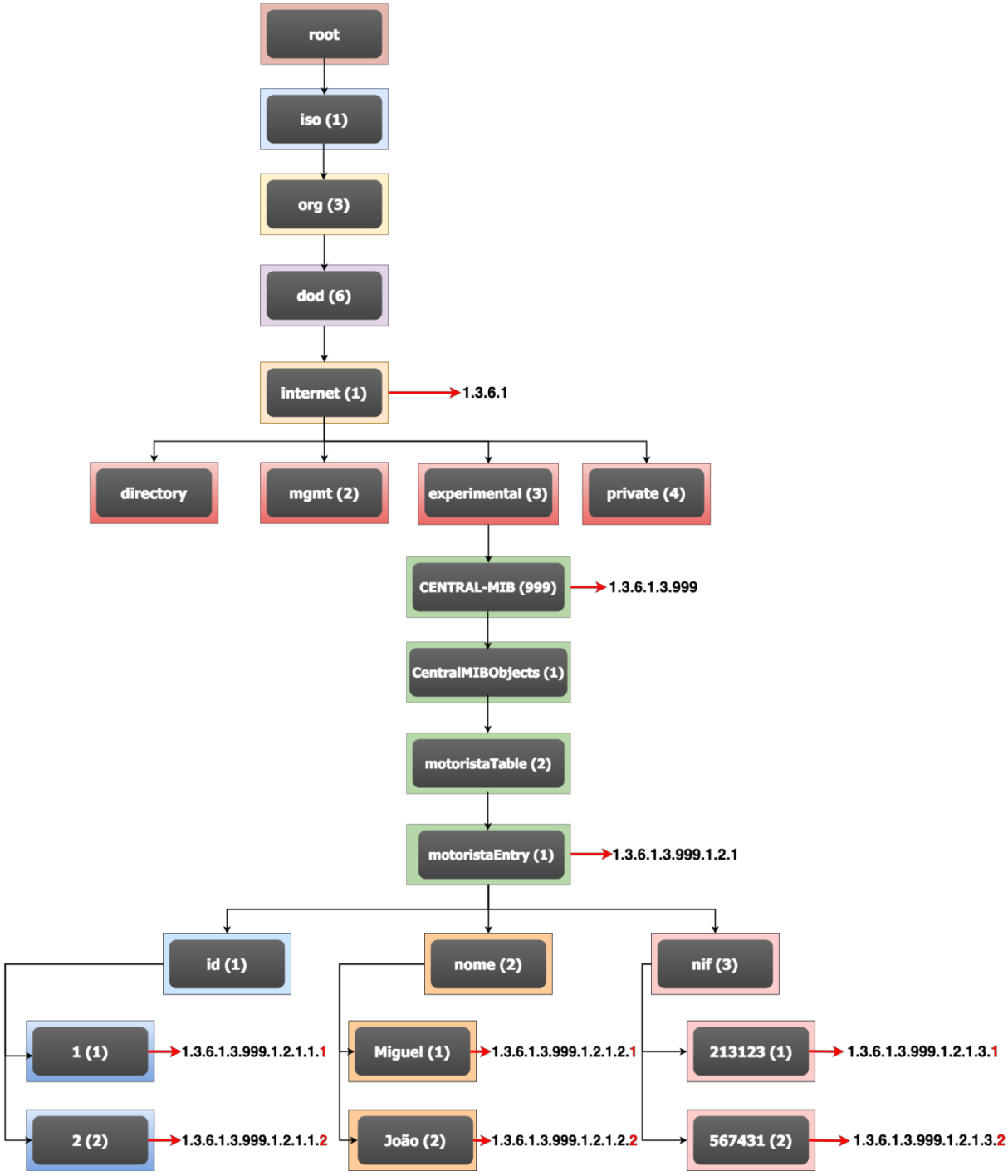


Figura 17 - Árvore de OIDs com inclusão da MIB experimental

Esta foi uma das partes mais importantes do projeto, uma vez que a representação formal dos dados nas MIBs é diferente dos formatos mais vulgares utilizados no desenvolvimento de aplicações de engenharia informática, como, por exemplo, o formato definido pela norma

SQL. No entanto, após a devida adaptação à linguagem ASN.1, a definição de MIBs foi feita recorrendo à aplicação de ajuda à programação MIB Designer [21].

A informação de todo o sistema para um determinado operador é gerida através duma MIB implementada num agente SNMP. A identificação dessa MIB utiliza um OID experimental pendurado na parte de gestão de redes na árvore de identificação global da internet (ver Figura 17, com um exemplo da introdução de alguns dados na MIB).

MIB do servidor central

Do ponto de vista de identificação, todas as definições abstratas de objetos de gestão criadas para a base de dados do componente do sistema central de cada operador estão debaixo da mesma entrada duma única MIB experimental em que o *root object identifier* escolhido foi o OID ***experimental*** (*1.3.6.1.3*) e o *root object identifier suffix* foi o ***999*** (*1.3.6.1.3.999*) uma vez que, nesta fase, se trata dum projecto académico e experimental.

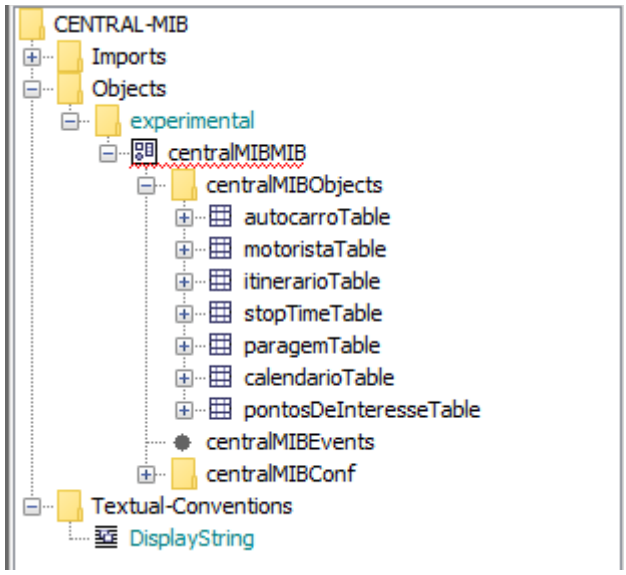


Figura 18 - Estrutura da MIB do sistema central

Apesar de ter sofrido várias alterações ao longo do desenvolvimento do sistema de modo a satisfazer as diferentes necessidades que foram surgindo, a estrutura da versão final é a que se encontra na
Figura 18 (definição obtida diretamente da sua definição feita no MIB Designer).

A estrutura mais detalhada de cada tabela desta MIB está documentada no Anexo III – MIBs do Sistema (para acesso à listagem em SMIv2/ASN.1 por favor contactar o autor).

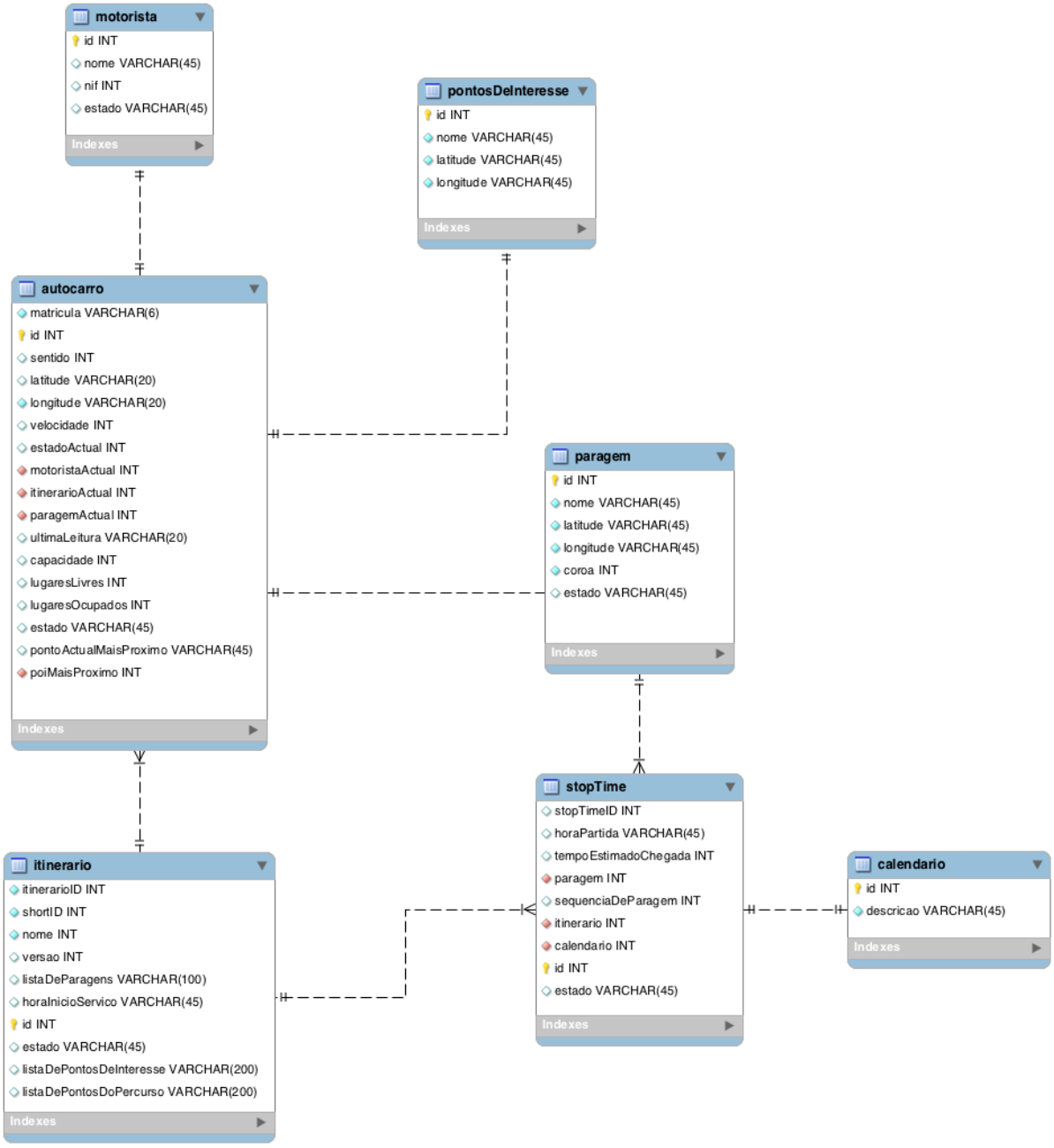


Figura 19 - Representação concetual ER da MIB do sistema central

Embora o modelo considerar a comunicação entre componentes e a implementação das bases de dados utilizando o paradigma SNMP, apresenta-se na Figura 19 o esquema concetual da

base de dados num formato relacional pois esta é uma das formas mais comuns de representar as relações entre as tabelas.

MIB do componente veicular

O agente SNMP do componente veicular implementa uma MIB para poder gerir localmente os dados referentes à informação de monitorização em tempo real do seu estado. A principal vantagem desta abordagem é o facto de permitir o seu funcionamento mesmo durante as alturas em que a ligação comunicacional com o sistema central do operador respetivo se perca. Através do SNMP *relay*, esta informação é depois transmitida para o sistema central quando for possível a comunicação, a uma frequência configurável e adequada para que a monitorização para a sua utilidade seja pertinente para o utilizador.

Por motivos semelhantes, a informação relevante do sistema central, que é necessária para garantir o correto funcionamento das funcionalidades deste componente, é obtida inicialmente através do mesmo SNMP *relay* e atualizada sempre que for atualizada centralmente.

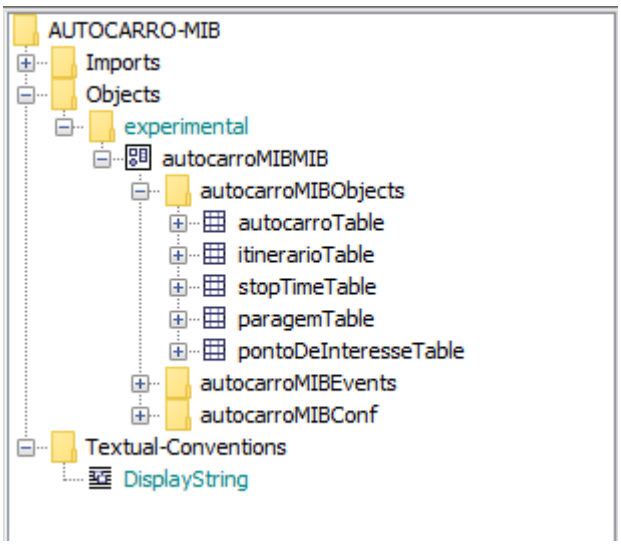


Figura 20 - Estrutura da MIB do componente veicular

Assim, a base de dados necessária no autocarro, em termos de definição dos objetos de gestão incluídos, assemelha-se bastante à que está presente na central, contendo apenas as

tabelas relevantes para a implementação das funcionalidades deste componente (ver Figura 20). A MIB implementada no agente SNMP tem o sufixo ***experimental.9999*** (*1.3.6.1.3.9999*).

A estrutura mais detalhada de cada tabela desta MIB está documentada no Anexo III – MIBs do Sistema (para acesso à listagem em SMIv2/ASN.1 por favor contactar o autor).

Em relação ao esquema ER de representa esta base de dados, é em tudo semelhante ao esquema apresentado para a base de dados do sistema central (ver Figura 19), menos as tabelas que não estão agora presentes, pelo que não é apresentado explicitamente neste documento.

Base de dados componente de interface do utilizador

No caso do passageiro, os dados da aplicação necessários para implementar as suas funcionalidades são obtidos através de comunicação SNMP com o agente do sistema central do operador respetivo. Os dados são depois guardados numa base de dados local por forma à aplicação poder funcionar mesmo sem a comunicação com o sistema central ser possível, ainda que, neste caso, a funcionalidade de monitorização em tempo real não esteja disponível.

O esquema concetual ER da base de dados deste componente é em tudo semelhante ao esquema das duas bases de dados anteriores, tendo em consideração que nem todas as tabelas são incluídas.

O modelo de desenvolvimento utilizado em todas as etapas do projecto foi o modelo em cascata, também denominado de *waterfall model*. É um método puramente sequencial, conforme pode ser confirmado pela Figura 21, e que pode ser reiniciado quantas vezes a fase de testes o exigir.

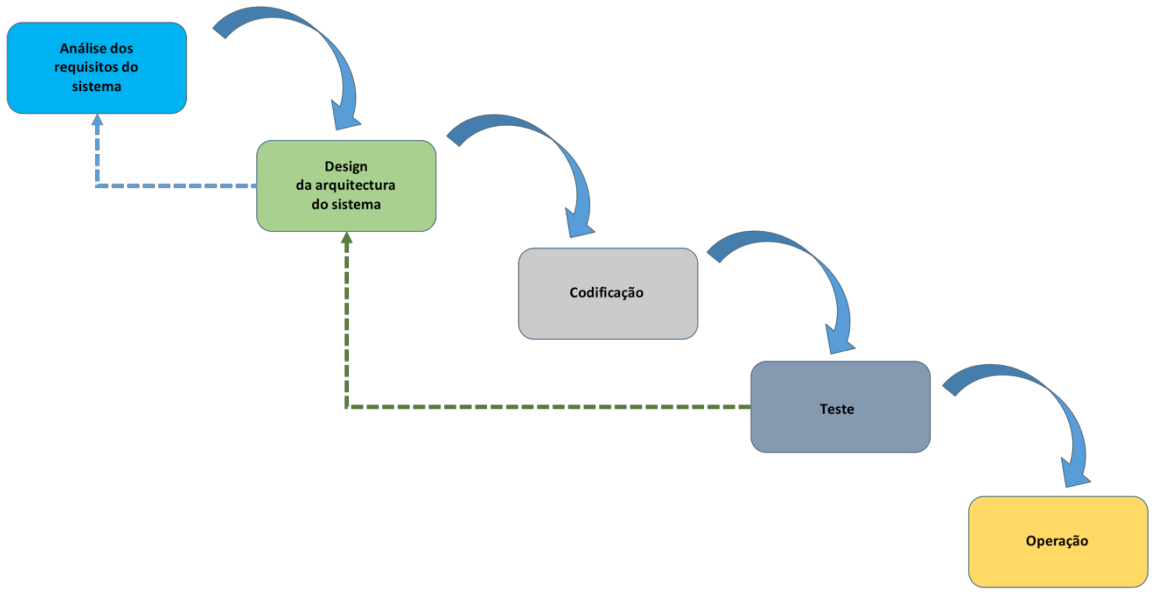


Figura 21 - Modelo de desenvolvimento do sistema

Todos os componentes do sistema – administração, autocarro e passageiro – seguiram este modelo de desenvolvimento. Depois da análise e definição detalhada dos requisitos do sistema, prosseguiu-se para o *design* da arquitetura e base de dados do mesmo. Após essa etapa estar concluída, seguiu-se a codificação e teste do sistema propriamente dito, inicialmente em isolado e depois em conjunção com os outros componentes já prontos. Sempre que durante o teste dos componentes ocorreram erros importantes teve que se voltar novamente à verificação do modelo do sistema, o que, dependendo da complexidade das funcionalidades, é praticamente inevitável neste método de desenvolvimento. Só após todo o sistema ser devidamente testado e validado é que o sistema pode entrar em operação. A exigência nos testes de validação está dependente do objetivo último do sistema a construir.

Um dos problemas com este modelo de desenvolvimento é o risco de se fazer um planeamento extenso e demasiadamente detalhado logo à partida. Além de ser demorado apresenta uma probabilidade enorme de deteção de falhas durante a fase de testes do projecto, o que obriga a que se tenha de voltar às etapas iniciais para resolver os erros encontrados.

Como seria de esperar, a construção dum sistema protótipo num projeto de índole académica não tem o mesmo nível de exigência em termos de validação do funcionamento do sistema que a validação dum sistema para operação no mundo real em contexto empresarial.

Assim, e no contexto de planeamento e desenvolvimento dum projeto académico desta dimensão por uma única pessoa, verificou-se que a escolha deste método de desenvolvimento foi adequado.

5.1 Arquitetura de implementação

No processo de desenvolvimento, o modelo anteriormente definido foi implementado de forma sistemática utilizando o mesmo tipo de módulos de software para todos componentes do sistema, conforme pode ser analisado na Figura 22.

De qualquer forma, a Figura 23 vem detalhar a arquitetura especial do caso de implementação do componente veicular em que existem um gestor SNMP *relay* e uma MIB local.

De notar que, a instrumentação SNMP para implementação interna/local das MIBs foi feita com recurso à tecnologia de bases de dados SQL.

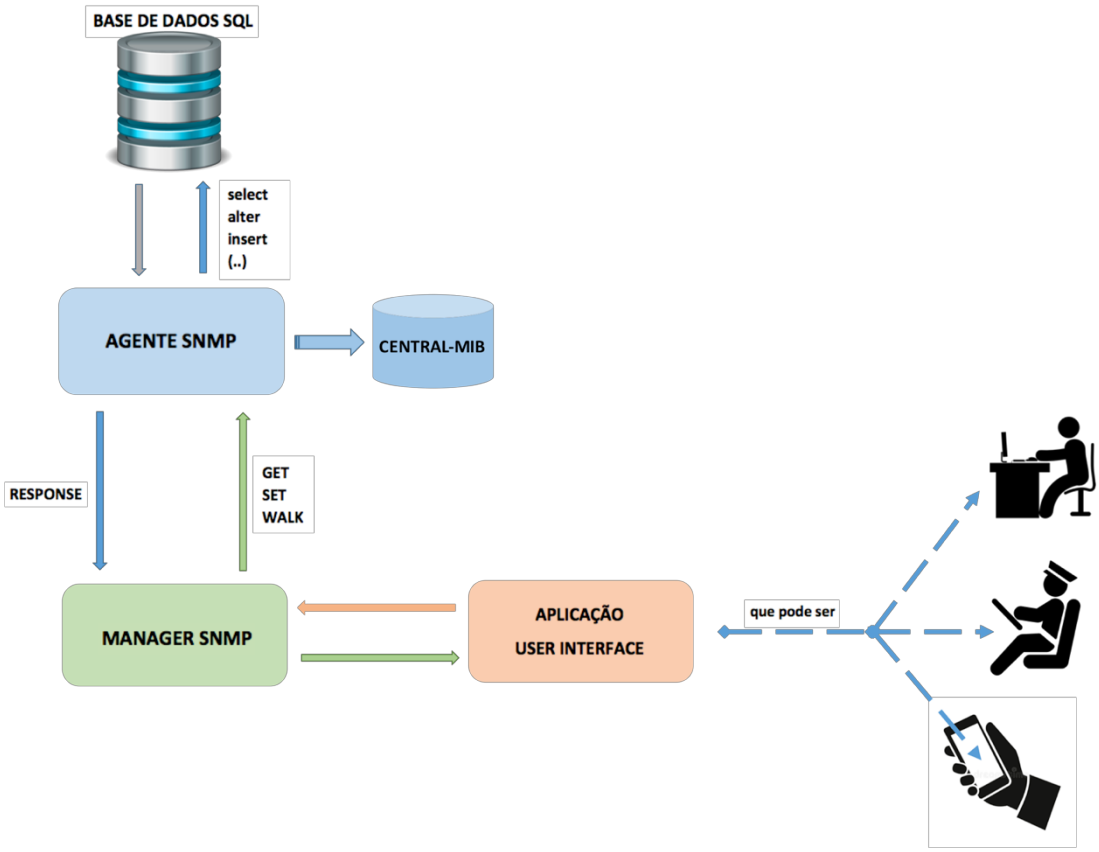


Figura 22 - Arquitetura geral de implementação

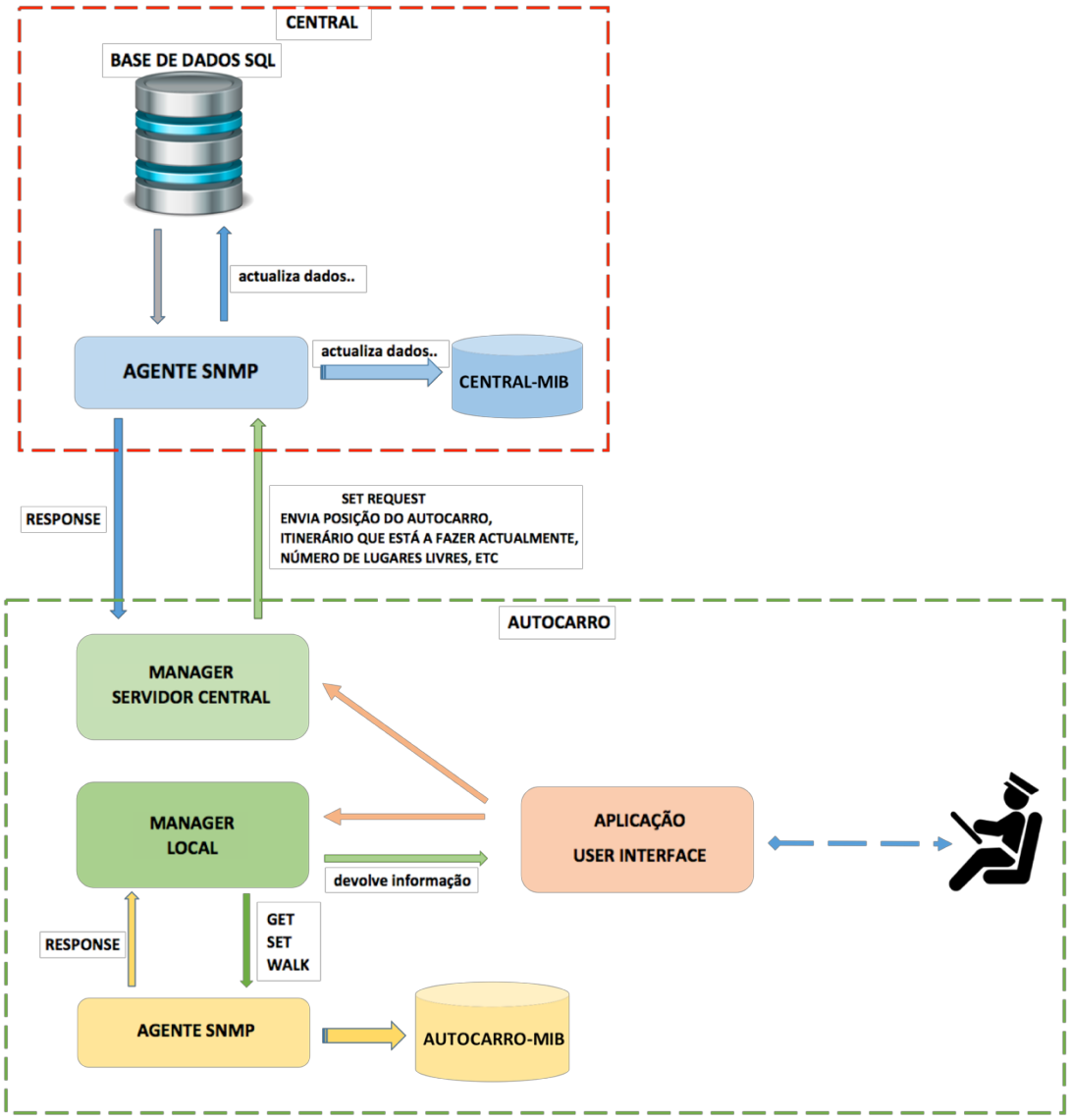


Figura 23 - Arquitetura de implementação no componente veicular

Os detalhes mais relevantes de implementação de cada um dos componentes, que têm a ver com os módulos gestores e agentes SNMP e das próprias MIBs, são descritos no Anexo IV – Instrumentação SNMP. De realçar que a instrumentação SNMP foi realizada com a ajuda da API de programação em Java denominada SNMP4J. Esta API é de uso gratuito e um software *opensource*. Infelizmente, e apesar de ser muito utilizada no desenvolvimento de protótipos em contexto académico, a documentação desta API é inconsistente, o que trouxe algumas dificuldades adicionais ao desenvolvimento destes módulos.

5.2 Testes funcionais

Ao longo desta secção irão ser apresentadas capturas de ecrã realizadas durante os testes funcionais dos vários componentes do sistema na versão final do protótipo.

Para testar e validar convenientemente as funcionalidades do sistema, integrou-se no sistema a informação completa do operador de transporte públicos da cidade de Braga (TUB). A informação incluída nas bases de dados foi obtida diretamente pela empresa responsável por gerir o serviço com a autorização oficial dos serviços da Câmara Municipal de Braga.

Na versão final do protótipo do sistema completo, que integra instanciações de todos os componentes do modelo apresentado anteriormente, todas as funcionalidades previstas e implementadas estavam a funcionar correctamente, tendo em consideração metas de validação em acordo com os objetivos principais do projeto e os requisitos funcionais definidos.

Administração do sistema central

O administrador pode não se encontrar na mesma máquina ou até na mesma rede que o servidor central (e.g. pode querer fazer atualizações remotamente), por isso, no arranque interface do sistema de gestão, o administrador deve indicar qual o endereço do servidor e qual a porta onde o agente está à escuta.

Figura 24 - Estabelecimento de conexão ao servidor central

Note-se que o administrador tem que indicar uma porta e endereço válidos, ou o estabelecimento de conexão irá falhar. O sistema não permite introduzir qualquer tipo de dados inválidos, quer seja na inserção ou alteração de dados.

Após estabelecer a ligação ao servidor com sucesso, o administrador entra no menu principal do sistema.

Figura 25 - Menu principal do sistema de administração central

Neste menu, é possível visualizar, alterar, criar e remover dados de todas as tabelas.

A Figura 26 exemplifica uma listagem total (não filtrada) que pode ser utilizada para efeitos de *debug* da informação da base de dados.

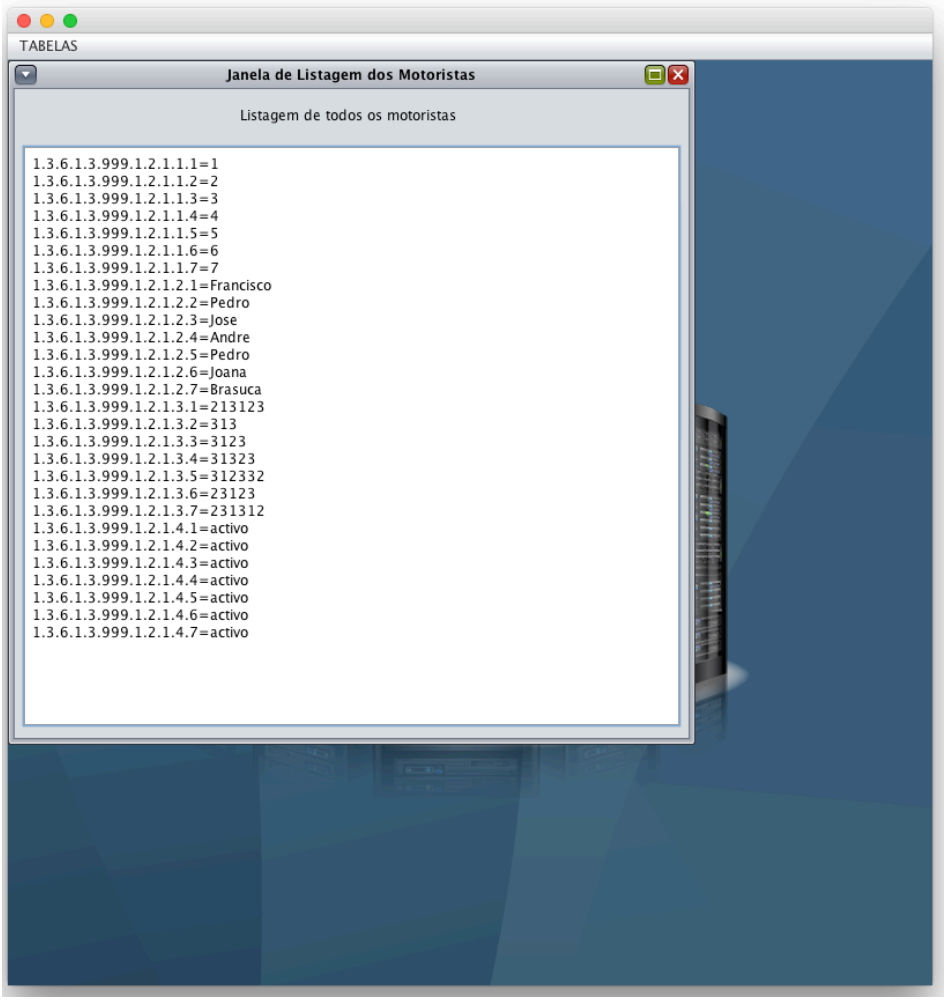


Figura 26 - Exemplo de listagem total duma tabela da MIB

Nesta opção do menu, o resultado aparece propositadamente sob a forma de uma lista sequencial e ordenada de OIDs + valor.

Existem opções para inserção de dados novos na MIB (ver exemplo de inserção de informação sobre uma nova paragem na Figura 27).

Se o administrador quiser visualizar dados de algum motorista em específico, ou um itinerário ou autocarro, etc., apenas tem que indicar o *id* do elemento em causa (ver exemplo

da Figura 28). De realçar que neste protótipo, a obtenção de informação por mecanismos de filtragem mais elaborados não foram contemplados.

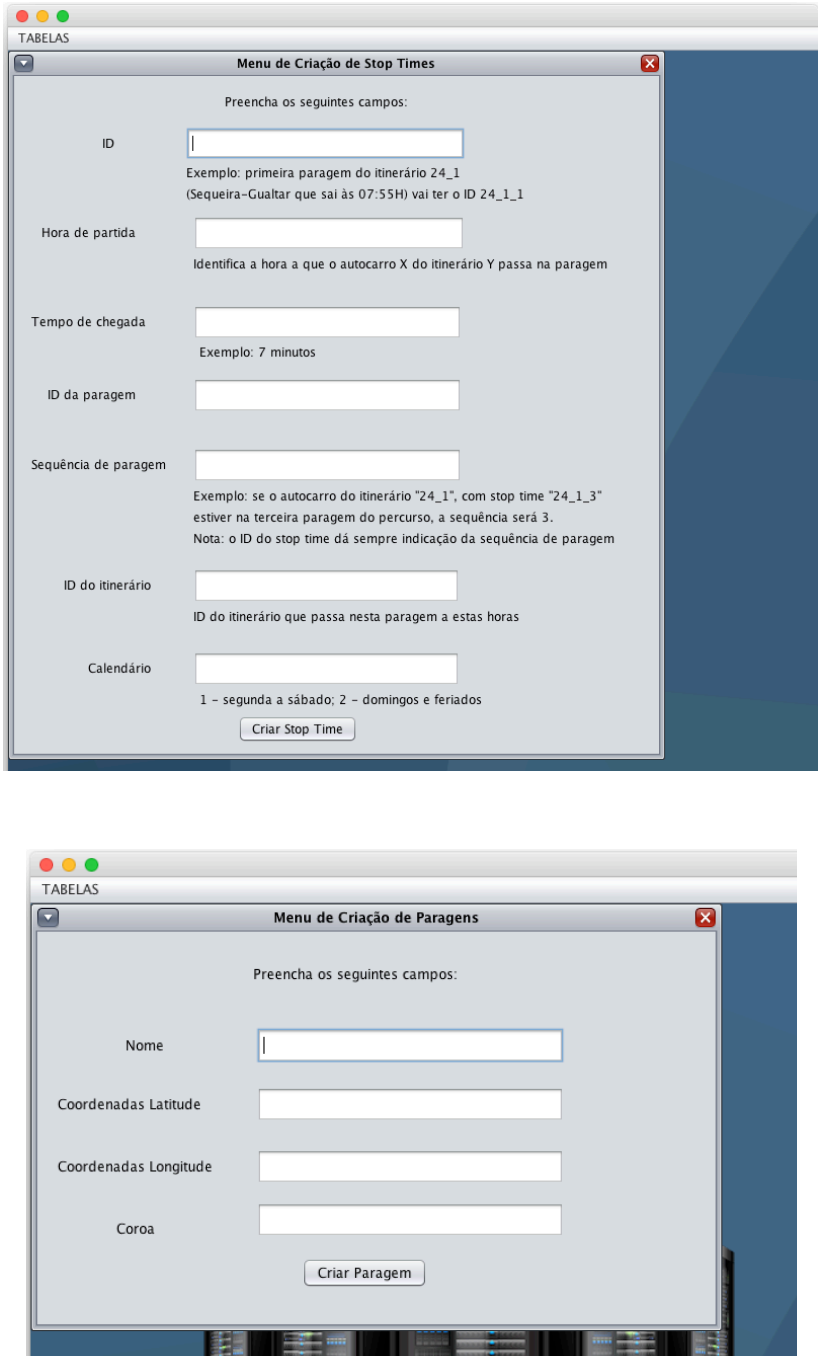


Figura 27 - Opção de inserção de informação na MIB

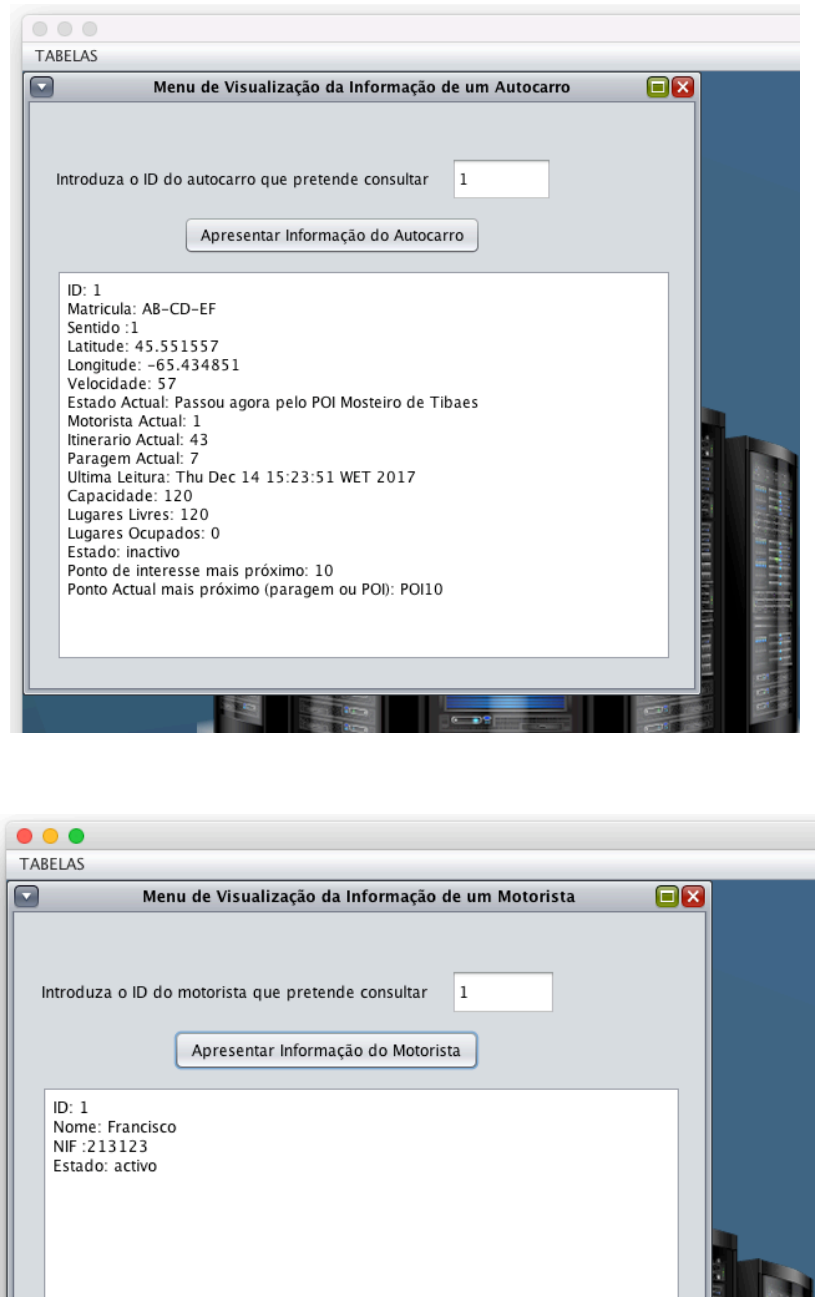


Figura 28 - Obter informação sobre um elemento da base de dados

Para alterar dados das tabelas da MIB, o administrador terá que indicar o *id* da entrada que deseja alterar (ver exemplo na Figura 29).

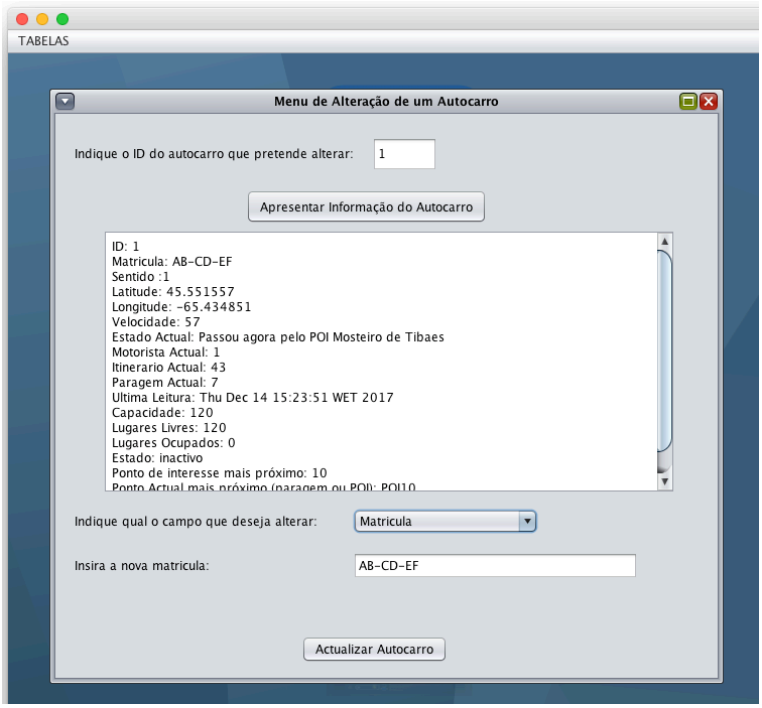


Figura 29 - Modificação de dados na MIB

Para garantir que os dados foram alterados com sucesso, é feito automaticamente o *reload* da interface a partir de um novo pedido *GET* ao agente. Desta forma, temos a certeza que do lado da central os dados foram definitivamente alterados, pois não existem dados guardados localmente no sistema de administração.

Tal como as outras operações sobre itens individuais da tabela, a remoção dum elemento da MIB é feita através dum opção do menu de administração (ver exemplo na Figura 30). Relembre-se que todas as funcionalidades são implementadas com recurso a operações protocolares SNMP.

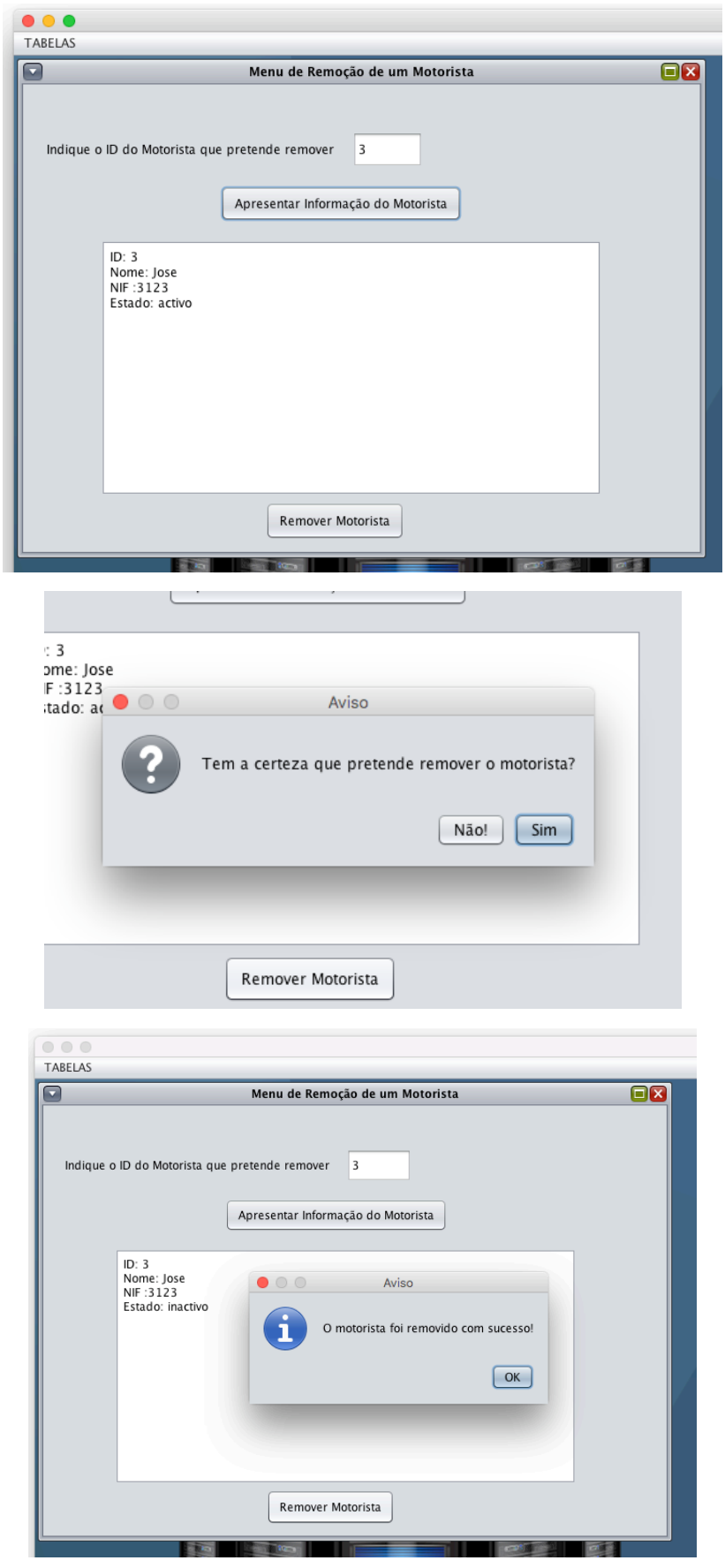


Figura 30 - Operação de remoção dum elemento da MIB

Componente veicular

Conforme foi referido anteriormente, no contexto deste projeto, o componente veicular implementado é um subsistema para integração dum autocarro de passageiros no sistema. A gestão local deste subsistema é responsabilidade do motorista ou duma pessoa com funções de gestão no interior do veículo (por exemplo, um revisor).

Assim que o motorista pretende iniciar uma viagem, ao aceder ao sistema é-lhe requisitado o seu *id*, que tem que ser válido (ver exemplo na Figura 31). Após efectuar a identificação com sucesso no sistema, o motorista tem que escolher um itinerário. Caso exista ligação internet e a *Autocarro-MIB* se encontre vazia, a informação é obtida por SNMP a partir da central. Depois desta operação, tanto a MIB local como a MIB da central passam a conter a informação do motorista que está a conduzir este autocarro.

De realçar que, neste protótipo, não existem mecanismos de autenticação, apenas de identificação, dos vários utilizadores dos componentes do sistema, uma vez que, por razões de limitação temporal, a implementação de mecanismos de autenticação e confidencialidade das operações nos vários interfaces não foi definida como objetivo do projeto.

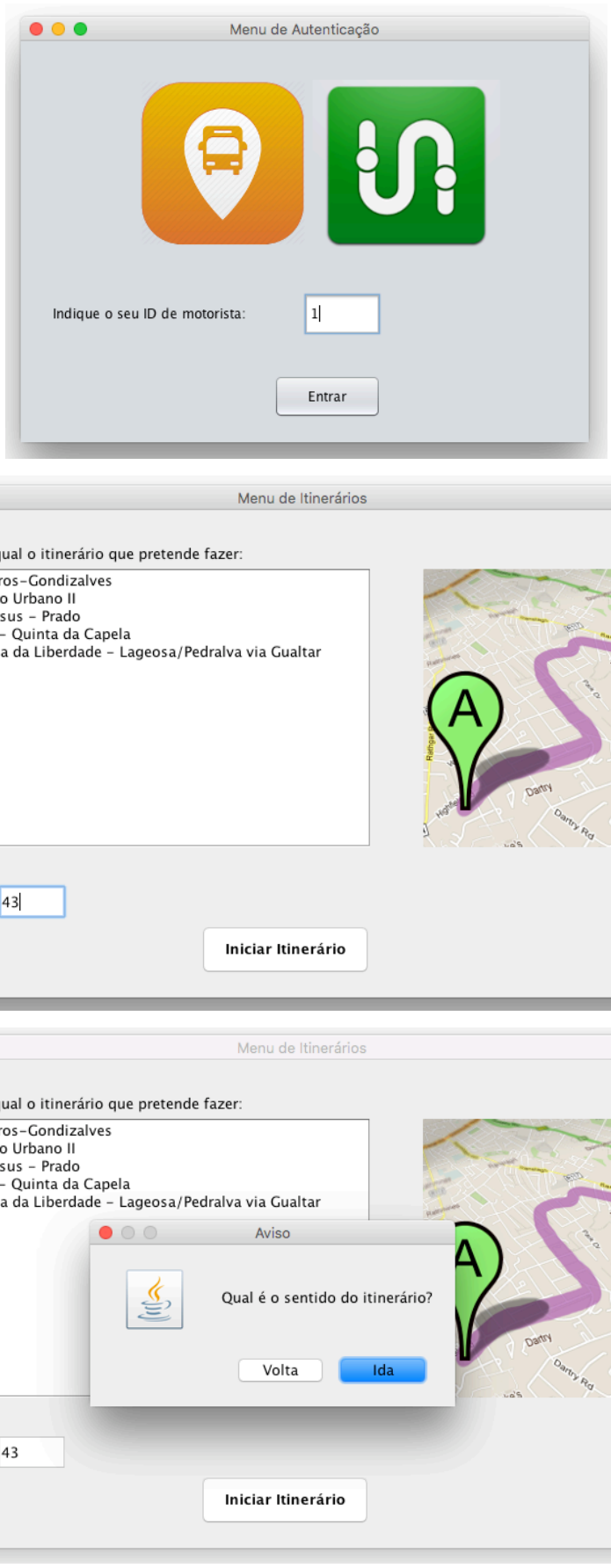


Figura 31 - Identificação do motorista e escolha de itenerário

Gestão do percurso

Após a escolha do itinerário e o autocarro iniciar a sua viajem, o motorista pode ir selecionando as paragens ou pontos de interesse por onde passa, registar entradas ou saídas do autocarro e ver os dados tem tempo real da sua viagem (ver exemplo na Figura 32). Integrando um sistema AVL esta função seria realizada sem a intervenção direta do motorista ou revisor.

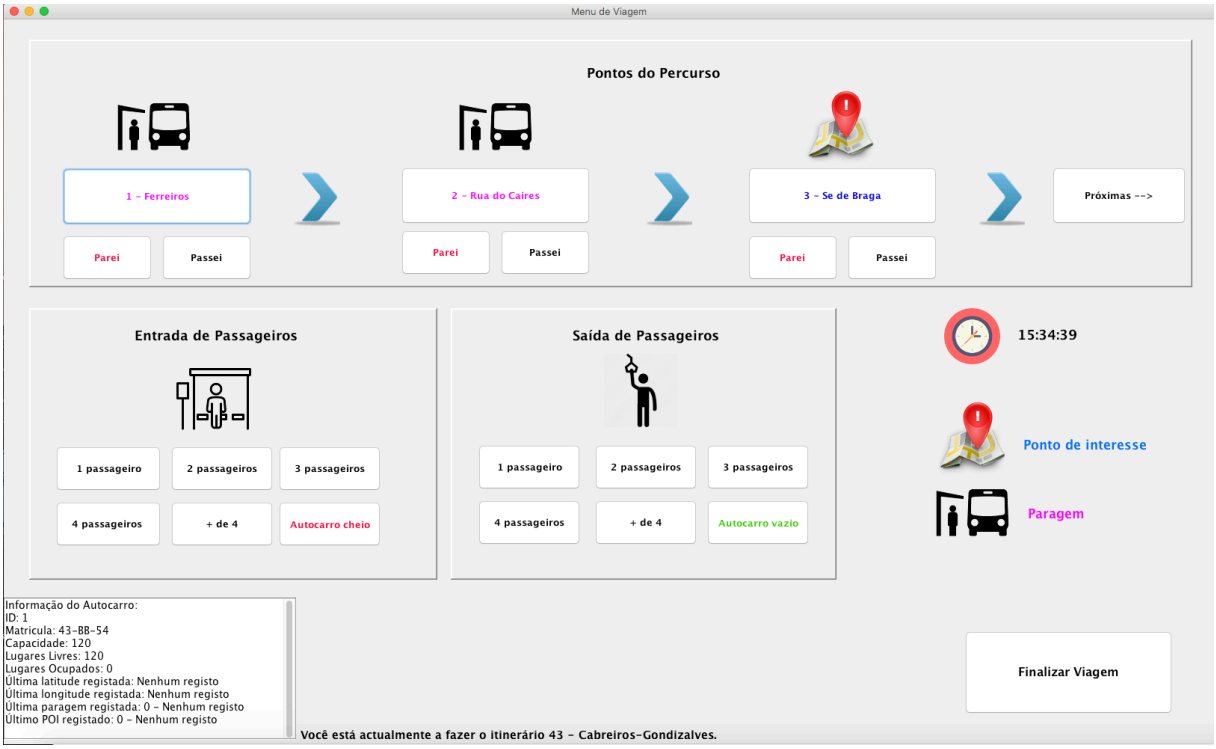


Figura 32 - Exemplo de gestão de percurso num autocarro

No canto inferior esquerdo, além da informação do itinerário que está a fazer atualmente, o motorista pode ver informações da posição do veículo em tempo real. As informações de posicionamento do veículo vão sendo atualizadas à medida que a viagem decorre. O motorista pode ir introduzindo no sistema a informação referente ao número de passageiros que entra e sai nas paragens, os pontos de interesse por onde passa ou que estão mais próximos, etc.

Para registar a passagem nas paragens ou pontos de interesse próximos, o motorista tem que ir selecionando uma das opções "*Parei*" ou "*Passei*". A sequência do ponto do percurso é sempre apresentada. As paragens e pontos de interesse são atualizadas dinamicamente na

interface. Uma paragem é representada por uma imagem e o ponto de interesse por outra, tal como está indicado na legenda.

Para que a central saiba onde os autocarros se encontram, é necessário que os motoristas registem a passagem ou paragem pelos diferentes pontos do percurso. No entanto, não é obrigatório que o motorista registe todas as passagens. Por exemplo, se o motorista indica que passou na terceira paragem do percurso, então o sistema assume automaticamente que o veículo já passou na primeira e na segunda.

Por vezes, os autocarros podem ficar parados durante algum tempo numa paragem pelo que dizer somente que passou pela paragem acaba por ser insuficiente no que toca a saber a localização geográfica do autocarro. No sentido de melhorar este mecanismo, o motorista pode indicar se está parado ou não numa paragem ou perto de um ponto de interesse. Depois, no momento de arrancar, seleciona a opção de "*Arrancar*" (ver exemplo na Figura 33).

Figura 33 - Opção de arrancar duma paragem

Enquanto está parado, o motorista pode registar entrada ou saída de passageiros. Quando arranca daquela paragem, essa informação é registada na MIB local e depois enviada para a central, caso exista ligação internet no autocarro.

À medida que a viagem vai decorrendo, o motorista pode também registar a entrada ou saída de passageiros. O sistema faz a gestão automática do número de lugares ocupados e livres, tendo em consideração a lotação do veículo e as indicações do motorista ou revisor. A

todo o momento o motorista também pode verificar na janela de informações o número de lugares livres ou ocupados do autocarro (ver exemplos de mensagens de erro e janela informativa de lugares na Figura 34).

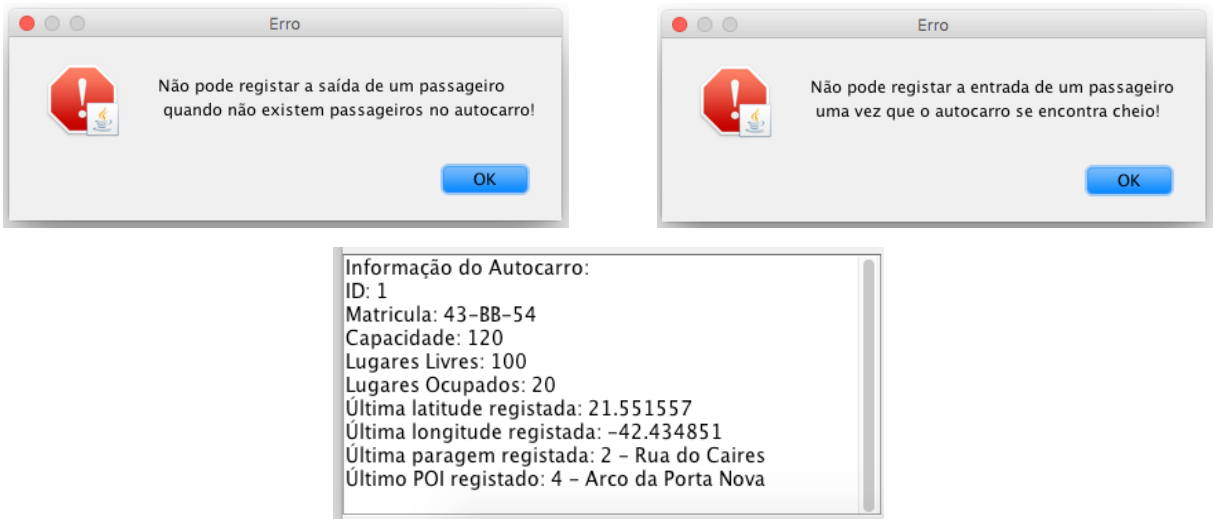


Figura 34 - Exemplo de gestão dos lugares livres/ocupados no veículo

Para indicar o final da viagem, o motorista apenas tem que selecionar a opção "*Finalizar*" e o relatório de viagem é-lhe apresentado (ver exemplo na Figura 35).

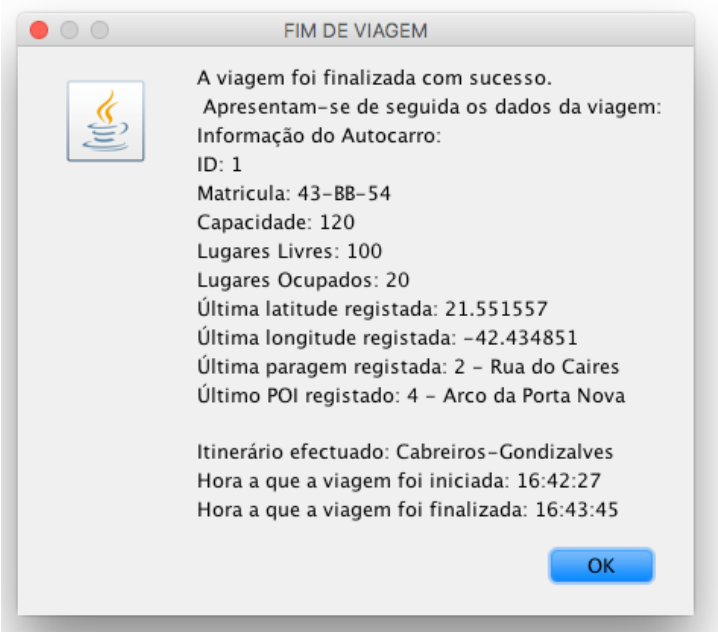


Figura 35 - Exemplo de relatório de viajem

Aplicação do passageiro

Na aplicação de interface com o passageiro (ou potencial passageiro) foi implementado um gestor SNMP para uma plataforma móvel baseada em *Android* para fazer a troca de informações com o agente SNMP que corre no servidor central. Toda a informação é mantida localmente numa base de dados SQLite.

A aplicação foi desenvolvida em linguagem nativa de modo a não existirem riscos de degradação de desempenho, ainda que a optimização de desempenho não fosse um dos objetivos do projeto.

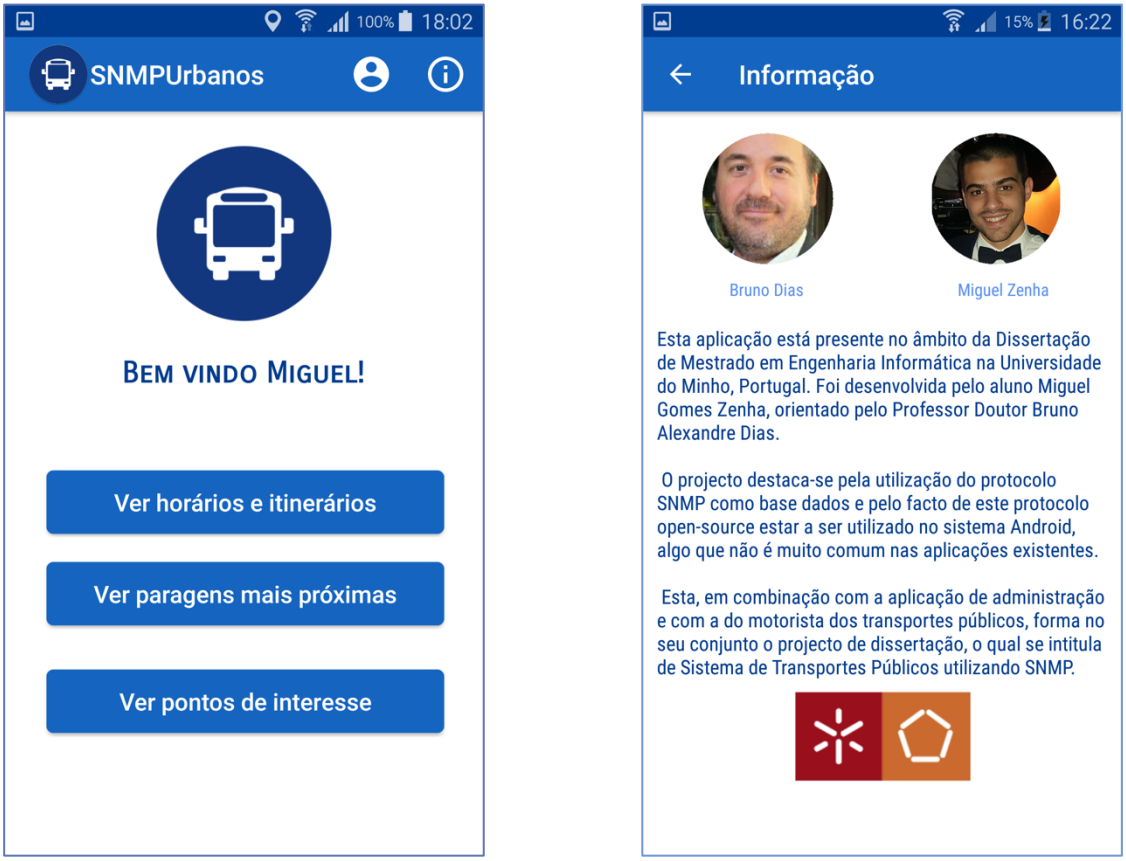


Figura 36 - Arranque da aplicação do passageiro

Depois de instalada, na primeira vez que a aplicação é executada (ver exemplo de arranque da aplicação na Figura 36), os dados necessários para o seu funcionamento correto são obtidos por SNMP consultando a MIB do sistema central. Os dados são então armazenados localmente, sendo a sua validade verificada regularmente. Sempre que novos dados são detetados no sistema central estes são atualizados localmente.

Caso o utilizador não possua o GPS ou a internet ligada, surgem avisos a aconselhá-lo para ligar os serviços que o levam directamente às definições do telemóvel, caso este aceite (ver Figura 37). Sem acesso à internet ou localização GPS certas funcionalidades não ficam disponíveis.

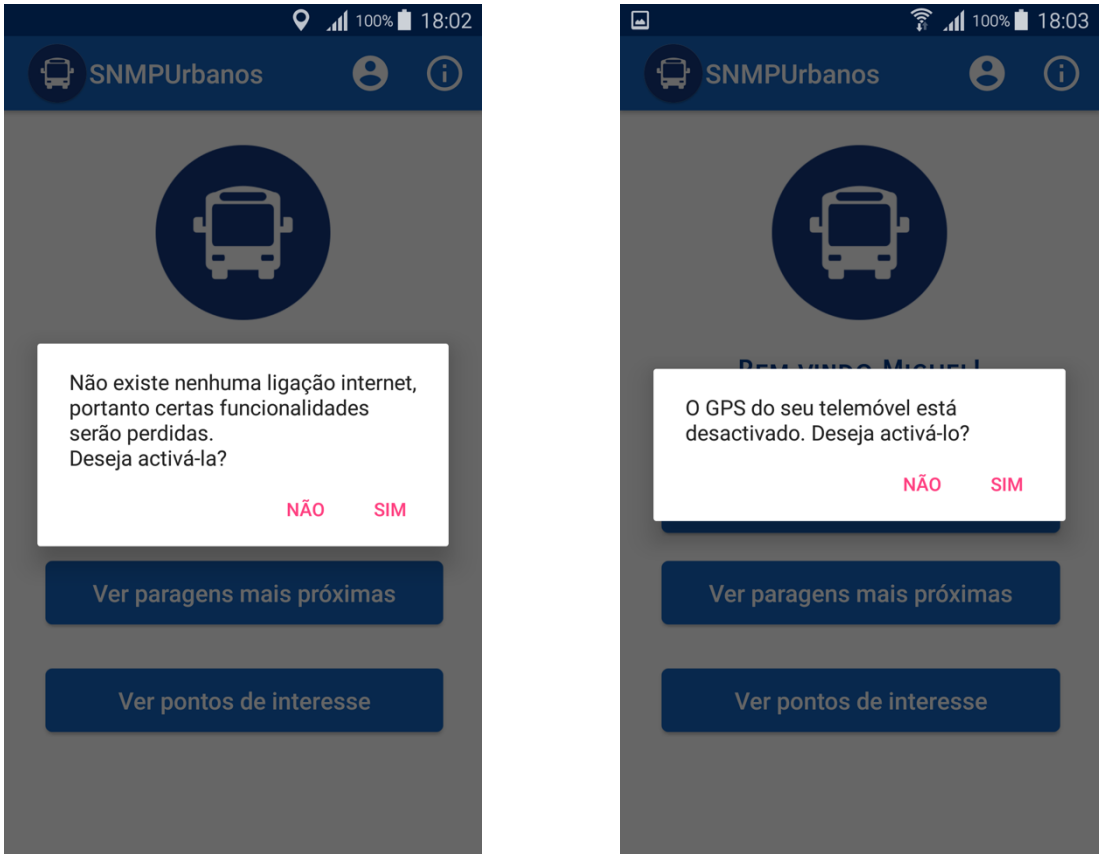


Figura 37 - Mensagens de aviso na aplicação do passageiro

Horários e itinerários

O utilizador pode ver os horários e itinerários disponíveis pelo operador na região/cidade onde se encontra. Ao visualizar os dados dum itinerário, o horário mais próximo da hora atual aparece em destaque. Após escolher o horário, os detalhes desse horário são apresentados e o utilizador pode optar por ver a lista de paragens do itinerário ou ver o percurso do itinerário no mapa. Ao longo do percurso de um itinerário, um autocarro pode passar perto de pontos de interesse de uma dada cidade.

Assim sendo, além de ver as paragens de um itinerário, o utilizador pode ver também quais os pontos de interesse desse percurso (ver Figura 38).

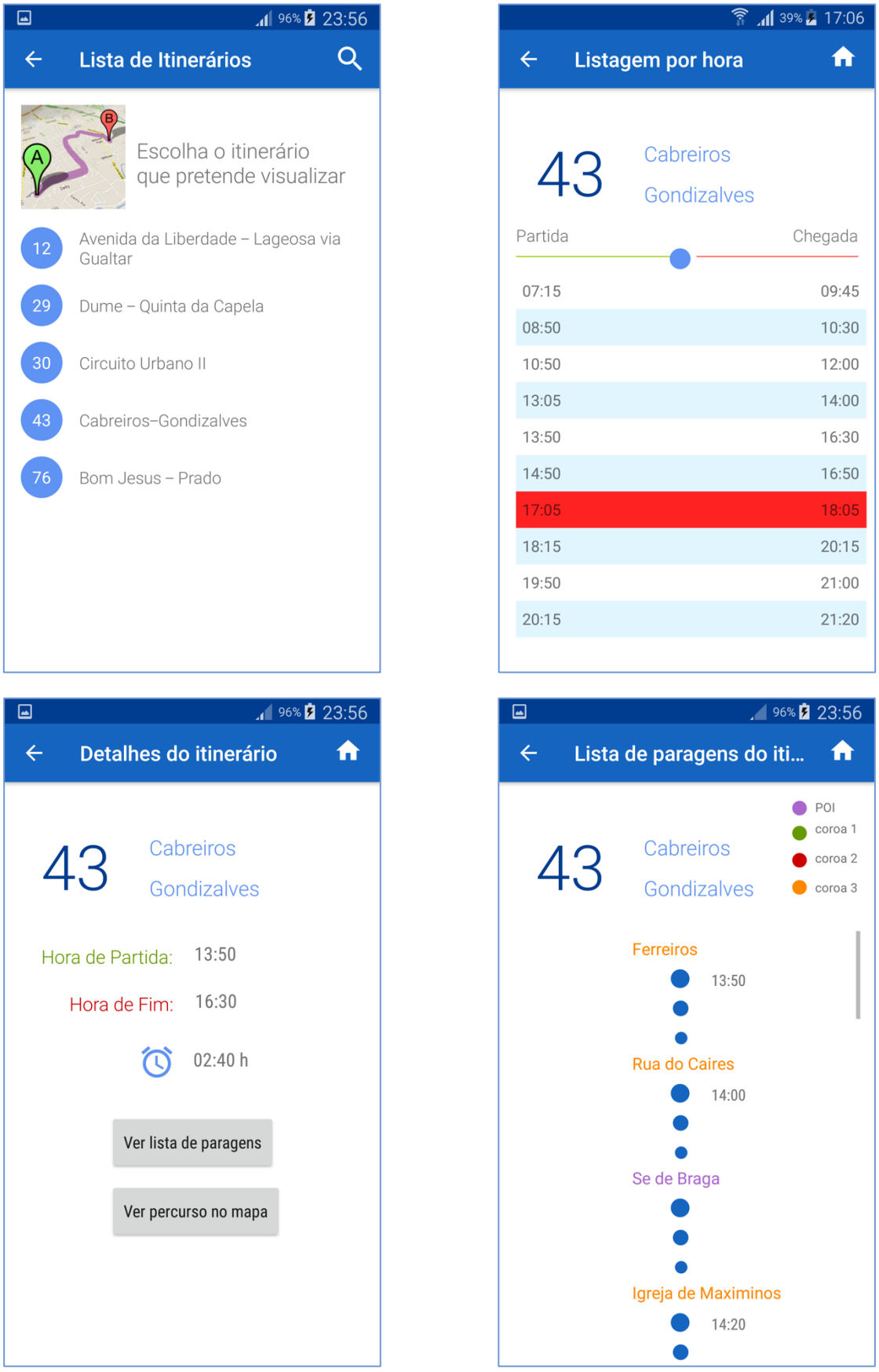


Figura 38 - Visualização da informação dos itinerários

Percursos e paragens

Ao escolher ver o percurso dum itinerário num mapa, são apresentados todos os pontos do itinerário em questão num mapa geográfico simbólico.

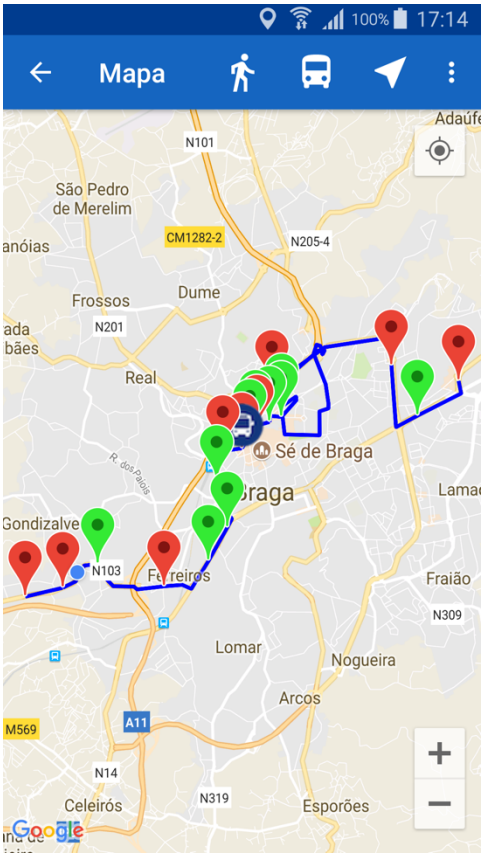


Figura 39 - Percurso dum itinerário no mapa

O utilizador consegue visualizar no mapa todo o percurso do itinerário, passando por todas as paragens e pontos de interesse. As paragens encontram-se identificadas pela cor vermelha e os pontos de interesse pela cor verde (ver exemplo na Figura 39).

O utilizador pode optar por ver o caminho para a paragem mais próxima ou pedir à aplicação que lhe indique onde se encontra o autocarro em tempo real. A aplicação recebe a informação de monitorização em tempo real do agente SNMP do sistema central. Ao selecionar os pontos do percurso, o utilizador encontra os detalhes desse ponto que tiverem sido introduzidos na base de dados central pelo operador de transporte.

Caso o utilizador opte por ver as paragens mais próximas, a paragem geograficamente mais próxima de si é apresentada no mapa, sendo possível ver o caminho para essa paragem. Selecionando uma determinada paragem são apresentadas informações adicionais, incluindo os itinerários que passam na paragem em questão (ver exemplo na Figura 40).

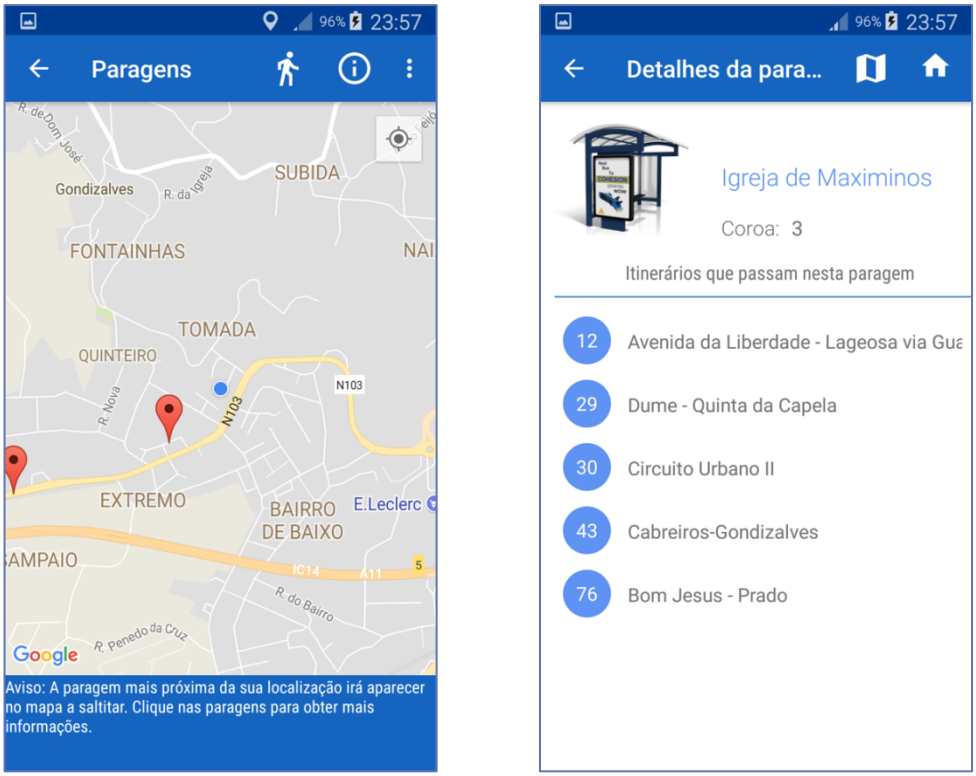


Figura 40 - Indicação das paragens mais próximas

Ao optar por ver os pontos de interesse, o utilizador acede à lista de pontos de interesse perto do local onde se encontra ou das paragens do percurso dos itinerários (ver Figura 41).

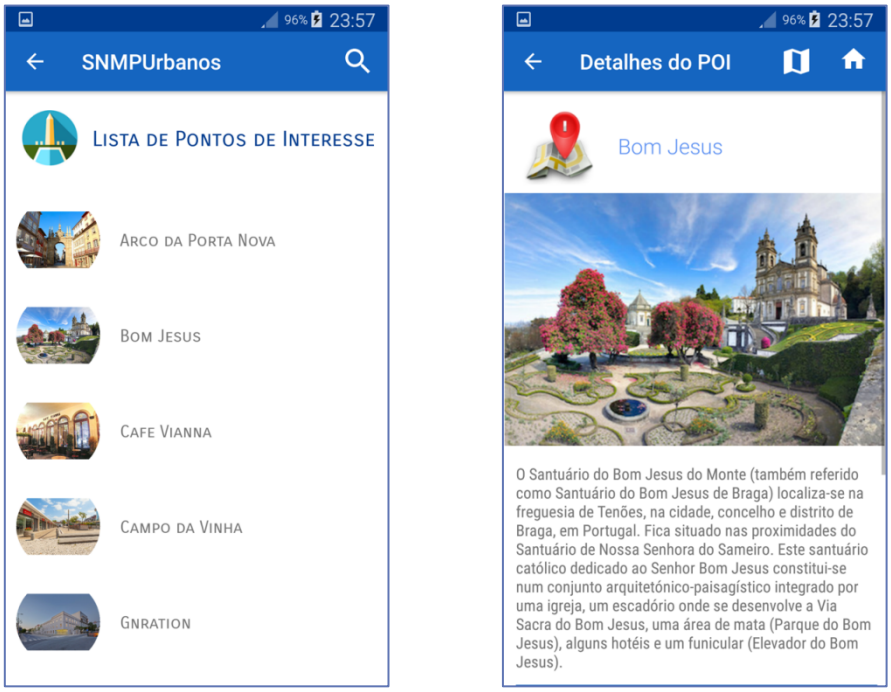


Figura 41 - Exemplo da informação sobre pontos de interesse

6. CONCLUSÕES

Neste projecto foram analisadas as principais soluções existentes no mercado e as tecnologias relacionadas com serviços de informação para apoio ao utilizador de transportes públicos. Em parte das aplicações estudadas, as possibilidades de integração com outros sistemas são inexistentes ou implicariam processos eventualmente complexos e/ou com custos elevados para os operadores do ramo.

As soluções atuais enquadram-se em dois tipos de abordagem bastante diferentes.

Uma que pretende ser genérica, passível de ser utilizada em diversas regiões ou cidades do mundo e integrando informação de vários operadores de transportes dessas regiões. Este tipo de solução obriga a que os responsáveis da aplicação mantenham uma base de dados o mais atualizada possível com as informações dos operadores de transporte de todas as regiões/cidades suportadas. Este processo de obtenção da informação dos operadores é, em geral, dependente de processos manuais de cedência de informação em formatos especificados para cada uma das aplicações em que quiserem estar suportados. Quando é possível a integração automática da informação os operadores têm que utilizar uma API própria da aplicação. Em qualquer dos casos, os operadores, não controlam/gerem diretamente a informação contida em cada uma das aplicações em que querem estar suportados. Além disso, em geral, as aplicações genéricas utilizam mecanismos baseados em *crowdsourcing* para implementar a monitorização em tempo real dos serviços. Este tipo de mecanismos, que está dependente dum número elevado de passageiros voluntários, tende a ser menos fiável que os mecanismos AVL implementados no lado dos operadores.

A outra abordagem comum é a implementada em aplicações mais específicas, em que o seu desenvolvimento é responsabilidade dum operador ou pequeno grupo de operadores de transporte do mesmo grupo empresarial. Estas aplicações suportam apenas esses operadores e só podem ser usados numa região ou cidade. Não existe integração com outros operadores de transporte, o que pode ser uma desvantagem relevante em regiões/cidades de maior

dimensão com vários operadores de transporte em atividade. Por outro lado, as soluções aplicacionais podem tirar maior partido de funcionalidades específicas a um determinado operador ou contexto geográfico, havendo assim uma maior diferenciação dos mesmos. Além disso, os mecanismos AVL implementados pelos operadores costumam ser mais precisos e fiáveis que os mecanismos de monitorização implementados nas aplicações de abordagem genérica.

Como forma de melhor conhecer os vários tipos de mecanismos de monitorização do estado dos serviços, como por exemplo a posição geográfica dos veículos de transporte e a estimativa de tempos de espera, foram estudadas as principais tecnologias usadas na implementação de mecanismos AVL.

Atendendo às limitações dos dois tipos principais de modelos estudados, definiu-se uma proposta de arquitetura com uma estratégia intermédia e que permita não só integrar vários operadores de transporte mas também suportar qualquer região/cidade de qualquer dimensão. Além disso, o modelo proposto permite que os operadores de transporte mantenham controlo direto e total da informação dos seus serviços, utilizando localmente o tipo de tecnologias que melhor se adeque às sua necessidades, incluindo mecanismos mais ou menos avançados de monitorização do serviço em tempo real.

A arquitetura modular definida também permite a integração de vários tipos de componentes, incluindo componentes dedicados à monitorização dos serviços de transporte (independentemente das tecnologias de suporte serem baseadas em AVL, *crowdsourcing* ou até com intervenção manual de algum tipo de operador humano como um motorista ou um revisor), componentes de interface de administração do sistema de cada operador e aplicações dedicadas aos utilizadores/passageiros que são a parte do modelo que tem visibilidade pública. O desenvolvimento destas aplicações e dos outros componentes é completamente independente entre si, podendo ser implementadas e disponibilizadas por entidades diferentes.

A escolha do paradigma normalizado SNMP, como tecnologia protocolar para a comunicação entre componentes e como tecnologia de representação da informação em MIBs, permite esta modularização e independência entre componentes.

Para testar a validade do modelo proposto, foi desenvolvido um sistema protótipo com todos os tipos de componentes relevantes nestes serviços. O protótipo foi alimentado com a informação completa e oficial do serviço de transportes públicos da cidade de Braga. Para além do módulo de interface para a administração do sistema de bases de dados central do operador de transporte, também foi incluído um componente de monitorização do serviço, implementado num módulo que reside nos veículos de transporte e que necessita da intervenção manual do motorista ou revisor. Por fim, foi desenvolvida uma aplicação móvel para os utilizadores/passageiros na plataforma *Android*.

O sistema foi sendo testado e corrigido durante alguns meses. A versão final do protótipo foi considerada funcionalmente correta em todas as funcionalidades testadas e desenvolvidas, validando assim o modelo proposto e a utilizando do paradigma SNMP como tecnologia base de integração normalizada de todos os componentes.

Como trabalho futuro listam-se algumas sugestões para melhoria do trabalho desenvolvido:

- Refinamento do modelo proposto numa forma mais universal e concetual e em que o aspeto da integração de vários operadores esteja melhor definido;
- Modificar o componente de monitorização do serviço desenvolvido para o protótipo para que possa ser utilizado em qualquer tipo de veículo de transporte;
- Desenvolvimento dum protótipo dum sistema AVL para complementar o sistema de monitorização já implementado;
- No protótipo, otimizar algumas operações de obtenção de informação das MIBs com primitivas SNMP GETBULK;
- Melhorar algumas funcionalidades do protótipo da aplicação móvel para os utilizadores;
- Adicionar ao protótipo uma aplicação com interface web;

- Integrar no sistema protótipo, um ou mais operadores noutras cidades de maior dimensão, como por exemplo, do Porto ou de Lisboa.

ANEXO I – Tecnologias AVL

Neste anexo estão analisadas algumas estratégias e principais tecnologias utilizadas em sistemas AVL e são discutidas as suas vantagens e desvantagens relativas.

A. RFID

Hoje em dia, um dos conceitos mais utilizados para localizar veículos consiste em colocar sensores com capacidade de comunicação em determinados locais fixos, ou em veículos. O veículo, ao entrar na área de alcance desses sensores, é identificado e essa informação é transmitida para um servidor. O conceito destes métodos de localização de veículos é utilizado principalmente para a localização de veículos que obedecem a uma rota fixa (e.g. comboios ou autocarros). Em relação às tecnologias utilizadas na implementação deste conceito, são na grande maioria das vezes colocadas *tags* RFID[22][23] em pontos fixos (e.g. paragens de autocarros, postes de semáforos, postes de electricidade, etc.), e instalados leitores de RF nos veículos. Ou vice-versa, o veículo possui o identificador RFID e os postos fixos possuem o leitor, estando esta decisão dependente de questões financeiras.

O RFID é uma rede de comunicação à distância sem fios, que funciona através da identificação de radiofrequências com alcance em torno dos 10 metros, isto dependendo da etiqueta utilizada. A comunicação ocorre através de uma etiqueta com *chip* RFID - a que chamamos de *tag* RFID - e que envia sinais para um leitor específico. A partir daí, um software é responsável pela conversão dos dados em informações significativas.

A tecnologia RFID pode ser utilizada com o objetivo de identificação ou rastreamento de objetos, aplicações do sector logístico, supermercados, transporte ou cargas. Apenas é necessário que o objeto possua a etiqueta RFID e desta forma os dados podem ser capturados pelo leitor, mesmo que os objetos estejam em movimento.

Funcionamento das Tags RFID

Uma *tag* RFID possui dois modos de funcionamento:

Activo: uma tag activa possui uma fonte de alimentação através de uma bateria, e possui um alcance de comunicação de cerca de 10 metros, sendo este tipo de *tag* capaz de enviar dados a um leitor por

conta própria. Uma *tag* RFID activa possui um tamanho consideravelmente maior quando comparada a uma *tag* passiva.

Exemplo de Tag RFID Activa. [25]

Essencialmente, estão disponíveis dois tipos de tags activas – *transponders* e *beacons*.

- ***Transponders***: num sistema que utiliza etiquetas *transponder* activas, o leitor - tal como acontece nos sistemas passivos - enviará primeiro um sinal, e de seguida o *transponder* activo irá enviar um sinal de volta com a informação relevante. As *tags transponder* são muito eficientes, uma vez que conservam a vida útil da bateria quando a *tag* se encontra fora do alcance do leitor. São usualmente utilizados no controlo de acesso seguro e em sistemas de pagamento de portagens.

- ***Beacons***: num sistema que utiliza uma *tag* activa do tipo *beacon*, a *tag* não aguardará para ouvir o sinal do leitor. Em vez disso, tal como o seu nome indica, a *tag* irá sinalizar (i.e. enviar um beacon), ou então irá enviar a sua informação específica a cada 3 - 5 segundos. As *tags beacon* são bastante comuns na indústria do petróleo e gás, assim como aplicações de *mining* e rastreamento de mercadorias. As *tags* activas do tipo *beacon* podem ser lidas a centenas de metros de distância mas, de forma a conservar a vida útil da bateria, têm que ser configuradas para uma potência de transmissão inferior, que ronda o alcance de leitura de cerca de 100 metros.

Uma vez que a este tipo de *tags* são atribuídas tarefas como resistir a condições atmosféricas adversas, tais como temperaturas extremas e humidade, a maioria das *tags* RFID activas são envoltas numa casca robusta. Devido ao tamanho da bateria e dos circuitos, as *tags* RFID activas são geralmente muito maiores do que as passivas.

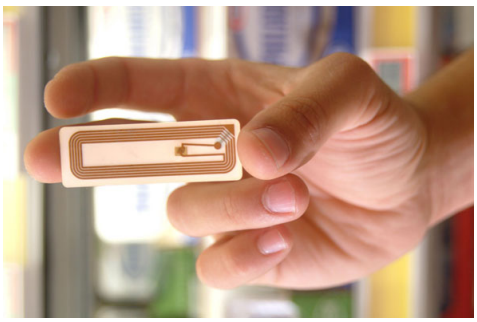
Comparação uma Tag Activa e uma Tag Passiva. [25]

Além disso, algumas *tags* activas podem ter sensores de bordo que rastreiam os parâmetros ambientais. Estes sensores podem rastrear os níveis de humidade, temperatura, e outros identificadores-chave que uma empresa pode utilizar para as suas aplicações.

Vantagens das *tags* RFID activas:

- Intervalos de leitura extremamente longos (em distância).
- Aumento das habilidades da *tag* com tecnologias associadas (e.g. GPS, sensores, etc).
- Opções de *tags* extremamente resistentes.

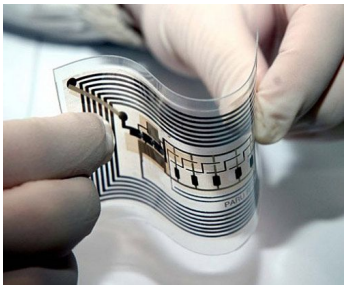
Passivo: neste modo não há bateria e a corrente é fornecida pelo leitor. Uma *tag* RFID passiva possui um alcance de leitura com uma distância e um tamanho menor, quando comparada com uma *tag* activa. Ao contrário das *tags* activas, as passivas apenas possuem duas componentes principais: a antena e o microchip ou circuito integrado (CI).



Exemplo de uma Tag Passiva. [26]

Tal como o nome implica, as *tags* passivas esperam por um sinal de um leitor RFID. O leitor envia energia para uma antena, que converte essa energia numa onda RF que é enviada para a zona de leitura. Uma vez que a *tag* é lida dentro da zona de leitura, a antena interna da tag RFID extrai energia

das ondas RF. A energia move-se da etiqueta para o CI e alimenta o chip, o que gera um sinal de volta para o sistema RF. Este processo denomina-se de retrodifusão, *"backscatter"*. A retrodifusão, ou mudança na onda electromagnética é detectada pelo leitor através da antena, que por sua vez interpreta a informação.

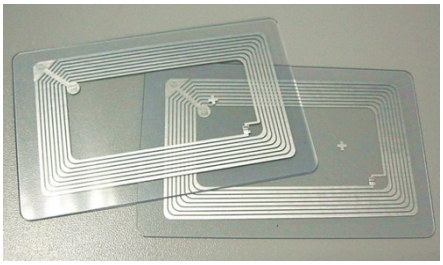


Outro Exemplo de uma Tag Passiva [27]

Como foi mencionado acima, as etiquetas RFID passivas não têm fonte de alimentação interna, e consiste apenas num CI e numa antena. Esta estrutura básica é chamada de revestimento *("inlay")* RFID. Existem inúmeros outros tipos de *tags* passivas no mercado, mas todas geralmente se dividem em duas categorias: *inlays* (revestidas) ou *hard tags* (tags fortes). As hard tags são resistentes e feitas de plástico, metal, cerâmica ou até mesmo de borracha. Elas vêm em todas as formas e tamanhos, normalmente concebidas para uma função, material ou aplicação únicas.

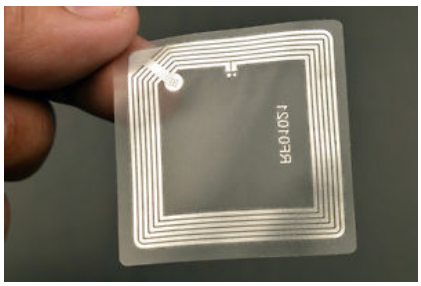
As *inlays* são geralmente as *tags* mais baratas, chegando a custar 0,12€ por *tag* em grandes quantidades, mas o preço não afecta a sua performance. As *tags inlay* estão agrupadas em três tipos principais:

- ***Dry Inlays***: um microchip RFID e uma antena conectada a um material ou substrato a que se dá o nome de teia. Estes *inlays* parecem ter sido laminados e vêm geralmente sem adesivo.



Exemplo de Tag Passiva do tipo Dry Inlay [28]

- ***Wet Inlays***: um microchip RFID e uma antena anexados a um material, geralmente PET ou PVT, com um adesivo de apoio. Na maioria das vezes estas *tags* são transparentes, podendo ser retiradas do rolo e imediatamente coladas num objeto.



Exemplo de Tag Passiva do tipo Wet Inlay [29]

- ***Paper Face Tags***: estas são essencialmente *wet inlays* com um papel branco ou face polida. São ideais para aplicações que precisam de números impressos ou logótipos para identificação.

Exemplo de *Tag* Passiva do tipo *Paper Face Tag* [30]

Nem todas as *tags* passivas operam na mesma frequência. Existem três frequências principais dentro das quais as etiquetas RFID passivas operam. O alcance da frequência, juntamente com outros factores, determina fortemente o alcance de leitura, os materiais anexos e as opções de aplicação:

- **125 a 134 KHz** - Baixa Frequência (LF ou *Low Frequency*): comprimento de onda extremamente longo, geralmente com um curto alcance de leitura que ronda 1 a 10 centímetros. Esta frequência é tipicamente utilizada no rastreio de animais, uma vez que não é muito afectada pela água ou metal.

- **13.56 MHz** – Alta Frequência (HF ou *High Frequecy*) e *Near Field Communication* (NFC): um comprimento de onda médio com um alcance de leitura típico de cerca de 1 centímetro até 1 metro. Esta frequência é utilizada em transmissões de dados, aplicações de controlo de acesso e segurança de passaportes – aplicações que não requerem um alcance de leitura longo.

- **865 a 960 MHz** – *Ultra High Frequency* (UHF): um comprimento de onda curto e de alta energia com cerca de 1 metro, que se traduz num alcance de leitura longo. As *tags* UHF

passivas podem ser lidas a partir de uma distância média de 5 – 6 metros, mas *tags* UHF maiores podem atingir até mais do que 30 metros de alcance de leitura, nas condições ideias. Esta frequência é normalmente utilizada com tempos de corrida, rastreamento de arquivos, gestão de lixo, entre outras, visto que todas essas aplicações normalmente necessitam de mais do que 1 metro de alcance de leitura.

Vantagens das *tags* RFID passivas:

- São mais pequenas.
- São muito mais baratas.
- São mais flexíveis e finas.
- Maior variedade de opções de *tags*.
- Podem durar uma vida inteira sem uma bateria (dependendo do uso e do desgaste).

Utilização da Tecnologia RFID

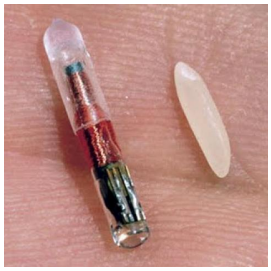
A tecnologia RFID tem vários usos, no entanto apenas iremos discutir sobre a sua aplicação nos transportes públicos. Citam-se alguns dos exemplos de uso:

- Comércio.
  - Controlo de Acesso.
  - Publicidade.
  - Rastreio de Promoções.
- Transportes e Logística.
  - Sistemas de transporte inteligentes .
  - Maior variedade de opções de *tags*.
- Transportes Públicos.
- Gestão e Proteção de Infraestruturas.
- Passaportes.
- Pagamentos de Transportes.
- Identificação de Animais.
- Identificação de Humanos.
- Instituições.
  - Hospitais e Centros de Saúde.
  - Livrarias.
  - Museus.

Fechadura Electrónica com Sistema de Identificação RFID [24]

Antena instalada num Semáforo para monitorização de transportes públicos [31]

- Escolas e Universidades.
- Desportos.
- Complemento para o Código de Barras.
- Telemetria.



Comparação de um grão de arroz com um chip RFID implantado nos animais para propósitos de identificação [32]

Vantagens e Desvantagens

Apesar do conceito simplista deste método de localização de veículos e de ser possível localizar os veículos onde não é possível receber o sinal GPS (e.g. túneis ou locais com fraca rede), as suas principais desvantagens são os custos associados à instalação de leitores RFID em todos os veículos ou pontos fixos de todos os percursos. Assim como alterar a localização destes leitores a cada vez que se verificarem alterações nos percursos ou quando são criados novos percursos, bem como a instalação e manutenção do serviço de comunicação.

Outra desvantagem identificada é o facto de que, se existir uma alteração forçada na rota de um veículo (e.g. obras temporárias em determinado segmento de uma rota de autocarros) e se a estrutura não for adaptada, deixa de ser possível fazer a localização do veículo nesses locais. Também em casos de atrasos, avarias ou assaltos não é possível saber a localização exacta do veículo, se este não passar pelos pontos onde os dois equipamentos necessários comunicam de forma a obter informação da localização.

Apesar das suas desvantagens, a solução RFID é talvez a solução que melhor se ajusta a um sistema de monitorização de transportes em tempo real num contexto urbano, pelo menos no que diz respeito à eficiência da localização dos transportes. A identificação de cada ponto é unívoca e não está sujeita ao erro, ou seja não existe a possibilidade de o sistema erradamente identificar um ponto como sendo outro, localizado próximo do anterior.

Caso exista uma infraestrutura de comunicações fixa, este método permite que não existam gastos em sistemas de comunicação móvel para as entidades fornecedoras do serviço. No entanto, ainda assim é necessário que todos os veículos estejam equipados com o dispositivo identificador e os pontos fixos estejam equipados com leitores RFID (ou vice-versa).

B. Processamento de Imagem

A análise de imagem tem sido nos últimos anos alvo de bastante investigação, sendo hoje possível processar imagens e identificar alguns objetos ou gráficos específicos. Fazendo uso destas investigações, existem várias abordagens de localização de viaturas através do processamento de imagens.

Através de uma câmara instalada no veículo o sistema consegue, ao analisar o ambiente em redor do veículo, tirar parâmetros que, cruzando os dados com informações anteriormente obtidas, lhe permite identificar a localização atual do veículo. Foram ainda identificados sistemas que recorriam a sistemas de vigilância das cidades para analisar o número do veículo inscrito no tecto do mesmo, sistemas de identificação da chapa da matrícula, ou ainda sistemas onde existe um dispositivo LED nas paragens de autocarro com uma determinada sequência de piscar, piscar esse que uma câmara a bordo do veículo descodifica num código e reconhece assim a paragem.

No geral, estes sistemas estão ainda numa fase de estado da arte, uma vez que representam custos elevados, quer de desenvolvimento quer da infraestrutura necessária.

C. Dead Reckoning

O sistema de localização *Dead Reckoning* [33] consiste no pré-conhecimento do percurso a efectuar, que através do odómetro do veículo identifica a distância percorrida desde o início da viagem até ao ponto atual. Este sistema está sujeito a erros acumulativos e não é muito preciso uma vez que, se por qualquer motivo o veículo tenha que se desviar da rota pré-definida, os dados ficam incorrectos. Os avanços nos sistemas de navegação que fornecem informações precisas sobre a posição, em particular os sistemas de navegação que utilizam GPS, fizeram com que o *Dead Reckoning* simples se tornasse obsoleto para a maioria dos propósitos.

Existem, no entanto, AVL's baseados em *dead reckoning* e no sistema de identificação RFID que conseguem garantir alguma fiabilidade. É uma solução que pode ser adaptada em pequenos percursos e onde a distância entre paragens não é muito elevada. Esta solução também é viável se for integrada com um giroscópio e/ou acelerómetro. Com isto, para além de ser possível medir a distância percorrida, é também possível identificar para onde é que o veículo se está a dirigir. Esta informação em tempo real, em conjunto com um mapa, permite saber a localização do veículo, com alguma margem de erro.

Os sistemas de navegação inercial, que fornecem informações bastante precisas acerca da direção, utilizam o *Dead Reckoning* e são amplamente aplicados.

D. Triangulação de Frequências Rádio

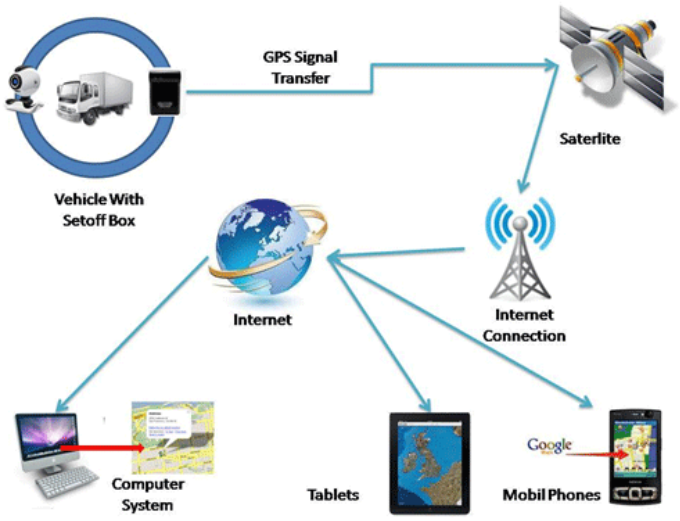
O sistema de localização através da triangulação de sinais de rádio é baseado nas redes GSM [34] existentes. Este sistema consiste na triangulação de sinais emitidos por estruturas construídas para o efeito, ou a partir de antenas das células da rede GSM, e a partir da medição do sinal é possível localizar o veículo.

Este sistema requer material extremamente sensível e dispendioso para que consiga medir os sinais com alguma precisão. Os resultados obtidos não são muito satisfatórios, uma vez que a margem de erro acaba por ser bastante maior, quando comparada com a maioria das alternativas que se apresentam.

E. GPS

O GPS - tecnologia que será utilizada no desenvolvimento deste projecto - é um sistema de localização por satélite que fornece a um dispositivo a sua localização. Este sistema é composto por uma "*constelação*" de 24 satélites colocados estrategicamente em órbita de modo a que, a partir de qualquer ponto do planeta, estejam sempre pelo menos 4 satélites em "*linha de vista*". Os satélites enviam constantemente sinais para a Terra com a sua identificação e a hora existente nos seus relógios atómicos, permitindo ao receptor - através do cálculo da distância deste aos satélites - calcular as coordenadas do local onde se encontra.

O sistema GPS foi desenvolvido e pertence ao Departamento de Defesa dos Estados Unidos da América, e desde meados de 2000 o serviço está disponível ao público de todo o mundo. No entanto, este serviço apenas está disponível livremente com uma margem de erro de erro à volta dos 10 metros, estando os recursos mais precisos para utilização exclusiva das Forças Armadas dos EUA. Existe também já em funcionamento o sistema Russo GLONASS, que disponibiliza o serviço de forma gratuita, com uma precisão máxima de decímetros. Apesar da sua precisão e dos EUA desactivarem o GPS para uso civil quando necessitam dos recursos para operações militares, o GLONASS ainda não é muito comum, devido a só estar ainda disponível desde o final do ano de 2011. Como tal, ainda não existem muitos equipamentos associados a esta tecnologia. Existem ainda alguns sistemas em desenvolvimento, como o GALILEO da União Europeia, que pretende aperfeiçoar a localização do GPS, reduzindo o erro de 10 metros para um intervalo de 1 a 3 metros, e o Compass da China.



Exemplo de Funcionamento do Sistema GPS [12]

Atualmente, para resolver a questão da precisão do GPS, utilizam-se abordagens que permitem diminuir a margem de erro de 10 para menos de 2 metros, dependendo das implementações e dos equipamentos. As duas abordagens mais comuns são a implementação de correções baseadas em *Differential* GPS [35] e a implementação de correções baseadas no sistema EGNOS [36].

Pelo excelente desempenho do GPS e pelo baixo custo de implementação, desenvolvimento e manutenção do serviço, este mostra-se como a principal abordagem a tomar nos AVL's existentes atualmente. O preço baixo e a omnipresença do GPS fazem aumentar a confiança neste sistema de localização. Os sinais de GPS são impermeáveis à maioria dos sinais eléctricos e não requerem a instalação de um sistema inteiro, necessitando apenas do receptor para recolher os sinais provenientes do segmento dos satélites.

Num contexto urbano, em que os diferentes pontos de paragem estão em muitos casos geograficamente muito próximos na rede de transportes, uma imprecisão na localização, dentro da gama de erros que o GPS considera, pode provocar um erro na identificação de uma posição por parte do sistema. Além disso, quando não existe linha de vista com os satélites (e.g. túneis, cidades com grandes edifícios ou árvores), o sistema pode perder a cobertura ou apanhar sinais reflectidos e apresentar dados errados.

No entanto, apesar das suas limitações, por ser uma tecnologia muito reconhecida, utilizada, em constante aperfeiçoamento e economicamente acessível, o GPS revela-se uma componente essencial

em muitos sistemas de monitorização de transportes urbanos em tempo real, mas quase sempre em combinação com outra tecnologia ou mecanismo que permita superar ou sustentar de alguma forma as suas limitações.

F. Distância + Abertura de Portas

Este método de localização [11] consiste em cada veículo informar automaticamente um sistema central acerca dos quilómetros percorridos até ao instante, tal como o momento de chegada a um ponto da rede, associada à abertura de portas do veículo, neste caso um veículo de transporte público. A informação do número de quilómetros pode ser recolhida a partir da leitura do odómetro, equipamento destinado a medir a distância percorrida, ou do GPS dos veículos, sendo que a primeira opção é a mais utilizada e fiável. No entanto, em alguns casos, por exemplo com o desgaste dos pneus do veículo, podem observar-se algumas falhas consideráveis.

Como limitações, este método apresenta essencialmente o facto de estar dependente da abertura de portas para a notificação do sistema central da chegada a um determinado nó, o que obriga a que o tripulante pare em todas as estações da rede, mesmo que nela não aguardem passageiros. Por outro lado, limita a abertura de portas à chegada aos nós, o que em algumas situações pode levar à incompreensão e insatisfação por parte dos passageiros.

G. Signpost Transmitters

O método de localização *Signpost Transmitters* [11] é utilizado para localizar veículos ao longo de rotas inalteráveis, a partir de pontos fixos. Este método é utilizado em vias de trânsito e em linhas férreas onde os veículos estão a operar continuamente na mesma rota. Um receptor – transmissor ou chip com tecnologia RFID - é colocado ao longo das rotas dos veículos, e cada unidade de transporte que passa por um destes postos fixos envia e recebe um sinal para o transmissor do sistema. O transmissor móvel reporta a passagem pelo poste X do controlador do sistema. Esta supervisão permite, por exemplo, a um centro de controlo monitorizar o progresso dos veículos e vislumbrar se está a cumprir os horários definidos.

A maioria dos AVLs estudados apresentam o GPS como a principal fonte de dados, mas há também a abordagem de integrar vários sistemas num só. Por exemplo, conjugar o GPS com sistemas *dead-reckoning*; ou GPS com o sistema de identificação de postos fixos (RFID) num determinado percurso; ou o GPS com um detector de abertura de portas de veículos; ou ainda a conjugação destes todos.

H. Tecnologias de Comunicação Sem Fios

Um SAEIP (Sistema de Apoio à Exploração e Informação ao Passageiro) baseia-se na possibilidade de adquirir dados e de os transmitir em tempo real para um sistema com poder computacional, onde serão processados para originar informação útil a vários utilizadores com diferentes necessidades. Para que esta transmissão seja possível, é necessário recorrer a tecnologias de comunicação que garantam disponibilidade, largura de banda e fiabilidade. Atualmente, as redes sem fios mais adequadas ao cenário dos transportes públicos são apresentadas nesta secção.

Wi-Fi

Designa-se como "tradicional" o sistema utilizado nas redes locais sem fios domésticas e de empresas. *Wi-Fi* [37] é uma tecnologia utilizada por produtos certificados que pertencem à classe de dispositivos de rede local sem fios (WLAN) baseados no padrão IEEE 802.11 [38]. Dispositivos que sejam compatíveis com *Wi-Fi* podem conectar-se à internet através de uma rede WLAN ou um *hotspot* de acesso *wireless*, sendo que o padrão *Wi-Fi* opera em faixas de frequência que não necessitam de licença para instalação e/ou operação, o que torna estas redes atractivas.

Nos veículos de transportes públicos que sejam compatíveis com *Wi-Fi*, esta rede é utilizada em algumas situações. Por exemplo, quando os veículos estão no parque durante um espaço de tempo para fazer o *download/upload* de maiores volumes de informação para os equipamentos de processamento e armazenamento de informação, que não são necessariamente importantes em tempo real (e.g. vídeos de videovigilância, informação relativa aos sistemas de bilhética, atualização de conteúdos publicitários e informativos, etc.). Esta tecnologia tem a vantagem de poder obter uma maior largura de banda e garantia de qualidade de serviço. No entanto, é restringida à área abrangida pelo sinal de rede.

TETRA

A tecnologia *Terrestrial Turked Radio* (TETRA) [39] é uma tecnologia de rádio móvel, privada e digital. As características principais desta rede são a segurança, disponibilidade, a possibilidade de implementar QoS (*Quality of Service*) e o facto de ser *standard* para as comunicações móveis profissionais.

As principais diferenças entre TETRA e GSM são o facto de que a TETRA foi especialmente desenvolvida para entidades com tarefas críticas como autoridades, serviços de emergência,

transportes públicos e forças armadas, implementando sistemas especiais de redundância e nível de prioridade nas comunicações.

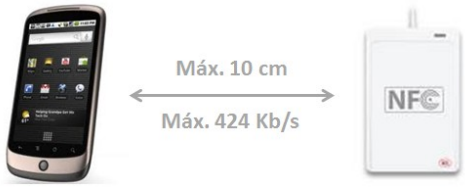
A principal desvantagem deste serviço é o facto de ser um serviço privado, e como tal tem custos elevados inerentes à sua utilização. Contudo, nos SAEIP instalados nos grandes centros urbanos, devido à necessidade de garantir a disponibilidade do serviço, a rede instalada é a TETRA. Verifica-se também a utilização de sistemas híbridos, onde esta tecnologia é utilizada somente na transmissão de vídeo e conferência por voz.

GSM (GPRS/UMTS)

Global System for Mobile Communications (GSM) [34] é o padrão em que a rede de telemóveis pública se baseia para implementar a comunicação entre dispositivos móveis. Esta tecnologia disponibiliza uma largura de banda limitada, mas suficiente para a transmissão de dados necessários às funcionalidades principais do sistema (e.g. coordenadas, localização, dados do veículo do *Engine Control Unit*, i.e. principalmente os dados em formato de texto). No entanto, a transmissão de dados em tempo real, como o vídeo dos sistemas de videovigilância ou a implementação de um sistema de voz (apenas UMTS, GPRS não disponibiliza largura de banda para estes serviços), não será aconselhável uma vez que pode comprometer a transmissão dos restantes dados.

NFC

O *Near Field Communication* (NFC) [40] é uma tecnologia que permite a transferência de dados numa comunicação sem fios de curta distância. O NFC surgiu a partir do RFID, tendo sido desenvolvido pela Sony e Philips em 2002 e impulsionado pelo "*NFC Forum*", que é conduzido por empresas como Samsung, Microsoft, Nokia, Google, Intel e Visa.



Funcionamento do NFC [41]

Assim como o RFID, a comunicação acontece de maneira simples e intuitiva, apenas aproximando os dispositivos NFC, isto é, dispositivos que possuam um chip NFC. Desta forma, os utilizadores podem realizar transações financeiras, aceder a conteúdos digitais ou contectar diferentes aparelhos. Os chips

podem vir em *smartphones* e outros dispositivos electrónicos, assim como em tags NFC/RFID. Com isto, é possível estabelecer a comunicação entre diferentes objetos físicos, como cartões, máquinas, entre outros.

O NFC possui dois modos de funcionamento:

Activo: neste modo, os dois dispositivos envolvidos na comunicação podem realizar transferências de dados, enviando e recebendo. O modo activo ocorre, por exemplo, no emparelhamento de *smartphones* para troca de arquivos de multimédia, como fotos e vídeos.

Exemplo de NFC Activo [42]

Passivo: neste modo, um dos dispositivos emite um sinal e o outro apenas recebe a informação, estabelecendo uma comunicação de uma via. É o caso de um *smartphone* que recebe os dados de uma *tag* NFC.

Exemplo de NFC Passivo em Pagamentos com cartões ou smartphones [43] [44]

As aplicações com NFC também podem ser categorizadas em três tipos de operação:

- ***Reader/Writer***, um dispositivo com NFC identifica dados numa *tag* NFC.
- ***Card Emulation***, um dispositivo com NFC simula o comportamento de um cartão, para a realização de transações financeiras ou acesso a portas.
- ***Peer-to-Peer***, dois dispositivos são emparelhados, estabelecendo uma comunicação entre ambos.

A utilização de NFC e RFID nos transportes urbanos

Os ITS [45] (Sistemas de Transporte Inteligente – *Intelligent Transportation Systems*) são sistemas que prevêm a utilização de tecnologias avançadas aplicadas a veículos e à infra-estrutura e que têm como objetivo a melhoria funcional e eficiência energética. Associado a isso, as tecnologias de comunicação assinalam-se cada vez mais como partes integrantes do processo de soluções para o cenário do transporte urbano. Procuram-se alternativas visando a flexibilidade e praticabilidade em cidades que enfrentam grandes problemas de trânsito.

Com estas tecnologias é possível passar informações acerca do transporte da cidade, como mapas e pontos de localização, horários de autocarros, comboios, metros, trânsito em tempo real, serviços de táxis, entre outros. Além disso, existem também as funcionalidades dos acessos aos torniquetes e a compra de bilhetes. Já há alguns anos, algumas cidades no mundo encontraram soluções em aplicações com NFC e RFID para auxiliarem os seus sistemas de transporte:

- **Programa Roda SP (São Paulo - Brasil)**

  O programa Roda SP[46] é realizado durante o verão em São Paulo e dá direito a circular durante 24h pelos municípios da cidade, apenas com um único bilhete de autocarro. Desta forma, o passageiro pode circular sobre todos os pontos participantes da cidade as vezes que quiser. Até 2012, o controlo era realizado através de uma pulseira simples no braço, o que dificultava muito a tarefa.

  Em 2013, o programa testou o Cartão de Acesso NF, desenvolvido pelo CPDT (Centro de Pesquisa e Desenvolvimento em Telecomunicações). Ao comprar o bilhete, o utilizador recebe um cartão NFC, que é utilizado em cada embarque e desembarque do autocarro. Ao passar o cartão pela primeira vez é iniciada a contagem das 24h de uso. Desta forma, o sistema com NFC contribuiu para um controlo exacto do número de passageiros que estiveram no autocarro, assim como os pontos com maior afluência e os períodos mais intensos de utilização.

- **RFID em Acessos a Portagens**

  Utilizado em Portugal desde 1991, este sistema foi desenvolvido pela Universidade de Aveiro, que é mais conhecido por "*Via Verde*. Este possibilita o acesso inteligente de veículos nas portagens das autoestradas. O sistema funciona a partir de uma *tag* RFID, mais especificamente um dispositivo DSRC, que é colocada no automóvel. Ao aproximar-se da

portagem, a *tag* presente no veículo é identificada e não é necessário parar o carro. No final do mês o utilizador recebe a factura com o valor de todas as portagens por onde circulou.

No Brasil está ainda em fase de testes o sistema *Ponto a Ponto*, que funciona de maneira similar à *Via Verde*, mas com um modelo de pré-pagamento. Desta forma, o utilizador compra créditos que lhe são descontados conforme a utilização das portagens. Além disso, o programa também instalou leitores RFID pelas estradas, que servem para identificar as *tags* dos carros, para que a cobrança seja feita de acordo com a distância percorrida, e sem a necessidade de portagens. Esta última alteração assemelha-se ao sistema de Portagens SCUT presente em Portugal.

• Pagamentos com NFC em Transportes Urbanos

Desde 2007, são realizados testes para a realização de pagamentos com NFC, substituindo os tradicionais bilhetes de passagem utilizados em transporte público. Em cidades da Alemanha, o projecto *Touch & Travel* testou com aproximadamente 3 mil participantes a aceitação do NFC em smartphones como meio de acesso ao check-in e check-out em comboios. Os benefícios reportados foram a flexibilidade na emissão de bilhetes, com baixo custo e infraestrutura.

Nos EUA, uma parceria entre o sistema de trânsito de São Francisco e empresas de *fast-food* resultou num projecto que possibilitava o pagamento móvel com NFC nos transportes públicos, e ainda a compra de refeições nos estabelecimentos parceiros, através de *smart posters* presentes nas estações de embarque.

Em Londres, foi também testado um projecto que utilizava *smartphones* com NFC para o acesso e compra de bilhetes, em substituição dos cartões de transporte tradicionais. Como resultado, os participantes demonstraram alto nível de interesse e satisfação no projecto, pela sua conveniência, facilidade de uso e *status* que sentiram com o uso da aplicação.

LTE

A recente entrada da tecnologia de comunicação ***Long Term Evolution*** (LTE) [47] no mercado veio abrir portas à implementação de novos serviços em tempo real em ambientes móveis, disponibilizando nesta primeira versão uma largura de banda máxima teórica de *170 Mbps*, muito superior às anteriores tecnologias.

No entanto, atualmente a implementação desta tecnologia pelos operadores de comunicações encontra-se ainda numa fase inicial, onde a cobertura disponível se resume apenas a algumas cidades e zonas em específico. Num futuro próximo, será esta a tecnologia que irá substituir o atual GPRS/UMTS.

I. Outras Tecnologias

São ainda utilizadas, dentro do contexto de transportes públicos em ambiente urbano, outros tipos de tecnologias, como UHF [48] (*Ultra High Frequency*) e/ou VHF [49] (*Very High Frequency*), DSRC [50] (*Dedicated Short Range Communications*), *Wimax* [51] ou ainda as *Mesh Networks* [52].

ANEXO II – Especificação Funcional

Neste anexo pode ser encontrada a especificação funcional do modelo propostos, tendo em conta as limitações impostas ao projeto. Todos os requisitos são importantes, mas devem ser priorizados de forma a entregar com antecedência os maiores e mais imediatos benefícios de um sistema. Portanto, após o levantamento dos requisitos utilizou-se para a atribuição da sua prioridade a técnica de ***MoSCoW*** [53]**,** segundo o critério de importância. Neste caso apenas se utilizaram os critérios "*Essencial*" (*Must*) e "*Não-Essencial*" (*Should* ou *Won't*). Os requisitos que se enquadram nos não-essenciais são requisitos de segundo plano ou para trabalho futuro, isto é, não são essenciais ao funcionamento do sistema.

A primeira fase da especificação do sistema passou então por identificar os requisitos do sistema a desenvolver. Apesar do modelo ser constituído por três subsistemas distintos e que foram desenvolvidos separadamente, todos juntos trabalham atingem um objetivo final único, pelo que, inicialmente, os requisitos de todos os subsistemas poderão ser descritos como requisitos do sistema. Posteriormente os requisitos serão associados ao subsistema a que dizem respeito.

Existem quatro intervenientes: as pessoas que utilizam os transportes públicos no seu dia-a-dia e que são os principais utilizadores do sistema (a quem esta se destina), a que chamamos simplesmente de **passageiro**; os **motoristas** dos autocarros que atualizam os dados do autocarro em tempo real; os **administradores** que tratam da gestão dos dados da aplicação; e por fim, os **programadores,** que são os desenvolvedores do produto, embora, para estes últimos, não existam nenhum subsistema desenvolvido.

No entanto, apesar do vasto leque de utilizadores, a aplicação tem como verdadeiro objetivo o auxílio do utilizadores de transportes públicos. Pretende-se, não que o seu tempo de espera pelos autocarros seja reduzido, mas sim que esse tempo de espera lhe seja transmitido, de forma a que o utilizador possa planear eficientemente as suas viagens, tendo em conta o tempo que o autocarro demora a chegar ao ponto de paragem mais próximo do utilizador.

Nas próximas secções serão abordados os requisitos do sistema, agrupados tendo em conta o tipo de utilizador, em alguns casos a respectiva importância ou prioridade, e a possível ocorrência de algumas

situações durante o uso por parte dos seus utilizadores. No caso dos requisitos não-funcionais, estes encontram-se agrupados por categoria.

A. Requisitos do Passageiro

Caso o utilizador tenha o GPS ligado e a internet activos, seja a partir de uma ligação *Wi-Fi* ou GSM, este deve ser capaz de visualizar a sua posição e as paragens de autocarro mais próximas num mapa, assim como a posição em tempo real dos autocarros. Além disso, o utilizador deverá também ser capaz de visualizar todos os horários fixos da rede de transportes da cidade onde se encontra. Poderá optar por ver o percurso de um itinerário no mapa ou em lista, com todas as paragens e pontos de interesse por onde esse percurso passa.

Caso o utilizador esteja *offline*, isto é, não tenha a internet ligada ou haja alguma falha da ligação mas no entanto tem o GPS activo, este pode fazer tudo o que foi dito anteriormente, à excepção de algumas funcionalidades. Sem internet o utilizador não será capaz de visualizar o tempo estimado de chegada do autocarro nem visualizar a posição do autocarro em tempo real no mapa, uma vez que a aplicação sem internet não consegue estimar o tempo que o autocarro demorará a chegar à paragem onde o utilizador se encontra. O utilizador também não é capaz de ver no mapa a posição em tempo real do autocarro, uma vez que sem internet o sistema não consegue transmitir essa informação ao utilizador.

No caso de falha temporária da ligação à internet, mas com o GPS activo, o sistema guarda os dados da posição do utilizador localmente e tenta enviá-los assim que a ligação é novamente estabelecida.

Se o utilizador não tiver internet nem o GPS activo não é possível obter as coordenadas da sua posição atual, logo o sistema não consegue fazer praticamente nada do que foi previamente dito. Pode ver no mapa os itinerários e horários existentes no sistema, no entanto sem sinal GPS não é possível obter o caminho para a paragem mais próxima dele.

Sendo este o utilizador mais importante do sistema, os requisitos do passageiro encontram-se de seguida organizados segundo a ordem de importância. Segundo a técnica de ***MoSCoW***, os requisitos essenciais são aqueles que devem ser o objetivo principal e devem ser implementados em primeiro lugar. Os não-essenciais, ou menos importantes, são aqueles que devem ser deixados para segundo plano ou para um trabalho futuro.

Essenciais:

- O utilizador deve ser capaz de visualizar a sua posição atual no mapa.
- O utilizador deve ser capaz de visualizar as paragens mais próximas da sua localização atual.
- O utilizador deve ser capaz de visualizar o caminho para a paragem mais próxima da sua localização atual.
- O utilizador deve ser capaz de visualizar num mapa a posição do autocarro em tempo real.
- O utilizador deve ser capaz de visualizar os horários fixos de todos os autocarros.
- O utilizador deve ser capaz de visualizar o percurso de um itinerário no mapa.
- O utilizador deve ser capaz de visualizar todos os itinerários possíveis.
- O utilizador deve ser capaz de visualizar todas as paragens e pontos de interesse de um dado percurso.
- O utilizador deve ser capaz de visualizar os detalhes de uma paragem.
- O utilizador deve ser capaz de visualizar os detalhes de um ponto de interesse.

Não - Essenciais:

- O utilizador deve ser capaz de visualizar num mapa a área de cobertura da rede de transportes.
- O utilizador deve ser capaz de visualizar o tempo estimada de chegada de um autocarro.
- O utilizador deve ser capaz introduzir um destino.
- O utilizador deve ser capaz de ver o tempo e custo de viagem para o destino que introduziu.

Vejamos agora como se comportam os requisitos, em função de certas situações que possam ocorrer durante a utilização da aplicação.

Se o utilizador tiver uma ligação internet e o GPS activos:

- O utilizador deve ser capaz de visualizar a sua posição atual no mapa.
- O utilizador deve ser capaz de visualizar as paragens mais próximas da sua localização atual.
- O utilizador deve ser capaz de visualizar o caminho para a paragem mais próxima da sua localização atual.
- O utilizador deve ser capaz de visualizar num mapa a posição do autocarro em tempo real.
- O utilizador deve ser capaz de visualizar os horários fixos de todos os autocarros.
- O utilizador deve ser capaz de visualizar o percurso de um itinerário no mapa.

- O utilizador deve ser capaz de visualizar todos os itinerários possíveis.
- O utilizador deve ser capaz de visualizar o tempo estimada de chegada de um autocarro.

Se o utilizador NÃO tiver uma ligação internet mas TIVER o GPS activo:

- Funcionalidades que **NÃO** são possíveis:

- O utilizador não é capaz de visualizar o tempo previsto de chegada do autocarro à paragem mais próxima.
- O utilizador não é capaz de visualizar num mapa a posição de um dado autocarro em tempo real.

- Funcionalidades que **SÃO** possíveis:

- O utilizador deve ser capaz de visualizar a sua posição atual no mapa.
- O utilizador deve ser capaz de visualizar as paragens mais próximas da sua localização atual.
- O utilizador deve ser capaz de visualizar o preço total da viagem desde o ponto onde se encontra até ao seu destino.
- O utilizador deve ser capaz de visualizar os horários fixos de todos os autocarros.
- O utilizador deve ser capaz de visualizar todos os itinerários possíveis.

No caso de falha temporária da ligação à internet, mas com o GPS activo:

- O sistema tenta estabelecer a ligação assim que possível para recolher informações em tempo real.

Se NÃO existir ligação à internet e o GPS estiver INACTIVO:

- Não é possível obter as coordenadas da sua posição do utilizador, logo o sistema não consegue fazer nada do que foi previamente dito.

B. Requisitos do Administrador

Os utilizadores *admins* - ou simplesmente administradores - devem ser capazes de visualizar, remover, adicionar ou alterar os dados necessários ao funcionamento do sistema, isto é, os dados que estão presentes nos veículos e na central. Exemplo desses dados são os itinerários; a localização das paragens de autocarro e respectivos horários dessas paragem tendo em conta o itinerário; os dados dos autocarros; os dados dos motoristas; as coroas das zonas da cidade, entre outros. Esta é a informação que se encontra na Central-MIB presente na central, que contém a informação estática e

dinâmica de todo o sistema. Para levar a cabo este tipo de acções os administradores necessitam obrigatoriamente de uma ligação à internet.

Geralmente os administradores fazem a sua gestão a partir de um computador, mas por vezes podem ocorrer falhas de rede.

Se o administrador tiver uma ligação internet:

- O administrador deve ser capaz de visualizar, adicionar, remover ou alterar os dados presentes na central. Esses dados podem ser itinerários, coroas, localizações de paragens de autocarro e respectivos horários dessas paragem, dados de autocarros, motoristas, horários, etc.

Se o administrador NÃO tiver uma ligação internet:

- O administrador não consegue fazer nada no sistema de gestão, uma vez que necessita obrigatoriamente de uma ligação à internet para levar a cabo as suas funções.

C. Requisitos dos Programadores da Aplicação

Tal como os administradores de um nível mais baixo, assume-se que os administradores programadores geralmente executam as suas funções num local com rede fixa com baixa probabilidade de falhas de ligação.

Se o administrador programador tiver uma ligação internet:

- Os administradores podem fazer a configuração do sistema , isto é, todos os ficheiros de configuração que permitam a comunicação entre os vários subsistemas , como os utilizadores, central e veículos. No fundo, tratam da parte SNMP da aplicação.

Se o administrador programador NÃO tiver uma ligação internet:

- Sem uma ligação internet, o administrador programador não consegue fazer nada no sistema, uma vez que necessita obrigatoriamente de uma ligação à internet para levar a cabo as suas funções.

D. Requisitos do Motorista do Autocarro

O motorista pode indicar à aplicação quando e qual o itinerário que vão iniciar com o autocarro que está a conduzir e notificar a aplicação quando o autocarro chega ao fim desse itinerário.

Quando é iniciado um novo itinerário:

- O motorista deve ser capaz de indicar à aplicação quando e qual o itinerário que vai iniciar com o autocarro que está a conduzir.

Quando um itinerário é finalizado:

- O motorista deve ser capaz de indicar ao sistema quando o autocarro chega ao fim desse itinerário e qual o próximo itinerário que vai ser iniciado com esse autocarro.

E. Requisitos Funcionais de Sistema

Essenciais:

- O sistema deve monitorizar a posição dos autocarros em tempo real.
- O sistema deve indicar ao utilizador as paragens mais próximas que o levam ao seu destino.
- O sistema deve monitorizar as coordenadas latitude da posição atual do utilizador.
- O sistema deve monitorizar as coordenadas longitude da posição atual do utilizador.
- O sistema deve indicar ao utilizador o horário que deve visualizar consoante um itinerário, tendo em conta a hora atual.

Não - Essenciais:

- O sistema deve indicar ao utilizador como ir para as paragens que lhe são mais próximas.
- O utilizador deve ser capaz de visualizar num mapa a área de cobertura da rede de transportes.
- O sistema deve indicar ao utilizador o tempo e custo de viagem para um dado destino.

F. Requisitos Tecnológicos

Quanto aos requisitos tecnológicos, o sistema irá basear-se na tecnologia GPS como método de localização automática dos autocarros que se encontram em movimento e das respectivas paragens, assim como a localização do passageiro. Desta forma, é possível saber quando o autocarro passa num certo ponto da cidade, permitindo assim ao sistema prever o tempo de chegada do autocarro à paragem onde o utilizador se encontra. Os sensores GPS estão instalados nos autocarros para obter continuamente a localização dos autocarros, e estão também presentes nos dispositivos dos passageiros (e.g. *smartphones*) para obtenção a sua localização atual.

Então, resumindo:

- A aplicação irá utilizar a tecnologia GPS como método automático de localização dos utilizadores e dos autocarros.
- A aplicação irá utilizar a API gratuita do Google Maps para poder desempenhar todas as funções necessárias no mapa.

G. Requisitos de Comunicação

Quanto aos requisitos de comunicação, o sistema irá funcionar com base no protocolo SNMP para comunicação e trocas de informação entre as entidades envolvidas no sistema (passageiros, administradores, autocarros e central), sendo isto explicado mais sucintamente nas próximas secções, onde é abordada a arquitectura da aplicação. Além disso, para poder existir troca de informação entre os vários elementos, os utilizadores e os veículos terão que ter uma ligação à internet, quer seja por via *wi-fi* ou GSM. No caso dos passageiros, o sistema também funciona sem internet, mas as suas funcionalidades são mais reduzidas, como já foi explicitado anteriormente.

Em suma:

- O sistema irá funcionar com base no protocolo de gestão SNMP, para comunicação e trocas de informação entre as entidades envolvidas no sistema- Irá também utilizar o protocolo SNMP sistema de gestão, monitorização e base de dados.
- Para que a troca de informação entre entidades seja possível, é necessária uma ligação à internet, seja por via *wi-fi* ou GSM.
- No caso dos passageiros, o sistema também funciona sem internet, mas as suas funcionalidades são mais reduzidas, dependendo também do facto de terem o GPS activo ou não.

H. Requisitos Lógicos

Quanto aos requisitos lógicos, para que o sistema funcione correctamente são necessários dados acerca dos itinerários dos autocarros; localização das paragens de autocarro e respectivos horários dos autocarros que passam nessas paragens; dados de motoristas e autocarros; coroas das zonas da cidade, entre outros.

Além disso, terão que estar configurados ficheiros MIB tanto nos autocarros como na central. Na central terá que existir uma *Central-MIB*, constituída por dois conjuntos de informação distintos: a **informação dinâmica** que está em constante mudança, consoante a informação recolhida dos veículos

(e.g. posição de um dado autocarro); e a **informação estática** da aplicação, isto é, a informação que nunca muda tal como horários fixos, carreiras, paragens, motoristas, autocarros, etc.

Nos veículos terá que estar configurada a *Autocarro-MIB*, que contém informação acerca dos itinerários que o autocarro está a fazer, entre outros dados do veículo. Esta MIB é alterada pelos administradores.

Para que a aplicação funcione correctamente, esta deverá conter os seguintes dados:

- Itinerários dos autocarros e respectivos detalhes.
- Localização das paragens de autocarro.
- Localização dos pontos de interesse.
- Horários dos autocarros de cada itinerário.
- Dados dos motoristas dos autocarros.
- Dados dos autocarros.
- Calendários dos itinerários.
- Coroas das zonas da cidade.

De forma a armazenar toda esta informação, que será posteriormente alterada pelos administradores, terão que estar configuradas as duas MIB's distintas mas que irão trabalhar em conjunto:

- A ***Central-MIB***, que contem dois tipos de informação distintos. Contém a informação que está em constante mudança, consoante a informação recolhida dos veículos (e.g. posição de um dado autocarro), e a informação estática da aplicação, que contem os dados enunciados anteriormente e apenas é alterada pelos administradores.

- A ***Autocarro-MIB***, que contém informação acerca dos itinerários que o autocarro está a percorrer, entre outros dados do veículo. Contém também informação das paragens, pontos de interesse, lista de itinerários, dados dos motoristas, etc.

A estrutura destas MIBs encontra-se apresentada em detalhe no Anexo III – MIBs do Sistema.

I. Requisitos Não-Funcionais

Nesta secção são enunciados os requisitos não-funcionais do sistema, agrupados por categoria de acordo com as oito classes propostas por *Robertson e Robertson* [54].

Aparência

- O sistema deve ter uma interface apelativa.

Usabilidade

- O sistema deve ser de fácil utilização para pessoas com pouca formação no produto.
- O sistema será genérico ao ponto de ser capaz de ser implantado em qualquer cidade ou país, desde que exista uma rede de transportes e as respectivas infra-estruturas necessárias sejam instaladas.

Desempenho

- O sistema deverá estar disponível para uso 24h por dia, 365 dias por ano.

Manutenção e Suporte

Essenciais:

- O sistema central deverá correr em ambiente Android, Linux ou Windows.
- O sistema de administração do sistema deverá correr em qualquer sistema operativo.
- O sistema do passageiro deverá correr em ambiente *mobile*.

Não - Essenciais:

- O sistema deverá ter comentários para apoio ao manuseamento.
- O sistema deverá estar preparado para correr em qualquer língua.

Segurança

Essenciais:

- Apenas os administradores podem visualizar a informação que está central e altera-la.
- O sistema restringirá sempre o acesso a dados considerados como críticos de informação confidencial a utilizadores que não sejam administradores.
- Apenas os administradores podem adicionar, remover ou alterar os dados do sistema (e.g. itinerários, paragens de autocarro, preços, coroas, dados sobre autocarros, motoristas, etc).
- O sistema deve rejeitar a introdução de dados incorrectos.

Não - Essenciais:

- O sistema terá os dados encriptados para serem privados e confidenciais.
- Os administradores devem autenticar-se para aceder ao sistema.

- O sistema não permite o acesso a administradores que não estejam registados no sistema como tais.
- Apenas o administrador pode alterar a sua password e nome de acesso.

Culturais e Políticos

Essenciais:

- O sistema deve ser apresentado em língua portuguesa.

Não - Essenciais:

- O sistema deve ser apresentado em língua inglesa.

J. Diagramas UML

Nesta secção são apresentados os diagramas *use case* que identificam as fronteiras entre os actores - automatizados ou não - e o sistema. Os diagramas são úteis na medida em que, juntando todos os requisitos levantados na fase anterior, é possível desenhar um diagrama onde cada interveniente do sistema pode fazer diferentes operações. Neste caso, apenas se apresentam os casos que são essenciais, uma vez que neste ponto não há certezas se os não essenciais serão ou não implementados.

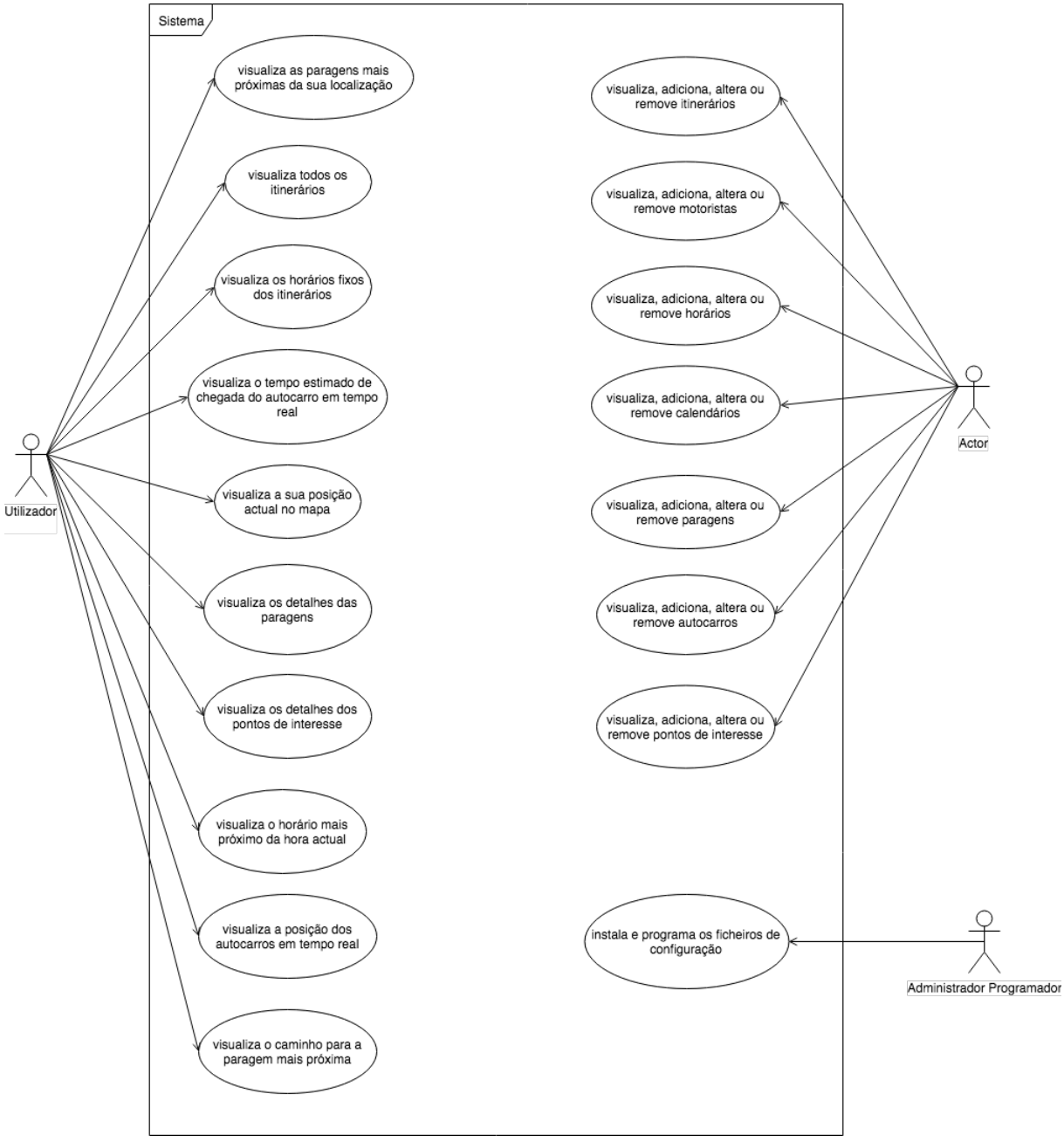


Diagrama UML global do sistema completo.

Seguidamente serão apresentados os diagramas de comportamentos que na fase de planeamento demonstraram ser os mais importantes. O facto de se ter feito os diagramas não significa necessariamente que todos os passos têm que ser seguidos da mesma forma durante a fase de desenvolvimento. No entanto, estes diagramas facilitam consideravelmente o trabalho de programação, uma vez que permitem organizar todos os passos a serem seguidos no decorrer das operações mais relevantes dos utilizadores, desta forma servindo como guia ao desenvolvimento.

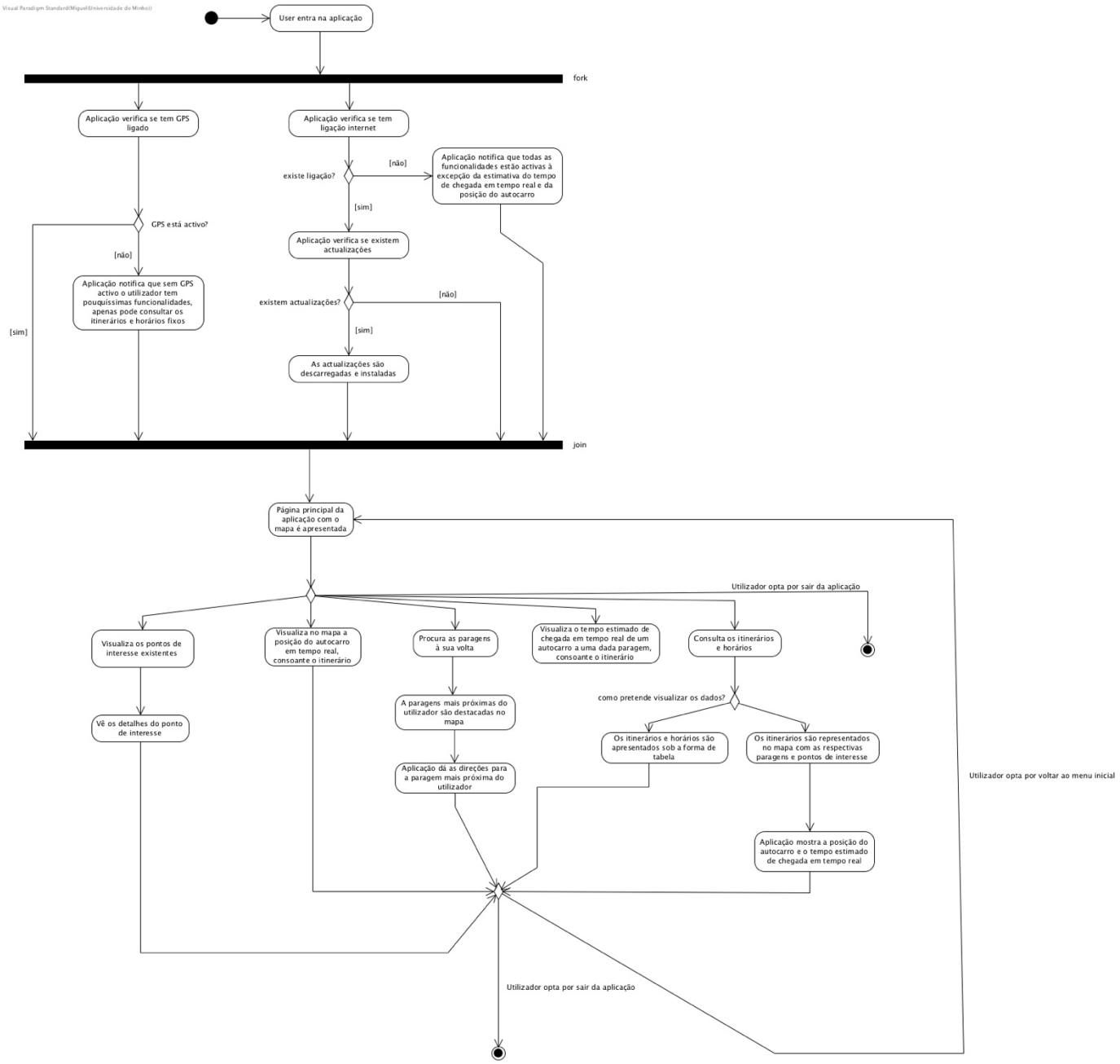


Diagrama de comportamento/atividade do passageiro

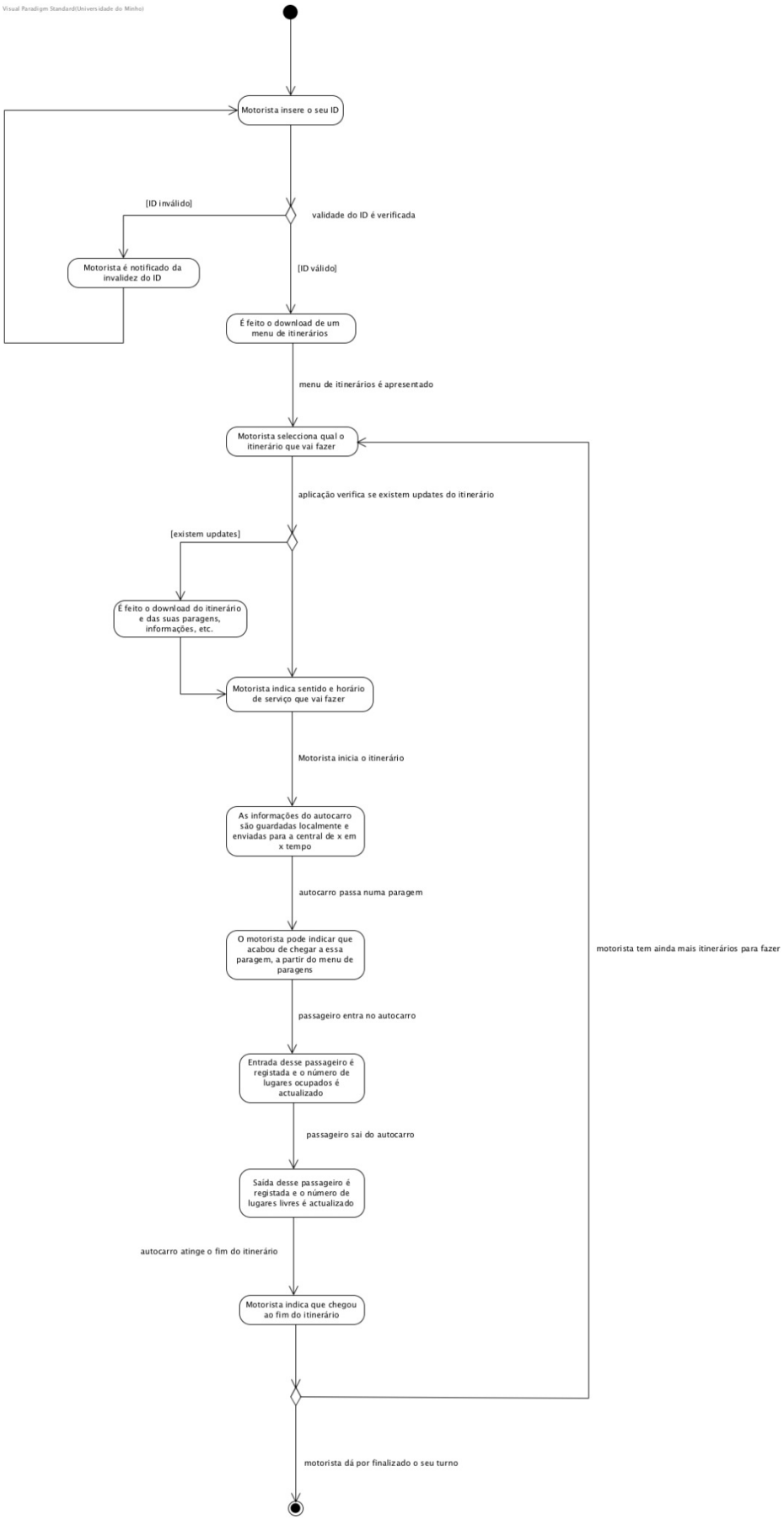


Diagrama de comportamento/atividade do motorista

K. Dados necessários ao funcionamento do sistema

Antes da instrumentação SNMP, torna-se importante saber que informação que cada MIB vai acolher, tendo em conta as operações e necessidades de cada elemento do sistema. Tendo isso em conta, nesta secção são analisados os dados necessários ao funcionamento da aplicação.

Servidor Central

O servidor central é o elemento fulcral do sistema, que permite a troca de informação entre o utilizador e os autocarros. Sendo assim, o servidor central contém todos os dados que alimentam os restantes intervenientes da arquitetura. Contém informação acerca de:

- autocarros e respectivos detalhes (e.g. matrícula, capacidade, etc.).
- horários e itinerários, e em que dias esses itinerários operam.
- localização das paragens de autocarro e os horários dessas paragens.
- lista de paragens e pontos de interesse que constituem um itinerário.
- posição dos autocarros em tempo real.
- tempo de chegada estimado dos autocarros a uma dada paragem.
- número de lugares sentados e número de lugares livres dos autocarros num dado momento.
- coroas das paragens.
- informações de motoristas, entre outros.

Autocarro

No caso do autocarro, é importante saber a informação em tempo real e a informação estática que é pedida ao servidor central, caso exista ligação internet, como é o caso de itinerários, paragens, etc. Dito isto, o autocarro contem informação acerca de:

- dados sobre o próprio autocarro, como matrícula, posição atual, etc.
- informações acerca do motorista que o conduz.
- itinerário que está a fazer num dado momento
- informações de itinerários e paragens, caso não exista ligação para pedir esses dados à central no momento em que vai iniciar o percurso, entre outros.

Passageiro

Para que o subsistema do passageiro funcione correctamente, aquando da instalação da aplicação por parte do utilizador é feito o *download* e a respectiva instalação local de alguns dados. O utilizador contém informação fixa acerca de:

- horários fixos e itinerários.

- paragens e respectivos detalhes.
- pontos de interesse e respectivos detalhes.
- entre outros.

Além dos dados fixos, quando o utilizador estiver a utilizar a aplicação com uma ligação internet, este irá receber informações acerca de:

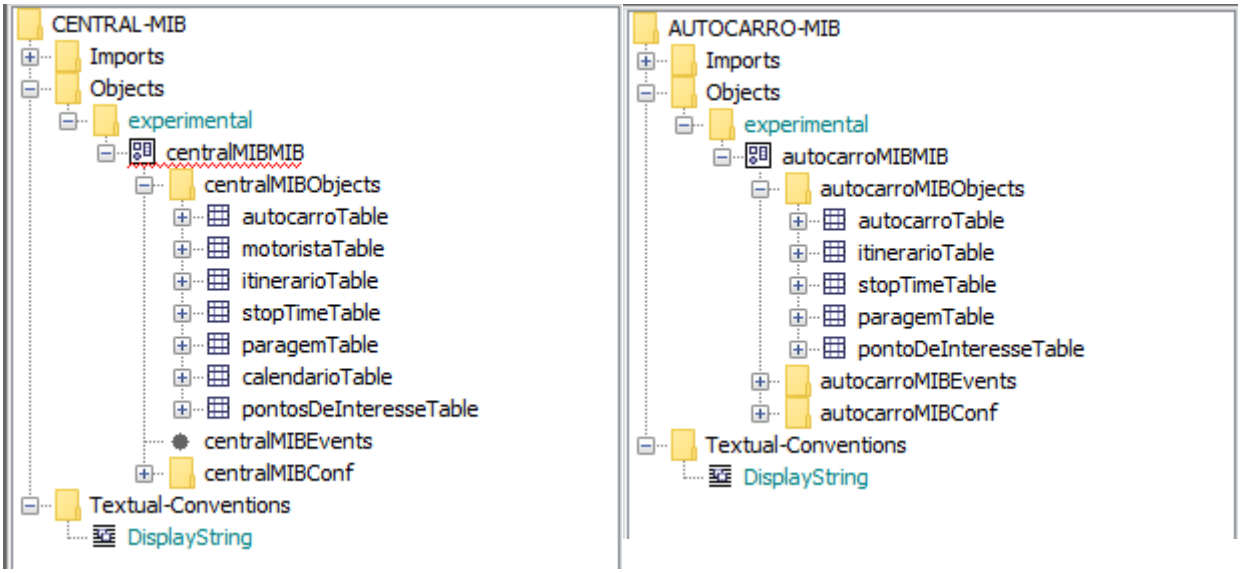
- onde se encontra um autocarro num dado momento.
- o tempo estimado de chegada de um dado autocarro.
- número de lugares sentados e número de lugares livres num dado autocarro.

ANEXO III – MIBs do Sistema

Neste anexo pode ser encontrada a definição da base de dados de objetos para a MIB do sistema central de cada operador de transportes públicos e da MIB do componente veicular, que, no caso deste projeto, se trata de autocarros de passageiros.

Para facilitar a compreensão da estrutura das duas MIBs (do sistema central e do componente veicular), as definições são acompanhadas de ilustrações da sua definição feita na aplicação MIB Designer. Para acesso à listagem na linguagem formal SMIv2/ASN.1 por favor contactar o autor.

As definições das variáveis de gestão desta base de dados estão debaixo da mesma entrada duma única MIB experimental em que o *root object identifier prefix* escolhido foi o OID ***experimental*** (*1.3.6.1.3*) e os *root object identifier suffix* foram o ***999*** (para a MIB do sistema central) e ***9999*** (*para a MIB do componente veicular*).

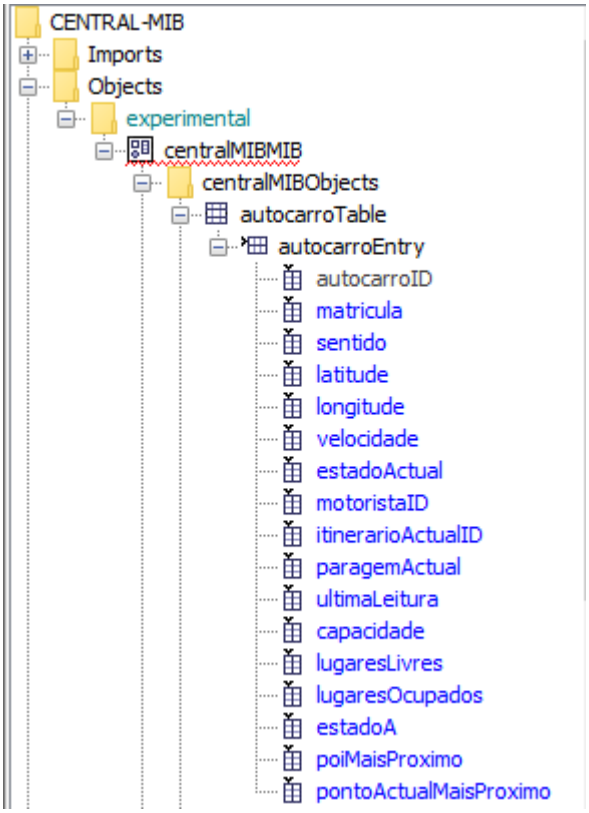


Estrutura das duas MIBs do sistema.

Segue-se uma descrição sucinta dos objetos de cada uma das tabelas relevantes (como todas as tabelas da MIB do componente veicular estão também definidas na MIB do sistema central, só é apresentada a definição constante desta última MIB). De notar que a cinzento estão indicados os objetos que são chave de cada uma das tabelas.

A. Tabela de Autocarros

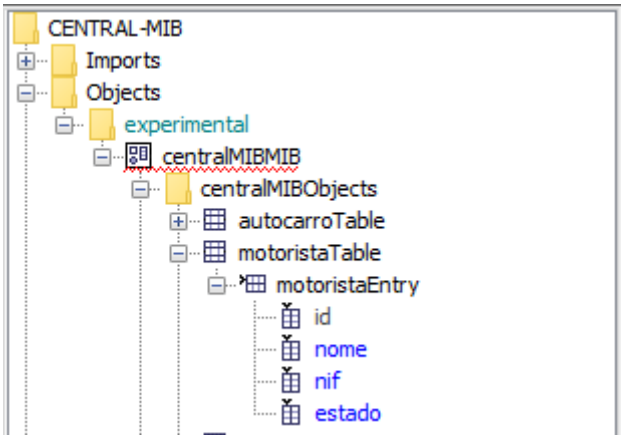
1. **autocarroID** (*experimental.999.1.1.1.1*) – identifica unicamente cada autocarro na base de dados. Este campo convém ser do tipo inteiro. Por exemplo, tendo em conta que este campo é identificado pelo OID *1.3.6.1.3.999.1.1.1*, o OID *1.3.6.1.3.999.1.1.1.**1*** identifica o autocarro com *id='1'* na base de dados.
2. **matricula** (*experimental.999.1.1.1.2*) – matrícula do autocarro.
3. **sentido** (*experimental.999.1.1.1.3*) – sentido da viagem no qual o autocarro está a operar.
4. **latitude** (*experimental.999.1.1.1.4*) – coordenadas latitude onde o autocarro se encontra num dado momento.
5. **longitude** (*experimental.999.1.1.1.5*) – coordenadas longitude onde o autocarro se encontra num dado momento.
6. **velocidade** (*experimental.999.1.1.1.6*) – velocidade do autocarro num dado momento.
7. **estadoAtual** (*experimental.999.1.1.1.7*) – representa o estado atual do autocarro.
8. **motoristaID** (*experimental.999.1.1.1.8*) – id do motorista que conduz atualmente o autocarro.
9. **itinerarioAtualID** (*experimental.999.1.1.1.9*) – itinerário que o autocarro está a percorrer atualmente.
10. **paragemAtual** (*experimental.999.1.1.1.10*) – paragem onde o autocarro se encontra atualmente.
11. **ultimaLeitura** (*experimental.999.1.1.1.11*) – momento da última leitura que foi feita pelo *SNMP Relay* do autocarro. Basicamente é a última vez que o autocarro comunicou e trocou informações com o servidor central.
12. **capacidade** (*experimental.999.1.1.1.12*) – capacidade do autocarro, isto é, quantos lugares o autocarro possui.
13. **lugaresLivres** (*experimental.999.1.1.1.13*) – lugares livres do autocarro num dado momento.
14. **lugaresOcupados** (*experimental.999.1.1.1.14*) – lugares ocupados do autocarro num dado momento.
15. **estadoA** (*experimental.999.1.1.1.15*) – estado dos dados: "*Activo*" ou "*inactivo*".
16. **poiMaisProximo** (*experimental.999.1.1.1.16*) – id do ponto de interesse mais próximo.
17. **pontoAtualMaisProximo** (*experimental.999.1.1.1.17*) – ponto atual mais próximo do autocarro.



Estrutura da tabela de objetos para os autocarros.

B. Tabela de Motoristas

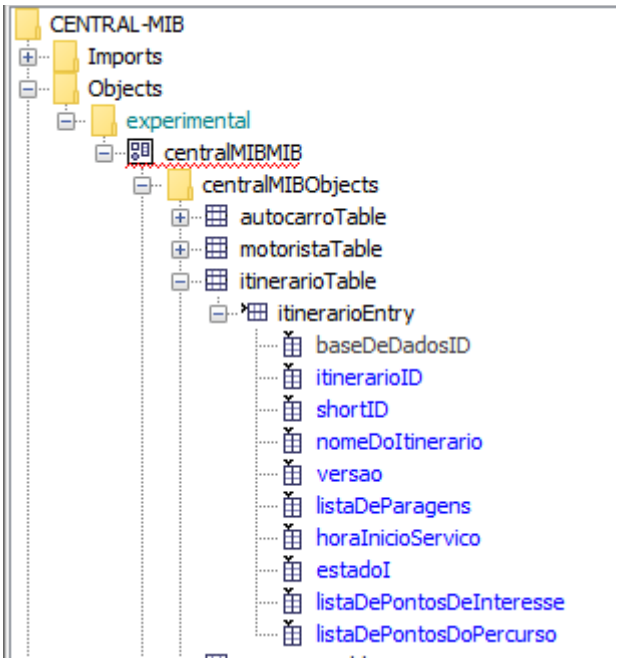
1. **id** (*experimental.999.1.2.1.1*) – id que identifica unicamente cada motorista.
2. **nome** (*experimental.999.1.2.1.2*) – nome do motorista.
3. **nif** (*experimental.999.1.2.1.3*) – número de identificação fiscal que identifica cada motorista.
4. **estado** (*experimental.999.1.2.1.4*) – estado da informação do motorista. "*Activo*" ou "*Inactivo*".
5.



Estrutura da tabela de objetos para os motoristas.

C. Tabela de Itinerários

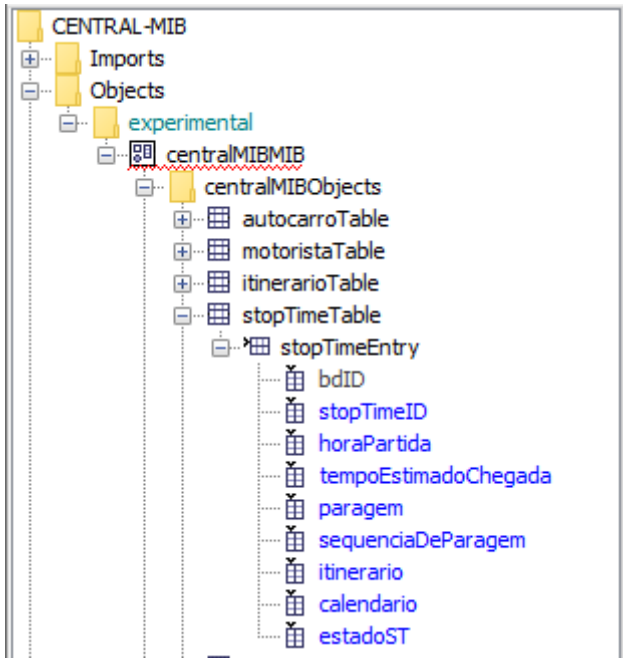
1. **baseDeDadosID** (*experimental.999.1.3.1.1*) – inteiro que permite rastrear o itinerário na BD da MIB.
2. **itinerarioID** (*experimental.999.1.3.1.2*) – identifica unicamente um itinerário. Isto porque um itinerário pode ser operado por vários autocarros, a horas diferentes. Exemplo: "24_1" (itinerário_trip). A segunda carreira do mesmo itinerário teria como id "24_2", e assim sucessivamente.
3. **shortID** (*experimental.999.1.3.1.3*) – short id do itinerário. Exemplo: "24".
4. **nomeDoItinerario** (*experimental.999.1.3.1.4*) – nome do itinerário.
5. **versao** (*experimental.999.1.3.1.5*) – serve para comparar versões da MIB presente na central com a MIB presente no autocarro.
6. **listaDeParagens** (*experimental.999.1.3.1.6*) – lista de paragens por onde um itinerário passa.
7. **horaInicioServico** (*experimental.999.1.3.1.7*) – hora a que o serviço começa.
8. **estadoI** (*experimental.999.1.3.1.8*) – estado do itinerário.
9. **listaDePontosDeInteresse** (*experimental.999.1.3.1.9*) – lista de pontos de interesse do itinerário.
10. **listaDePontosDoPercurso** (*experimental.999.1.3.1.10*) – lista de pontos do percurso de um itinerário.



Estrutura da tabela de objetos para itinerários.

D. Tabela de Horários

1. **bdID** (*experimental.999.1.4.1.1*) – identifica unicamente cada stop time.
2. **stopTimeID** (*experimental.999.1.4.1.2*) – descreve o id do stop time. Exemplo: primeira paragem do itinerário "24_1" vai ter o id "24_1_1"; a segunda terá o id "24_1_2" e assim sucessivamente.
3. **horaPartida** (*experimental.999.1.4.1.3*) – hora a que o autocarro sai desta paragem.
4. **tempoEstimadoChegada** (*experimental.999.1.4.1.4*) – estimativa do tempo que o autocarro leva a chegar a esta paragem.
5. **paragem** (*experimental.999.1.4.1.5*) – paragem onde o autocarro pára.
6. **sequenciaDeParagem** (*experimental.999.1.3.1.6*) – sequência atual de paragens do itinerário.
7. **itinerario** (*experimental.999.1.4.1.7*) – itinerário que passa nesta paragem, a estas horas.
8. **calendario** (*experimental.999.1.4.1.8*) – dias da semana em que o autocarro passa nesta paragem, a estas horas.
9. **estadoST** (*experimental.999.1.4.1.9*) – estado dos dados. "*Activo*" ou "*Inactivo*".

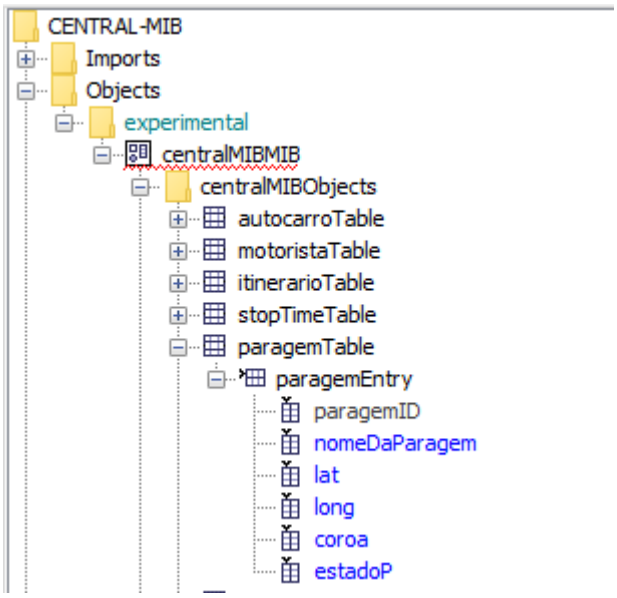


Estrutura da tabela de objetos para os horários.

E. Tabela de Paragens

1. **paragemID** (*experimental.999.1.5.1.1*) – id único da paragem.
2. **nomeDaParagem** (*experimental.999.1.5.1.2*) – nome da paragem.
3. **lat** (*experimental.999.1.5.1.3*) – coordenadas latitude da paragem.
4. **long** (*experimental.999.1.5.1.4*) – coordenadas longitude da paragem.

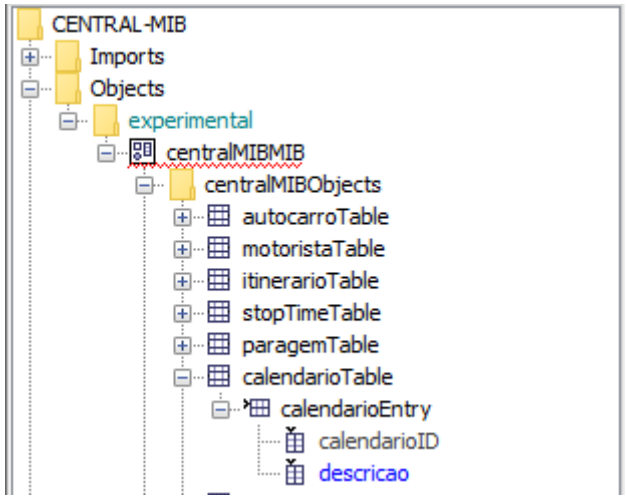
5. **coroa** (*experimental.999.1.5.1.5*) – coroa da paragem.
6. **estadoP** (*experimental.999.1.5.1.6*) – estado dos dados. "*Activo*" ou "*Inactivo*".



Estrutura da tabela de objetos para as paragens.

F. Tabela de Calendários

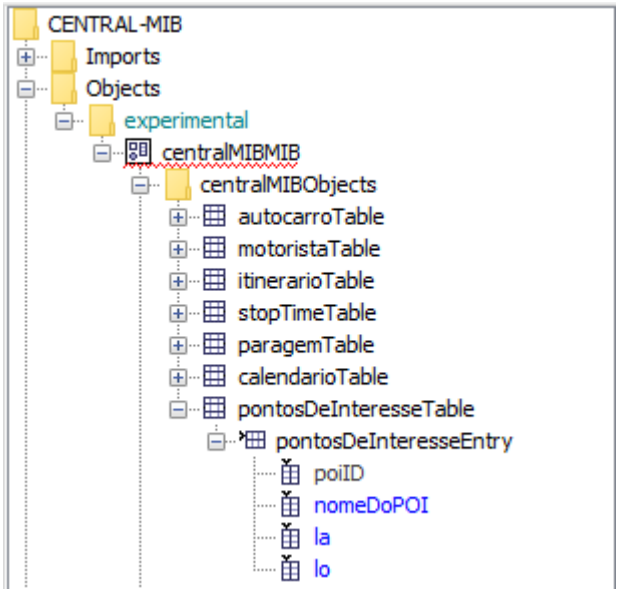
1. **calendarioID** (*experimental.999.1.6.1.1*) – id do calendário.
2. **descricao** (*experimental.999.1.6.1.2*) – descrição do calendário. Exemplo: "*segunda a sábado*" ou "*domingos e feriados*".



Estrutura da tabela de objetos para os calendários.

G. Tabela de Pontos de Interesse

1. **poiID** (*experimental.999.1.7.1.1*) – id do ponto de interesse.
2. **nomeDoPOI** (*experimental.999.1.7.1.2*) – nome do ponto de interesse.
3. **la** (*experimental.999.1.7.1.3*) – coordenadas latitude do ponto de interesse.
4. **lo** (*experimental.999.1.7.1.4*) – coordenadas longitude do ponto de interesse.



Estrutura da tabela de objetos para os pontos de interesse.

ANEXO IV – Instrumentação SNMP

Neste anexo podem ser encontrados os detalhes de implementação da instrumentação SNMP, nomeadamente dos módulos dos agentes, dos gestores e das MIBs.

Após a definição das MIBs que irão alocar a informação, o passo seguinte passou por gerar as MIBs no formato de texto, para mais tarde serem integradas no software AgenPro [51]. De seguida, através desse software faz-se a exportação da MIB no formato Java, que será mais tarde controlada pelo agente. De forma a ser mais claro, instrumentação é basicamente a construção do agente e manager SNMP e o respetivo tratamento da comunicação entre os dois.

Apesar de o estudo do funcionamento da tecnologia SNMP não ser complexo, o mesmo não se pode dizer da sua implementação. A instrumentação foi feita com base na biblioteca SNMP4J, que com base no *Java* mas utiliza uma API com conceitos completamente distintos daqueles que foram abordados até agora. Esta foi definitivamente a parte mais morosa da dissertação, uma vez que a investigação passou por entender a API e saber como usá-la, algo que provou ser bastante custoso pois não existe documentação acerca deste tema, ou aquela que existe não é suficientemente clara. Neste protótipo, a versão SNMP utilizada foi a versão 2c, que não inclui mecanismos de segurança (a passagem para a versão v2 ou v3, de forma a incluir mecanismos de segurança não era um objetivo do projeto atual, mas a sua utilização seria transparente em termos de arquitetura do sistema).

A. Agente presente no Servidor Central

O primeiro passo foi a construção do agente que controla a *Central-MIB*. Este agente está ligado a uma base de dados SQL que corre num servidor local, onde os dados são guardados em duplicado. Isto porque a MIB não guarda definitivamente os dados, estes têm que ser povoados a cada vez que o agente é iniciado. Caso o agente nunca seja interrompido, não há a necessidade de recarregar os dados.

```
public void loadDataFromBD()
{
    this.loadMotoristasFromBD();
    this.loadAutocarrosFromBD();
    this.loadItinerariosFromBD();
    this.loadStopTimesFromBD();
    this.loadParagensFromBD();
    this.loadCalendariosFromBD();
    this.loadPontosDeInteresseFromBD();
}
```

Quando o agente é iniciado, importa os dados do servidor SQL e utiliza-os para povoar a MIB. Visto que existem várias tabelas, terão que ser feitas tantas importações quanto tabelas. No entanto, como se pode ver pelo código seguinte, a estrutura é transversal a todas as tabelas.

```
public void loadAutocarrosFromBD()
{
    MOTable<AutocarroEntryRow,MOColumn,MOTableModel<AutocarroEntryRow>> autocarroEntry = mib.getAutocarroEntry();
    MOTableModel modelAutocarro = (MOTableModel)autocarroEntry.getModel();

    try {
        ResultSet rs = sql.manSelect("autocarro", new String[]{"*"}, null, new String[] {"id ASC"});
        while(rs.next())
        {
            Variable [] vars = new Variable[] {new Integer32(sql.getInteger(rs, "id")),
                                               new OctetString(rs.getString("matricula")),
                                               new Integer32(sql.getInteger(rs, "sentido")),
                                               new OctetString(rs.getString("latitude")),
                                               new OctetString(rs.getString("longitude")),
                                               new Integer32(sql.getInteger(rs, "velocidade")),
                                               new OctetString(rs.getString("estadoActual")),
                                               new Integer32(sql.getInteger(rs, "motoristaActual")),
                                               new Integer32(sql.getInteger(rs, "itinerarioActual")),
                                               new Integer32(sql.getInteger(rs, "paragemActual")),
                                               new OctetString(rs.getString("ultimaLeitura")),
                                               new Integer32(sql.getInteger(rs, "capacidade")),
                                               new Integer32(sql.getInteger(rs, "lugaresLivres")),
                                               new Integer32(sql.getInteger(rs, "lugaresOcupados")),
                                               new OctetString(rs.getString("estado")),
                                               new Integer32(sql.getInteger(rs, "poiMaisProximo")),
                                               new OctetString(rs.getString("pontoActualMaisProximo"))};

            OID index = new OID("");
            index.append(sql.getInteger(rs,"id"));

            autocarroEntry.addNewRow(index, vars);
        }
    } catch (SQLException ex) {
        System.out.println(ex.toString());
    }
}
```

Este é o caso de inserção de autocarros na MIB, após os dados serem importados da base de dados SQL do servidor. O exemplo abaixo é o caso de inserção de paragens.

```
public void loadParagensFromBD()
{
    MOTable<ParagemEntryRow,MOColumn,MOTableModel<ParagemEntryRow>> paragemEntry = mib.getParagemEntry();
    MOTableModel modelParagem = (MOTableModel)paragemEntry.getModel();

    try {
        ResultSet rs = sql.manSelect("paragem", new String[]{"*"}, null, new String[] {"id ASC"});
        while(rs.next())
        {
            Variable [] vars = new Variable[] {new Integer32(sql.getInteger(rs, "id")),
                                               new OctetString(rs.getString("nome")),
                                               new OctetString(rs.getString("latitude")),
                                               new OctetString(rs.getString("longitude")),
                                               new Integer32(sql.getInteger(rs, "coroa")),
                                               new OctetString(rs.getString("estado"))};

            OID index = new OID("");
            index.append(sql.getInteger(rs,"id"));

            paragemEntry.addNewRow(index, vars);
        }
    } catch (SQLException ex) {
        System.out.println(ex.toString());
    }
}
```

No entanto, à medida que o agente vai correndo a informação presente na MIB vai sendo continuamente alterada. Dito isto, é necessário guardar essas alterações na base de dados SQL do servidor, para os dados serem posteriormente recuperados no caso de ocorrer alguma falha no funcionamento do agente. A base de dados SQL funciona como um *backup* de dados.

```
public void readAndSaveRows()
{
    this.readAndSaveAutocarroRows();
    this.readAndSaveMotoristaRows();
    this.readAndSaveItinerarioRows();
    this.readAndSaveStopTimeRows();
    this.readAndSaveParagemRows();
    this.readAndSaveCalendarioRows();
}
```

Tal como na inserção de dados, é necessário fazer tantas gravações na base de dados quanto o número de tabelas. De forma a fazer as gravações, é necessário primeiro percorrer a MIB, recolher os valores nela presentes e enviá-los para a base de dados local SQL.

```java
public void readAndSaveItinerarioRows()
{
    MOTable<ItinerarioEntryRow,MOColumn,MOTableModel<ItinerarioEntryRow>> itinerarioEntry = mib.getItinerarioEntry();
    MOTableModel modelItinerario = (MOTableModel)itinerarioEntry.getModel();

    String id;
    String itinerarioID;
    String shortID;
    String nome;
    String versao;
    String listaDeParagens;
    String horaInicioServico;
    String estado;
    String listaDePontosDeInteresse;
    String listaDePontosDoPercurso;

    int size = modelItinerario.getRowCount();

    sql.manDeleteAll("itinerario");

    for(int i=1; i<size+1; i++)
    {
        id                      = "1.3.6.1.3.999.1.3.1.1."+i;
        itinerarioID            = "1.3.6.1.3.999.1.3.1.2."+i;
        shortID                 = "1.3.6.1.3.999.1.3.1.3."+i;
        nome                    = "1.3.6.1.3.999.1.3.1.4."+i;
        versao                  = "1.3.6.1.3.999.1.3.1.5."+i;
        listaDeParagens         = "1.3.6.1.3.999.1.3.1.6."+i;
        horaInicioServico       = "1.3.6.1.3.999.1.3.1.7."+i;
        estado                  = "1.3.6.1.3.999.1.3.1.8."+i;
        listaDePontosDeInteresse = "1.3.6.1.3.999.1.3.1.9."+i;
        listaDePontosDoPercurso  = "1.3.6.1.3.999.1.3.1.10."+i;

        int number = Integer.parseInt(itinerarioEntry.getValue(new OID(id)).toString());
        String itiID = itinerarioEntry.getValue(new OID(itinerarioID)).toString();
        int shID = Integer.parseInt(itinerarioEntry.getValue(new OID(shortID)).toString());
        String name = itinerarioEntry.getValue(new OID(nome)).toString();
        int version = Integer.parseInt(itinerarioEntry.getValue(new OID(versao)).toString());
        String lista = itinerarioEntry.getValue(new OID(listaDeParagens)).toString();
        String hora = itinerarioEntry.getValue(new OID(horaInicioServico)).toString();
        String state = itinerarioEntry.getValue(new OID(estado)).toString();
        String listPOI = itinerarioEntry.getValue(new OID(listaDePontosDeInteresse)).toString();
        String listPoints = itinerarioEntry.getValue(new OID(listaDePontosDoPercurso)).toString();

        sql.registarItinerarioNaBD(number, itiID, shID, name, version, lista, hora, state, listPOI, listPoints);
    }
}
```

É necessário aceder primeiro ao modelo e às *entries* ou entradas da tabela para fazer a leitura dos valores presentes na tabela. Depois, apaga-se a base de dados – no caso acima, todos os itinerários – e começa-se por percorrer todos os OID's que contêm dados de itinerários, registando de seguida esses dados na base de dados SQL. Quando atingimos o número de linhas da tabela de itinerários da MIB, sabemos que chegamos ao fim pois já percorremos todas as entradas da tabela.

```
public void readAndSaveAutocarroRows()
{
    MOTable<AutocarroEntryRow,MOColumn,MOTableModel<AutocarroEntryRow>> autocarroEntry = mib.getAutocarroEntry();
    MOTableModel modelAutocarro = (MOTableModel)autocarroEntry.getModel();

    String id;
    String matricula;
    String sentido;
    String latitude;
    String longitude;
    String velocidade;
    String estadoActual;
    String motorista;
    String itinerario;
    String paragem;
    String ultimaLeitura;
    String capacidade;
    String lugaresLivres;
    String lugaresOcupados;
    String estado;
    String poiMaisProximo;
    String pontoActualMaisProximo;

    int size = modelAutocarro.getRowCount();

    sql.manDeleteAll("autocarro");

    for(int i=1; i<size+1; i++)
    {
        id                      = "1.3.6.1.3.999.1.1.1.1."+i;
        matricula               = "1.3.6.1.3.999.1.1.1.2."+i;
        sentido                 = "1.3.6.1.3.999.1.1.1.3."+i;
        latitude                = "1.3.6.1.3.999.1.1.1.4."+i;
        longitude               = "1.3.6.1.3.999.1.1.1.5."+i;
        velocidade              = "1.3.6.1.3.999.1.1.1.6."+i;
        estadoActual            = "1.3.6.1.3.999.1.1.1.7."+i;
        motorista               = "1.3.6.1.3.999.1.1.1.8."+i;
        itinerario              = "1.3.6.1.3.999.1.1.1.9."+i;
        paragem                 = "1.3.6.1.3.999.1.1.1.10."+i;
        ultimaLeitura           = "1.3.6.1.3.999.1.1.1.11."+i;
        capacidade              = "1.3.6.1.3.999.1.1.1.12."+i;
        lugaresLivres           = "1.3.6.1.3.999.1.1.1.13."+i;
        lugaresOcupados         = "1.3.6.1.3.999.1.1.1.14."+i;
        estado                  = "1.3.6.1.3.999.1.1.1.15."+i;
        poiMaisProximo          = "1.3.6.1.3.999.1.1.1.16."+i;
        pontoActualMaisProximo  = "1.3.6.1.3.999.1.1.1.17."+i;

        int number = Integer.parseInt(autocarroEntry.getValue(new OID(id)).toString());
        String matr = autocarroEntry.getValue(new OID(matricula)).toString();
        int sent = Integer.parseInt(autocarroEntry.getValue(new OID(sentido)).toString());
        String la = autocarroEntry.getValue(new OID(latitude)).toString();
        String lo = autocarroEntry.getValue(new OID(longitude)).toString();
        int vel = Integer.parseInt(autocarroEntry.getValue(new OID(velocidade)).toString());
        String ea = autocarroEntry.getValue(new OID(estadoActual)).toString();
        int mot = Integer.parseInt(autocarroEntry.getValue(new OID(motorista)).toString());
        int iti = Integer.parseInt(autocarroEntry.getValue(new OID(itinerario)).toString());
        int stop = Integer.parseInt(autocarroEntry.getValue(new OID(paragem)).toString());
        String leitura = autocarroEntry.getValue(new OID(ultimaLeitura)).toString();
        int cap = Integer.parseInt(autocarroEntry.getValue(new OID(capacidade)).toString());
        int livres = Integer.parseInt(autocarroEntry.getValue(new OID(lugaresLivres)).toString());
        int ocupados = Integer.parseInt(autocarroEntry.getValue(new OID(lugaresOcupados)).toString());
        String state = autocarroEntry.getValue(new OID(estado)).toString();
        String poi = autocarroEntry.getValue(new OID(poiMaisProximo)).toString();
        String point = autocarroEntry.getValue(new OID(pontoActualMaisProximo)).toString();


        sql.registarAutocarroNaBD(number, matr, sent, la, lo, vel, ea, mot, iti, stop, leitura, cap, livres, ocupados, state, poi, point);
    }
}
```

Como se pode observar pela listagem acima, o caso dos autocarros segue o mesmo padrão, mudando apenas os campos e os OID's da tabela onde se encontram. Uma vez que os *id's* da base de dados SQL são os mesmos que os da MIB, temos a garantia que a ordem é mantida.

Permissões do Agente

Uma vez que a versão do SNMP utilizada é a *snmpv2c,* foi necessário definir a *community string* para poder identificar os pedidos autorizados provenientes dos managers.

```
@Override
protected void addCommunities(SnmpCommunityMIB scmib)
{
    Variable[] com2sec = new Variable[]{
        new OctetString("public"), // community name
        new OctetString("cpublic"), // security name
        getAgent().getContextEngineID(), // local engine ID
        new OctetString("public"), // default context name
        new OctetString(), // transport tag
        new Integer32(StorageType.nonVolatile), // storage type
        new Integer32(RowStatus.active) // row status
    };

    MOTableRow row = scmib.getSnmpCommunityEntry().createRow(
                        new OctetString("public2public").toSubIndex(true), com2sec);

    scmib.getSnmpCommunityEntry().addRow((SnmpCommunityMIB.SnmpCommunityEntryRow) row);
}
```

Quanto às permissões do agente, este permite ao manager acesso para escrita e leitura na *Central-MIB*.

```
/**
 * Minimal View Based Access Control (VACM)
 * Adds initial VACM configuration.
 * http://www.faqs.org/rfcs/rfc2575.html
 * @param vacm
*/
@Override
protected void addViews(VacmMIB vacm)
{
    vacm.addGroup(
                    SecurityModel.SECURITY_MODEL_SNMPv2c,
                    new OctetString("cpublic"),
                    new OctetString("v1v2group"),
                    StorageType.nonVolatile
                );

    vacm.addAccess(
                    new OctetString("v1v2group"),
                    new OctetString("public"),
                    SecurityModel.SECURITY_MODEL_ANY,
                    SecurityLevel.NOAUTH_NOPRIV,
                    MutableVACM.VACM_MATCH_EXACT,
                    new OctetString("fullReadView"),
                    new OctetString("fullWriteView"),
                    new OctetString("fullNotifyView"),
                    StorageType.nonVolatile
                );

    vacm.addViewTreeFamily(
                            new OctetString("fullReadView"),
                            new OID("1.3"),
                            new OctetString(),
                            vacmMIB.vacmViewIncluded,
                            StorageType.nonVolatile
                        );
    vacm.addViewTreeFamily(
                            new OctetString("fullWriteView"),
                            new OID("1.3"),
                            new OctetString(),
                            vacmMIB.vacmViewIncluded,
                            StorageType.nonVolatile
                        );
}
```

No entanto, no que toca a inserir novas linhas na base de dados, que é um dos requisitos para o administrador do sistema, há um pequeno detalhe que merece ser destacado. O SNMP não funciona como o SQL, não deixa criar automaticamente novas linhas nas tabelas aquando a inserção de novos dados.

Durante a definição das MIBs, para cada objeto foi necessário escolher qual o tipo de acesso, que pode ser:

- ***read-write***.
- ***not-acessible.***
- ***read-only.***
- ***read-create.***
- ***accessible-for-notify.***

Os tipos mais utilizados foram o *read-only* para os *ids* das tabelas e o *read-write*, para os restantes campos. Ora, tendo em conta que o *MibDesigner* obriga a que o o *Index Object* (i.e. o id que identifica cada tabela) seja *read-only*, é fácil concluir que não será então possível alterar esse campo nem criar novos, uma vez que não tem permissão *read-write* ou *read-create*.

Após um largo número de tentativas, foi possível concluir que não era possível criar novas linhas na tabela a pedido do manager (pedido *SET*), uma vez que o SNMP não deixava criar novas linhas, pelo menos neste contexto onde se utiliza o SNMP como base de dados e é necessário existir um id para distinguir cada entrada da base de dados. O SNMP apenas permitiu alterar dados das tabelas (*SET*) e obter informação (*GET*). Quando se tentava fazer um *SET* com um OID que ainda não existia na tabela, o SNMP indicava que não havia permissão para criar novas linhas da base de dados, devido à permissão *read-only* do id da tabela.

Tentou utilizar-se a permissão *read-create* em todos os campos à excepção do id da tabela, mas mesmo assim não funcionava pois o SNMP obriga sempre a que o *índex* da tabela seja do tipo *read-only*. Segundo o SNMP, se uma tabela tem mais do que uma entrada, uma das colunas da MIB tem que assumir o papel de *índex object* para que haja distinção entre as várias linhas. No entanto, o agente tem a capacidade de criar novas entradas, visto que é o agente que controla a MIB.

Surgiram então duas alternativas possíveis para contornar este problema:

1. Criar um elevado número de linhas vazias (à excepção do *id*) prontas a serem preenchidas. Assim, no momento em que o administrador adiciona um itinerário novo, o manager envia um pedido *SET* com essas informações ao agente central, e este percorre a MIB e coloca os dados desse itinerário na primeira entrada livre que encontrar.
2. Criar uma via de comunicação entre o agente e o manager, para que o agente interprete esses pacotes vindos do manager de forma distinta dos pacotes *GET* e *SET* (de alteração, não de criação). Desta forma, o agente sabe que esse *SET* é para criação, recolhe os dados contidos no pacote e coloca-os numa nova linha da base de dados.

A opção adoptada para solucionar o problema foi a opção 2. Quando o administrador quer inserir uma nova entrada de uma dada tabela na base de dados (e.g. um itinerário), o manager que corre no seu sistema envia um pacote UDP com as informações desse itinerário e insere-o na MIB. De seguida, tal como supracitado, o agente percorre a MIB e guarda essas alterações na base de dados SQL.

```
    // cria um socket UDP
    //DatagramSocket socket = new DatagramSocket(6789);
    DatagramSocket socket = new DatagramSocket(6788);
    byte[] buffer = new byte[1000];
    System.out.println("Servidor está a aguardar pedidos...");


while(true)
{
    agent.readAndSaveRows();

    // cria datagrama para receber request do cliente
    DatagramPacket request = new DatagramPacket(buffer, buffer.length);
    socket.receive(request);
    System.out.println("*** Request recebido de: " + request.getAddress());

    byte[] msg = request.getData();

    String s = new String(request.getData());

    String recorte[] = s.split("\\|");

    //resp = processaMensagem(recorte[]);
    System.out.println("Recorte[0]: " +recorte[0]);
    System.out.println("Recorte[1]: " +recorte[1]);

    String resp = "Erro!";

    resp = agent.processaMensagem(recorte);

    //envia resposta
    DatagramPacket response = new DatagramPacket(resp.getBytes(), resp.length(), request.getAddress(), request.getPort());
    socket.send(response);
}
```

Como pode ser observado no código acima, no momento em que o agente começa a correr no servidor é criado um *socket* UDP numa porta, ficando à escuta de pedidos provenientes dos vários managers que correm no sistema. Caso receba um pacote com um pedido de criação, a mensagem nele contida é

recuperada e processada. Após o sucesso ou insucesso da operação, uma resposta é enviada de volta para o manager, que apresenta uma mensagem na tela do administrador, tendo em conta o resultado da operação. Neste caso sabemos que o manager que recebe esta mensagem é aquele que corre no sistema de administração, visto que este é o único que tem permissão para criar (*GET*) ou alterar dados (*SET*) da base de dados. No caso do manager presente no autocarro ou no telemóvel do utilizador, as únicas operações possíveis são as de leitura (*GET*).

```
public String processaMensagem(String[] recorte)
{
    String resp="Ocorreu um erro";

    if(recorte[0].equals("autocarro"))
    {
        int newID = 0;
        String matricula = recorte[1];
        int sentido = 1;
        String latitude ="0";
        String longitude ="0";
        int velocidade = 0;
        int estadoActual = 2;
        int motoristaID = 0;
        int itinerarioActual = 0;
        int paragemActual = 0;
        String ultimaLeitura = "09-08-2017";
        int capacidade = Integer.parseInt(recorte[2]);
        int lugaresLivres = capacidade;
        int lugaresOcupados = 0;
        String estado = "activo";
        int poiMaisProximo = 0;
        String pontoActualMaisProximo = "sem registo";

        Variable[] values = new Variable[] {new Integer32(newID),
                                            new OctetString(matricula),
                                            new Integer32(sentido),
                                            new OctetString(latitude),
                                            new OctetString(longitude),
                                            new Integer32(velocidade),
                                            new Integer32(estadoActual),
                                            new Integer32(motoristaID),
                                            new Integer32(itinerarioActual),
                                            new Integer32(paragemActual),
                                            new OctetString(ultimaLeitura),
                                            new Integer32(capacidade),
                                            new Integer32(lugaresLivres),
                                            new Integer32(lugaresOcupados),
                                            new OctetString(estado),
                                            new Integer32(poiMaisProximo),
                                            new OctetString(pontoActualMaisProximo)};

        this.createRowGeneral("autocarro", values);
        resp = "Autocarro adicionado com sucesso!";
    }

    if(recorte[0].equals("motorista"))
    {
```

Após receber o pacote UDP proveniente do manager, o agente da central processa a mensagem contida nesse pacote de forma a inserir a informação na MIB. A estrutura deste pacote será vista em mais detalhe na secção seguinte. Contudo, para entendermos como a mensagem é processada, a mensagem do pacote segue a seguinte estrutura:

***nome_tabela* | *coluna_1* | *coluna_2* | *coluna_3 (..)* |**

Ora, sabendo o nome da tabela onde temos que inserir, conseguimos adivinhar o número de colunas que vêm a seguir. No caso do código acima, no momento da criação do autocarro apenas necessitamos de dois campos: a matrícula e a capacidade. Todos os outros campos, como é o exemplo dos dados do autocarro em tempo real, são povoados com valores por definição. O mesmo acontece com as outras tabelas.

```
if(recorte[0].equals("itinerario"))
{
    int newID = 0;
    String itinerarioID =  recorte[1];
    int shortID = Integer.parseInt(recorte[2]);
    String nome = recorte[3];
    int versao = 1;
    String listaParagens = recorte[4];
    String horaInicioServico = recorte[5];
    String estado = "activo";
    String listaDePontosDeInteresse = recorte[6];
    String listaDePontosDoPercurso = recorte[7];


    Variable[] values = new Variable[] {new Integer32(newID),
                                        new OctetString(itinerarioID),
                                        new Integer32(shortID),
                                        new OctetString(nome),
                                        new Integer32(versao),
                                        new OctetString(listaParagens),
                                        new OctetString(horaInicioServico),
                                        new OctetString(estado),
                                        new OctetString(listaDePontosDeInteresse),
                                        new OctetString(listaDePontosDoPercurso)};


    this.createRowGeneral("itinerario", values);
    resp = "Itinerario adicionado com sucesso!";
}
```

Ora, sabendo então qual a tabela onde o agente tem que inserir e tendo os dados necessários, procede-se à criação de uma nova entrada dessa tabela na MIB.

```
public void createRowGeneral(String tabela, Variable[] values)
{
    boolean erro = true;

    if(tabela.equals("motorista"))
    {
        MOTable<MotoristaEntryRow,MOColumn,MOTableModel<MotoristaEntryRow>> motoristaEntry = mib.getMotoristaEntry();
        MOTableModel model = (MOTableModel)motoristaEntry.getModel();

        OID index = new OID("");
        // adiciona o último índice.
        index.append(model.getRowCount() + 1);

        values[0] = new Integer32(model.getRowCount() + 1);

        MotoristaEntryRow newRow = motoristaEntry.createRow(index, values);
        motoristaEntry.addRow(newRow);

        System.out.println("Numero de linhas: "+model.getRowCount());
        erro = false;
    }

    if(tabela.equals("autocarro"))
    {
        MOTable<AutocarroEntryRow,MOColumn,MOTableModel<AutocarroEntryRow>> autocarroEntry = mib.getAutocarroEntry();
        MOTableModel model = (MOTableModel)autocarroEntry.getModel();

        OID index = new OID("");
        // adiciona o último índice.
        index.append(model.getRowCount() + 1);

        values[0] = new Integer32(model.getRowCount() + 1);

        AutocarroEntryRow newRow = autocarroEntry.createRow(index, values);
        autocarroEntry.addRow(newRow);

        System.out.println("Numero de linhas: "+model.getRowCount());
        erro = false;
    }

    if(tabela.equals("itinerario"))
    {
```

Como podemos observar, tudo o resto segue o mesmo processo tomado na criação da MIB. No final desta operação, é enviada uma resposta para o manager, resposta essa que vai depender do sucesso ou insucesso da inserção (ver código acima do *while true*..).

```
this.createRowGeneral("autocarro", values);
resp = "Autocarro adicionado com sucesso!";
```

Para isso existe a variável ***resp*** (ver código do "processamensagem") que no caso de uma inserção com sucesso indica que a inserção nessa tabela foi feita com sucesso.

```
String resp="Ocorreu um erro";
```

No caso de insucesso, a variável assume o valor acima apresentado. Desta forma, o agente após processar a mensagem sabe se a inserção foi ou não feita com sucesso, e envia essa resposta para o manager (ver código *while true*..).

```
resp = agent.processaMensagem(recorte);

//envia resposta
DatagramPacket response = new DatagramPacket(resp.getBytes(), resp.length(), request.getAddress(), request.getPort());
socket.send(response);
```

Assim, quando o manager recebe a resposta, sabe qual o tipo de notificação que pode apresentar na tela do administrador que está a tentar inserir novos dados na MIB.

Esta pode não ser a melhor alternativa para contornar o problema das permissões, mas funciona e a outra também não é melhor. Criar 500 linhas vazias não seria uma boa solução, pois se lidarmos com uma tabela em que os dados estão continuamente a aumentar, estamos a impor um limite na tabela que terá que ser mais tarde alterado, o que não é uma boa opção.

A alternativa escolhida tem como desvantagem o factor segurança. Abrir uma via de comunicação entre o agente e o manager pode levantar problemas de segurança, uma vez que não há um controlo sob quem é o manager que está a enviar pedidos de criação de novas entradas na tabela. No entanto, isto é apenas um estudo e esse factor não é relevante, daí a *community string* ser igual para todos os managers. Caso quiséssemos aumentar o nível de segurança, teríamos que definir *community strings* para cada manager, ou então utilizar simplesmente a *SNMPv3*. Conclui-se que a melhor solução seria mesmo a criação de novos dados na MIB a partir do comando *SET*, mas tal não foi possível no contexto deste estudo.

Iniciação do Agente

Neste estudo, a base de dados SQL do servidor onde os dados são guardados corre na mesma rede e máquina que o agente. Tendo isso em conta, colocou-se o agente a correr no endereço ***127.0.0.1*** (***localhost***), porta ***2001***, pelo que os managers que quiserem comunicar com o agente terão que saber o seu endereço IPV4 e porta onde corre, ou os pacotes enviados não obterá qualquer resposta. Caso o servidor SQL não corresse na mesma máquina ou mesma rede, apenas teria que se alterar este campo.

```java
// OS PRINTS SERVEM PARA FAZER DEBUG
public static void main(String[] args) throws IOException, InterruptedException
{
        String hostAddress = getHostIP();

        //Agent agent = new Agent("udp:127.0.0.1/161");
        Agente agent = new Agente("127.0.0.1/2001");
        System.out.println("Vou iniciar o agente");

        agent.start(); // ja regista os Managed Objects todos.
        System.out.println("Agente iniciado com sucesso.");

        agent.loadDataFromBD();

        // cria um socket UDP
        //DatagramSocket socket = new DatagramSocket(6789);
        DatagramSocket socket = new DatagramSocket(6788);
        byte[] buffer = new byte[1000];
        System.out.println("Servidor está a aguardar pedidos...");

        while(true)
        {
            agent.readAndSaveRows();

            // cria datagrama para receber request do cliente
            DatagramPacket request = new DatagramPacket(buffer, buffer.length);
            socket.receive(request);
            System.out.println("*** Request recebido de: " + request.getAddress());

            byte[] msg = request.getData();

            String s = new String(request.getData());

            String recorte[] = s.split("\\|");

            //resp = processaMensagem(recorte[]);
            System.out.println("Recorte[0]: " +recorte[0]);
            System.out.println("Recorte[1]: " +recorte[1]);

            String resp = "Erro!";

            resp = agent.processaMensagem(recorte);

            //envia resposta
            DatagramPacket response = new DatagramPacket(resp.getBytes(), resp.length(), request.getAddress(), request.getPort());
            socket.send(response);
        }
}
```

Após tomar todos os passos necessários à criação do agente e iniciação do protocolo de transporte (ver código abaixo) no endereço e porta escolhidos, faz-se o carregamento dos dados da base de dados SQL para a MIB.

```java
public Agente(String address) throws IOException
{
    super(
            new File("conf.agent"), new File("bootCounter.agent"),
            new CommandProcessor(new OctetString(MPv3.createLocalEngineID()))
        );

    this.address = address;
}

@Override
protected void initTransportMappings() throws IOException
{
    transportMappings = new TransportMapping[1];
    Address addr = GenericAddress.parse(address);
    TransportMapping tm = TransportMappings.getInstance().createTransportMapping(addr);
    transportMappings[0] = tm;
}
```

Após iniciar o agente, é criado o *socket* UDP para abrir a via de comunicação entre o agente e o manager que corre no sistema de administração, via essa onde o agente fica a aguardar pedidos. De seguida, corre até ser interrompido. Durante a sua execução vai estar à escuta de possíveis pedidos *GET* e *SET*. Nos casos em que a informação não foi alterada por si (e.g. alteração de informação existente) o agente faz cópia dos dados da MIB para a base de dados SQL, de forma a manter sempre uma cópia de segurança atualizada. Caso os dados sejam alterados por si próprio (e.g. criação de uma nova entrada numa tabela da MIB), faz também essa cópia.

É importante o agente sempre estar atento a possíveis alterações dos dados, pois este é o único que tem acesso à base de dados SQL. Por definição, os managers nunca têm acesso directo aos dados, têm sempre que passar pelo agente.

Tanto o utilizador, como o motorista da componente veicular e o administrador do sistema central do operador têm um programa gestor/cliente SNMP. Sempre que necessitam de visualizar ou alterar dados que estão presentes na MIB, este gestor tem primeiro que fazer o respetivo pedido ao agente da central. Tal como foi visto nos parágrafos anteriores, numa situação de alteração de dados o manager envia o *SET request* para o agente, e em caso de sucesso este envia uma resposta com os dados de volta para o manager, que seguidamente os apresenta na aplicação UI. Após alterar os dados na MIB - acção consequente do pedido *SET* – o agente atualiza a base de dados SQL.

Quanto à duração do programa agente, este corre até ser interrompido. Num contexto real, a não ser que ocorram falhas inesperadas o agente nunca é interrompido. Desta forma não tem que fazer repetidamente o carregamento dos dados e não há riscos de perda de dados.

Após observar o sucesso da construção manual de linhas na tabela posteriormente o carregamento de dados a partir da base de dados SQL, o passo seguinte foi construir um manager para testar remotamente as funcionalidades do agente.

B. Gestor presente no Sistema de Administração

Após a construção do agente da central veio a do gestor (ou manager), uma vez que só com este programa é que é possível testar a eficiência remota das funcionalidades do agente. Neste caso estamos a falar do manager que corre no sistema de administração, cujas funções são adicionar, remover e alterar os dados presentes na Central-MIB.

Para que a comunicação entre o manager e o agente seja possível, é necessário iniciar a via de ligação, tal como aconteceu no agente. No entanto, neste caso o endereço IP passado em parâmetro deve ser o IP do agente da central e não o *localhost*.

```
// Constructor
public Manager(String address) throws IOException
{
    super();
    this.address = address;
    this.SNMPversion = 2;
    this.requestID=1;

    try{
        start();
    }
    catch (IOException e) {
        throw new RuntimeException(e);
    }
}

/**
 * Start the SNMP session
 * If you forget the listen() method you will not get
 * any answers because the communication is asynchronous
 * and the listen() method listens for answers.
 *
 * @throws IOException
 */
public void start() throws IOException
{
    TransportMapping transport = new DefaultUdpTransportMapping();
    snmp = new Snmp(transport);

    transport.listen();

}
```

A criação da via de comunicação tem que ocorrer nas duas partes visto tratar-se de uma via de comunicação bidirecional. Ao correr o método "***start()***", se não se colocar o método "***transport.listen()***", o manager poderá até enviar pedidos para o agente mas nunca irá obter resposta, pois não fica à escuta. Novamente, a versão utilizada nos pedidos enviados para o agente é a *SNMPv2*.

Pedidos GET

Após a criação do manager, segue-se a definição dos pedidos a serem enviados para o agente. Sabemos que o manager que corre no sistema de administração deve poder adicionar, remover e alterar a informação da *Central-MIB*, logo as operações necessárias são o *SET*, *GET* e *WALK* (e.g. quando queremos percorrer uma tabela inteira). Antes de mais, o manager necessita de saber a quem é que vai enviar pacotes, isto é, quem é o *target*.

```java
/**
 * This method returns a Target, which contains information about
 *  where the data should be fetched and how.
 */
public Target getTarget()
{
    Address targetAddress = GenericAddress.parse(address);
    CommunityTarget target = new CommunityTarget();

    target.setCommunity(new OctetString("public"));
    target.setAddress(targetAddress);

    target.setRetries(2);
    target.setTimeout(1500);
    target.setVersion(SnmpConstants.version2c);

    return target;

}
```

A *community string* utilizada é a *string* ***public***, tal como já foi supracitado na secção do agente. Tendo o target, pode-se então definir o método *GET*, que é o mais importante pois permite obter informação da MIB.

```java
/**
 * Method which takes a single OID and returns the response from the
 * agent as a String
 *
 * @param oid
 * @return
 * @throws IOException
 */
public String getAsString(OID oid) throws IOException
{
    ResponseEvent event = get(new OID[] { oid });
    return event.getResponse().get(0).getVariable().toString();
}

/**
 * This method is more generic and is capable of handling multiple OIDs
 * In a real application with lots of agents you would probably implement
 * this asynchronously with a ResponseListener instead to prevent your
 * threadpool from being exhausted.
 *
 * @param oids
 * @return
 * @throws IOException
 */
public ResponseEvent get(OID oids[]) throws IOException
{
    PDU pdu = new PDU();
    for(OID object : oids)
    {
        pdu.add(new VariableBinding(object));
    }

    pdu.setType(PDU.GET);

    ResponseEvent event = snmp.send(pdu, getTarget(), null);

    if(event != null)
    {
        return event;
    }
    throw new RuntimeException("GET timed out..");
}
```

Ao enviar um pedido *GET* para o agente, o manager recebe a resposta sob a forma de ***ResponseEvent***, pelo que temos que interpretar a resposta, retirar o valor da variável nela contida e convertê-lo para ***string***. Ao criar o pacote PDU a ser enviado, temos que especificar qual é o tipo do pacote (*SET* ou *GET*) para que o agente do outro lado possa rapidamente interpretá-lo. Quanto ao *WALK*, como vamos observar mais à frente não existe um tipo de pacote *WALK*, isto porque na realidade um pedido *WALK* são vários *GET's* sucessivos de um dado OID.

Caso quiséssemos fazer um *GET* dos dados de uma tabela inteira, também seria possível.

```
/**
 * To test that we receive expected table data.
 *
 * @param oids
 * @return
 */
public List<List<String>> getTableAsStrings(OID[] oids)
{
    TableUtils tUtils = new TableUtils(snmp, new DefaultPDUFactory());

    @SuppressWarnings("unchecked")
    List<TableEvent> events = tUtils.getTable(getTarget(), oids, null, null);

    List<List<String>> list = new ArrayList<List<String>>();

    for(TableEvent event : events)
    {
        if(event.isError())
        {
            throw new RuntimeException(event.getErrorMessage());
        }
        List<String> strList = new ArrayList<String>();
        list.add(strList);

        for(VariableBinding vb : event.getColumns())
        {
            strList.add(vb.getVariable().toString());
        }
    }

    return list;
}
```

Pedidos WALK

Quando necessitamos de pedir vários dados de uma só vez e não queremos fazer *GET's* sucessivos, faz-se um *WALK*. O comando *snmpwalk* essencialmente faz de forma automática *getnext's* sucessivos, para nos poupar ao trabalho de os fazermos manualmente. Isto é, vai percorrer automaticamente a OID *Tree* e retornar todos os valores ao seu alcance, começando no OID que lhe passamos em parâmetro. O *WALK* apenas pára quando retorna resultados que já não se encontram dentro do alcance do OID que inicialmente foi especificado. Por exemplo, se fizéssemos um *snmpwalk* ***experimental.999.1.2.1.1,*** os resultados iriam ser ***experimental.999.1.2.1.1.1*** e ***experimental.999.1.2.1.1.2,***

com os valores *"Miguel"* e *"João"*, respectivamente. Se fizermos um *WALK* a um OID que esteja perto da raiz da árvore, vai-nos dar todos os valores que se encontram nos níveis inferiores a esse elemento.

```
public String snmpwalk(String object) throws IOException
{
    String results = "";
    String targetAddr = "127.0.0.1";
    String oidStr = object;
    String commStr = "public";
    int snmpVersion = SnmpConstants.version2c;
    String portNum = "2001";


    Snmp snmp = null;
    TransportMapping transport = null;
    try {
        //Address targetAddress = GenericAddress.parse("udp:" + targetAddr + "/" + portNum);
        Address targetAddress = GenericAddress.parse(this.address);
        transport = new DefaultUdpTransportMapping();
        snmp = new Snmp(transport);
        transport.listen();

        // setting up target
        CommunityTarget target = new CommunityTarget();
        target.setCommunity(new OctetString(commStr));
        target.setAddress(targetAddress);
        target.setRetries(3);
        target.setTimeout(1000 * 3);
        target.setVersion(snmpVersion);

        OID oid = new OID(oidStr);

        TreeUtils treeUtils = new TreeUtils(snmp, new DefaultPDUFactory());
        List<TreeEvent> events = treeUtils.getSubtree(target, oid);
        if (events == null || events.size() == 0) {
            return null;
        }

        // Get snmpwalk result.
        for (TreeEvent event : events) {
            if (event.isError()) {
                System.out.println("error :OID has an error: {}" + event.getErrorMessage() +"");
            } else {
                VariableBinding[] varBindings = event.getVariableBindings();
                if (varBindings == null || varBindings.length == 0) {
                    //System.out.println("VarBinding: No result returned.");
                }
                for (VariableBinding varBinding : varBindings) {
                    varBinding.toValueString();
                    results += (varBinding.getOid().toString() + "=" + varBinding.getVariable().toString()) + "\n";
                }
            }
        }
```

Neste tipo de sistema o comando *WALK* não é muito útil, mas para aqueles que gerem os dados e dominam SNMP pode tornar-se útil pois permite visualizar fisicamente o estado da MIB *Tree*.

Pedidos SET - Alteração de Dados

Devido ao problema das permissões que já foi explicado anteriormente, os pedidos *SET* servem apenas para alteração de dados, e não criação. O método de alteração não é muito complexo pois apenas envia o pedido com os dados. Quanto à resposta, cabe ao sistema UI interpretar a resposta proveniente do agente e apresenta-la ao utilizador, tendo em conta o sucesso ou insucesso da operação de alteração.

Visto que existem dois tipos de dados na MIB – *strings* e inteiros – terão que ser feitos dois métodos distintos, pois o tipo de dados que é passado em parâmetro é diferente. Tudo isto é feito automaticamente no sistema, ou seja, quando o administrador está a alterar o *nome* ou o *nif* de um motorista não se apercebe que estão a ser chamados métodos diferentes.

```
public ResponseEvent set(OID oid,String val) throws IOException
{
    PDU pdu = new PDU();
    VariableBinding varBind = new VariableBinding(oid,new OctetString(val));
    pdu.add(varBind);
    pdu.setType(PDU.SET);
    pdu.setRequestID(new Integer32(1));
    Target target=getTargetForWrite();

    ResponseEvent event = snmp.set(pdu, target);
    if(event != null)
    {
      System.out.println("\nResponse:\nGot Snmp Set Response from Agent");
      System.out.println("Snmp Set Request = " + event.getRequest().getVariableBindings());
      PDU responsePDU = event.getResponse();
      System.out.println("\nresponsePDU = "+responsePDU);
      if (responsePDU != null)
      {
        int errorStatus = responsePDU.getErrorStatus();
        int errorIndex = responsePDU.getErrorIndex();
        String errorStatusText = responsePDU.getErrorStatusText();
        System.out.println("\nresponsePDU = "+responsePDU);
        if (errorStatus == PDU.noError)
        {
          System.out.println("Snmp Set Response = " + responsePDU.getVariableBindings());
        }
        else
        {
          System.out.println("errorStatus = "+responsePDU);
          System.out.println("Error: Request Failed");
          System.out.println("Error Status = " + errorStatus);
          System.out.println("Error Index = " + errorIndex);
          System.out.println("Error Status Text = " + errorStatusText);
        }
       }

      return event;
    }
    throw new RuntimeException("GET timed out");
}
```

Como podemos observar pelo código acima, é atribuído o tipo *SET* ao pacote PDU que está a ser criado, e a variável que estamos a fazer *bind (*i.e. anexar ao pacote PDU*)* é uma *string*. Vejamos agora o caso de alteração de um campo inteiro.

```
public ResponseEvent setInt(OID oid,int val) throws IOException
{
    PDU pdu = new PDU();
    VariableBinding varBind = new VariableBinding(oid,new Integer32(val));
    pdu.add(varBind);
    pdu.setType(PDU.SET);
    pdu.setRequestID(new Integer32(1));
    Target target=getTargetForWrite();

    ResponseEvent event = snmp.set(pdu, target);
    if(event != null)
    {
      System.out.println("\nResponse:\nGot Snmp Set Response from Agent");
      System.out.println("Snmp Set Request = " + event.getRequest().getVariableBindings());
      PDU responsePDU = event.getResponse();
      System.out.println("\nresponsePDU = "+responsePDU);
      if (responsePDU != null)
      {
        int errorStatus = responsePDU.getErrorStatus();
        int errorIndex = responsePDU.getErrorIndex();
        String errorStatusText = responsePDU.getErrorStatusText();
        System.out.println("\nresponsePDU = "+responsePDU);
        if (errorStatus == PDU.noError)
        {
          System.out.println("Snmp Set Response = " + responsePDU.getVariableBindings());
        }
        else
        {
          System.out.println("errorStatus = "+responsePDU);
          System.out.println("Error: Request Failed");
          System.out.println("Error Status = " + errorStatus);
          System.out.println("Error Index = " + errorIndex);
          System.out.println("Error Status Text = " + errorStatusText);
        }
       }

      return event;
    }
    throw new RuntimeException("GET timed out");
}
```

A única diferença é o tipo de dados que vai no pacote, neste caso um *Integer32*. Uma alternativa seria fazer apenas um método e verificar se a *string* passada em parâmetro continha inteiros, e nesse caso seria um set a um campo inteiro. No entanto, neste contexto não funciona pois há campos da MIB que necessitam obrigatoriamente de serem *strings* e também contêm inteiros (e.g. "*POI1*", que identifica o ponto de interesse com *id*='1' na MIB).

Pedidos SET - Remoção de dados

Este tipo de pedido não difere muito do pedido *SET* anterior. Isto porque na realidade não existe remoção de dados, apenas a alteração da coluna estado nas tabelas da MIB. Se existisse a remoção de linhas numa tabela da MIB, poderia gerar inconsistência de dados. Se, por exemplo, fizéssemos um comando *walk* a uma tabela após a remoção de uma linha, o *walk* iria parar antes da linha removida, mesmo que essa não fosse a última. Isto é, o *walk* vai fazendo *getnext's* sucessivos até encontrar um valor "*noSuchInstance*" (i.e. não existe instância de linha), e quando encontra esse valor pára. Caso uma

linha seja removida, o SNMP irá dar essa linha como "*noSuchInstance*", o que irá arruinar as pesquisas nas tabelas.

Esta foi a razão pela qual se optou por alterar apenas o campo "*estado*" em vez de remover uma linha. Em cada tabela da MIB a coluna estado assume os valores "*activo*" ou "*inactivo*". Assim, quando o carregamento de itinerários for feito, apenas se carregam os itinerários válidos, isto é, aqueles que se encontram activos.

```
this.manager.set(oid, "inactivo");
```

Quando o administrador pretende remover um itinerário, ele escolhe na UI o itinerário que pretende remover e o sistema automaticamente faz a alteração dessa linha a partir do manager que nele corre. O OID desse itinerário é passado em parâmetro e como é uma remoção o valor passado em parâmetro é "*inactivo*". Após esta "*remoção*" o sistema passa a saber que esse itinerário não é mais dado como válido.

Pedidos de Criação de Dados

Mais uma vez, como não é possível criar dados a partir do comando *SET* criou-se a via de comunicação alternativa entre agente e manager. Como vimos no agente, o pacote UDP que lhe é enviado pelo manager é processado e a sua mensagem analisada, de forma a saber qual a tabela onde deve criar dados, assim como o número e tipo de campos esperados.

Portanto, imaginemos que no sistema de administração está a ser inserido um novo motorista e todos os dados são preenchidos no respetivo formulário. De seguida, o sistema recolhe esses dados e constrói uma mensagem com os dados e o nome da tabela. Posteriormente constrói um datagrama UDP e anexa a mensagem ao pacote, enviando-a de seguida para o agente e esperando a resposta. Por fim, recebe a resposta do agente e apresenta uma mensagem ao administrador. Note-se que independentemente do sucesso ou insucesso do pedido, o manager recebe sempre um resposta, a não ser que tenha enviado o pedido para um endereço ou porta errada.

```java
public void enviaDatagramaParaAgente(String nome, int nif) throws SocketException, IOException
{
    // cria um socket UDP
    DatagramSocket socket = new DatagramSocket();
    System.out.println("* Socket criado na porta: " + socket.getLocalPort());

    String msg = "motorista|"+nome+"|"+nif+"|";

    byte[] m = msg.getBytes(); // transforma arg em bytes
    InetAddress serv = InetAddress.getByName(address);
    //int porta = 6789;
    int porta = 6788;
    DatagramPacket request = new DatagramPacket(m, msg.length(), serv, porta);

    // envia datagrama contendo a mensagem m
    socket.send(request);


    byte[] buffer = new byte[1000];
    DatagramPacket resposta = new DatagramPacket(buffer, buffer.length);
    socket.setSoTimeout(10000); // timeout em ms

    // recebe resposta do servidor - fica em wait ateh chegada
    socket.receive(resposta);
    System.out.println("* Resposta do servidor:" + new String(resposta.getData()));

    socket.close();
}
```

No caso do pedido de criação de um novo motorista, apenas é necessário o *nome* e o *nif*, sendo que o agente trata de atribuir automaticamente o *id*.

```java
public void enviaDatagramaCreateAutocarro(String matricula, int capacidade) throws SocketException, IOException
{
    // cria um socket UDP
    DatagramSocket socket = new DatagramSocket();
    System.out.println("* Socket criado na porta: " + socket.getLocalPort());

    String msg = "autocarro|"+matricula+"|"+capacidade+"|";

    byte[] m = msg.getBytes(); // transforma arg em bytes
    InetAddress serv = InetAddress.getByName(address);
    int porta = 6789;
    DatagramPacket request = new DatagramPacket(m, msg.length(), serv, porta);

    // envia datagrama contendo a mensagem m
    socket.send(request);


    byte[] buffer = new byte[1000];
    DatagramPacket resposta = new DatagramPacket(buffer, buffer.length);
    socket.setSoTimeout(10000); // timeout em ms

    // recebe resposta do servidor - fica em wait ateh chegada
    socket.receive(resposta);
    System.out.println("* Resposta do servidor:" + new String(resposta.getData()));

    socket.close();
}
```

No caso do pedido de criação de um novo autocarro, o administrador necessita somente de indicar a *matrícula* e a *capacidade* do autocarro que deseja criar. Como já foi visto anteriormente, o agente trata dos restantes valores, assumindo valores por definição (e.g. última localização, lugares livres, etc.).

```java
public void enviaDatagramaCreateItinerario(String id, int shortID, String nome, String listaParagens,
                                           String horaInicioServico, String listaDePontosDeInteresse,
                                           String listaDePontosDoPercurso) throws SocketException, IOException
{
    // cria um socket UDP
    DatagramSocket socket = new DatagramSocket();
    System.out.println("* Socket criado na porta: " + socket.getLocalPort());

    String msg = "itinerario|"+id+"|"+shortID+"|"+nome+"|"+listaParagens+"|"+horaInicioServico+"|"+listaDePontosDeInteresse+"|"+listaDePontosDoPercurso+"|";
    System.out.println("msg: " +msg);

    byte[] m = msg.getBytes(); // transforma arg em bytes
    InetAddress serv = InetAddress.getByName(address);
    int porta = 6789;
    DatagramPacket request = new DatagramPacket(m, msg.length(), serv, porta);

    // envia datagrama contendo a mensagem m
    socket.send(request);


    byte[] buffer = new byte[1000];
    DatagramPacket resposta = new DatagramPacket(buffer, buffer.length);
    socket.setSoTimeout(10000); // timeout em ms

    // recebe resposta do servidor - fica em wait ateh chegada
    socket.receive(resposta);
    System.out.println("* Resposta do servidor:" + new String(resposta.getData()));

    socket.close();
}
```

No caso de criação de um novo itinerário, o datagrama enviado ao agente contem alguma informação sobre o itinerário, sendo que a restante informação é completada pelo próprio agente aquando a inserção na MIB.

Quanto aos restantes casos de inserção, todos seguem a mesma norma. A informação essencial é requisitada ao administrador e enviada para o agente num datagrama, e o agente trata de criar novas linhas nas tabelas da MIB e povoá-las com os dados.

C. Agente e Managers presentes no sistema do autocarro

Após criar os dois primeiros elementos mais importantes do sistema, seguiu-se a criação das partes que integram o sistema do autocarro, sistema esse que é utilizado pelos motoristas: dois managers e um agente. Relembrando o modelo concetual do componente do sistema presente no veículo, existe um gestor SNMP especial que funciona como um *relay* para troca bidirecional de informação entre este componente e o sistema central, um gestor SNMP local que gere os dados na MIB local (*Autocarro-MIB*) implementada pela instrumentação dum agente SNMP local de modo a atualizar os dados do veículo.

À medida que o autocarro vai percorrendo um itinerário, os dois managers vão automaticamente recolhendo informação. Além de recolher informação, o manager local vai também buscar a informação necessária ao funcionamento do sistema (e.g. lista de paragens do itinerário, informação da próxima paragem, etc).

O manager servidor central tem como principal função o download dos dados do itinerário que o motorista pretende percorrer. Depois disso, apenas vai enviar para o agente central as atualizações dos dados do autocarro (e.g. número de lugares livres, posição atual, paragem mais próxima, etc.). Desta forma, quando o passageiro pretender saber onde se encontra um dado autocarro, essa informação é requisitada ao agente da central que guarda toda a informação, em vez de tentar comunicar com o autocarro em questão. Aquando do download de todos os dados necessários, esses dados são guardados na *Autocarro-MIB*, para que não seja necessário estar sempre em contacto com a central.

Todos os dados em tempo real são guardados localmente, para que não haja risco de perda de dados. Quando existe alguma alteração que o sistema considera importante enviar para a central, essa alteração é enviada pelo manager servidor central caso exista ligação internet. Se não existir ligação os dados estão guardados localmente portanto não há problema, tenta-se mais tarde. A *Central-MIB* e a *Autocarro-MIB* possuem uma etiqueta temporal, portanto é possível saber quando é que foi a última vez que comunicaram e se a informação se encontra atualizada.

Os dois managers têm praticamente os mesmos métodos, pois utilizam os mesmos dados. A única coisa que os distingue é o agente para onde eles enviam os *requests*. Além disso, a partir de um certo ponto o manager servidor central apenas envia dados, enquanto que o manager local está constantemente a enviar e a pedir dados ao agente.

Agente

O agente é praticamente igual ao da central (permissões, configurações, etc.), à excepção dos seus métodos.

```
public Agente(String address) throws IOException
{
    super(
            new File("conf.agent"), new File("bootCounter.agent"),
            new CommandProcessor(new OctetString(MPv3.createLocalEngineID()))
         );

    this.address = address;
}
```

Tal como o agente central, também necessita de criar entradas de tabelas, para quando o download da informação é feito.

```
public void createItinerarioRow(Variable[] values)
{
    MOTable<ItinerarioEntryRow,MOColumn,MOTableModel<ItinerarioEntryRow>> itinerarioEntry = mib.getItinerarioEntry();
    MOTableModel model = (MOTableModel)itinerarioEntry.getModel();

    int newID = model.getRowCount() +1;

    OID index = new OID("");

    index.append(model.getRowCount() + 1);
    // adiciona o último índice.

    ItinerarioEntryRow newRow = itinerarioEntry.createRow(index, values);
    itinerarioEntry.addRow(newRow);
}
```

Visto que durante o estudo o agente encontrou-se na mesma máquina que o manager local e o manager servidor central, o endereço onde este correu foi o *localhost*. A porta escolhida foi a *2003*, para não entrar em conflito com a porta *2001*, que é a porta onde agente da central se encontra à escuta.

```
public static void main(String[] args) throws IOException, InterruptedException
{

        //Agent agent = new Agent("udp:127.0.0.1/161");
        Agente agent = new Agente("127.0.0.1/2003");
        System.out.println("Vou iniciar o agente");

        agent.start(); // ja regista os Managed Objects todos.
        System.out.println("Agente iniciado com sucesso.");

        while(true) {}
}
```

Manager

Quanto ao manager local, a configuração e a iniciação é igual à do manager que está no sistema de administração. Além disso, os métodos *GET*, *SET* e *WALK* também são iguais, pois só o endereço é que muda.

```
// Constructor
public ManagerLocal(String address) throws IOException
{
    super();
    this.address = address;
    this.SNMPversion = 2;
    this.requestID=1;

    try{
        start();
    }
    catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Quantos ao métodos *GET*, estes são utilizados de acordo com as necessidades do motorista à medida que vai percorrendo um itinerário. Seguem-se alguns exemplos de pedidos de dados estáticos ao agente local.

```
public String getNomeDaParagem(int idParagem) throws IOException
{
    String nome = "";
    OID oid = new OID("1.3.6.1.3.9999.1.4.1.2."+idParagem);
    nome = this.getAsString(oid);

    return nome;
}

public String getNomeDoPOI(int idPOI) throws IOException
{
    String nome = "";
    OID oid = new OID("1.3.6.1.3.9999.1.5.1.2."+idPOI);
    nome = this.getAsString(oid);

    return nome;
}


public String getLatitudeDaParagem(int idParagem) throws IOException
{
    OID oidLatitude = new OID("1.3.6.1.3.9999.1.4.1.3."+idParagem);
    String latitude = this.getAsString(oidLatitude);
    return latitude;
}
```

Pode ser necessário ir buscar dados mais extensos, como listas de pontos.

```
public String getListaDePontosDeInteresse(int itinerario) throws IOException
{
    String lista = "";
    OID oid;
    OID res = new OID("");

    for(int i=1; i<500; i++)
    {
        oid = new OID("1.3.6.1.3.9999.1.2.1.3."+i);
        if(Integer.parseInt(getAsString(oid)) == itinerario)
        {
            res = new OID("1.3.6.1.3.9999.1.2.1.8."+i);
            break;
        }
    }

    lista = this.getAsString(res);
    return lista;
}
```

Além de dados estáticos, pode ser necessário pedir ao agente informação acerca de dados em tempo real.

```java
public String getLatitudeActual()
{
    String latitude = "erro";
    try
    {
        latitude = this.getAsString(new OID("1.3.6.1.3.9999.1.1.1.4.1"));
    }
    catch (IOException ex)
    {
        Logger.getLogger(ManagerLocal.class.getName()).log(Level.SEVERE, null, ex);
    }
    return latitude;
}

public String getLongitudeActual()
{
    String longitude = "erro";
    try
    {
        longitude = this.getAsString(new OID("1.3.6.1.3.9999.1.1.1.5.1"));
    }
    catch (IOException ex)
    {
        Logger.getLogger(ManagerLocal.class.getName()).log(Level.SEVERE, null, ex);
    }
    return longitude;
}

public int getVelocidadeActual()
{
    int v = 0;
    try
    {
        v = Integer.parseInt(this.getAsString(new OID("1.3.6.1.3.9999.1.1.1.6.1")));
    }
    catch (IOException ex)
    {
        Logger.getLogger(ManagerLocal.class.getName()).log(Level.SEVERE, null, ex);
    }
    return v;
}
```

Outros dados como a paragem mais próxima, ponto atual, entre outros, podem também ser pedidos durante a execução do sistema.

Quanto aos pedidos *SET*, apenas são de alteração de dados não-estáticos. É importante manter sempre a cópia desses dados localmente para o caso de falha ou até mesmo inexistência de ligação entre o autocarro e a central.

```
public void actualizaEstadoAutocarro(String estado) throws IOException
{
  OID oid = new OID("1.3.6.1.3.9999.1.1.1.7.1");
  // o estado do autocarro e actualizado para "1 - Em servico".
  this.set(oid, estado);
}

public void actualizaItinerarioAutocarro(int itinerario) throws IOException
{
  OID oid = new OID("1.3.6.1.3.9999.1.1.1.9.1");
  this.setInt(oid, itinerario);
}


public void actualizaSentido(int sentido) throws IOException
{
  OID oid = new OID("1.3.6.1.3.9999.1.1.1.3.1");
  this.setInt(oid, sentido);
}
```

Seguem-se mais alguns exemplos.

```
public void actualizaPosicaoActualDoAutocarro(String latitude, String longitude) throws IOException
{
    OID oidLatitude = new OID("1.3.6.1.3.9999.1.1.1.4.1");
    OID oidLongitude = new OID("1.3.6.1.3.9999.1.1.1.5.1");

    this.set(oidLatitude, latitude);
    this.set(oidLongitude, longitude);
}

public void actualizaVelocidadeAutocarro(int velocidade) throws IOException
{
    OID oid = new OID("1.3.6.1.3.9999.1.1.1.6.1");
    this.setInt(oid, velocidade);
}
```

Como o motorista vê muita desta informação em tempo real, é também importante guardar as saídas e entradas de passageiros no autocarro.

```
public void registaEntradaNoAutocarro(int passageiros) throws IOException
{
    OID oidLugaresLivres = new OID("1.3.6.1.3.9999.1.1.1.13.1");
    OID oidLugaresOcupados = new OID("1.3.6.1.3.9999.1.1.1.14.1");
    OID oidCapacidade = new OID("1.3.6.1.3.9999.1.1.1.12.1");

    int lugaresLivres = Integer.parseInt(this.getAsString(oidLugaresLivres));
    int lugaresOcupados = Integer.parseInt(this.getAsString(oidLugaresOcupados));
    int capacidade = Integer.parseInt(this.getAsString(oidCapacidade));

    lugaresLivres = lugaresLivres - passageiros;
    lugaresOcupados = lugaresOcupados + passageiros;

    if(lugaresOcupados > capacidade)
    {
        lugaresOcupados = capacidade;
    }

    if(lugaresLivres < 0)
    {
        lugaresLivres = 0;
    }

    this.setInt(oidLugaresLivres, lugaresLivres);
    this.setInt(oidLugaresOcupados, lugaresOcupados);
}
```

Convém tratar dos erros e excepções, visto que o motorista não pode registar a saída de um passageiro se o autocarro está vazio, ou registar a entrada de um passageiro se o autocarro está cheio. Estes dados são importantes para que o passageiro possa consultar o número de lugares livres do autocarro que pretende apanhar.

D. Gestor do Servidor Central

O manager servidor central, que é o manager que comunica com o agente presente na central, tem a mesma configuração que o manager local, à excepção do endereço para onde envia *requests*.

```
// Constructor
public ManagerServidorCentral(String address) throws IOException
{
    super();
    this.address = address;
    this.SNMPversion = 2;
    this.requestID=1;

    try{
        start();
    }
    catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Tal como o seu irmão, os métodos *GET*, *SET* e *WALK* são iguais ao manager presente no sistema de administração. No início do sistema, se a *Autocarro-MIB* estiver desatualizada, o manager servidor central tem que ser fazer o download de todos os dados da central.

```java
public HashMap loadItinerariosFromCentral() throws IOException
{
    HashMap<Integer, Variable[]> map = new HashMap<>();

    boolean continua = true;

    String BDid;
    String id;
    String shortID;
    String nome;
    String versao;
    String listaDeParagens;
    String horaInicioServico;
    String listaPOIS;
    String listaPontos;
    int proximoID;

    for(int i=1; i<100 && continua; i++)
    {
        BDid                = "1.3.6.1.3.999.1.3.1.1."+i;
        id                  = "1.3.6.1.3.999.1.3.1.2."+i;
        shortID             = "1.3.6.1.3.999.1.3.1.3."+i;
        nome                = "1.3.6.1.3.999.1.3.1.4."+i;
        versao              = "1.3.6.1.3.999.1.3.1.5."+i;
        listaDeParagens     = "1.3.6.1.3.999.1.3.1.6."+i;
        horaInicioServico   = "1.3.6.1.3.999.1.3.1.7."+i;
        listaPOIS           = "1.3.6.1.3.999.1.3.1.9."+i;
        listaPontos         = "1.3.6.1.3.999.1.3.1.10."+i;

        int Id = Integer.parseInt(this.getAsString(new OID(BDid)));
        String itinerarioID = this.getAsString(new OID(id));
        int shid = Integer.parseInt(this.getAsString(new OID(shortID)));
        String name = this.getAsString(new OID(nome));
        int version = Integer.parseInt(this.getAsString(new OID(versao)));
        String paragens = this.getAsString(new OID(listaDeParagens));
        String hora = this.getAsString(new OID(horaInicioServico));
        String pois = this.getAsString(new OID(listaPOIS));
        String pontos = this.getAsString(new OID(listaPontos));

        Variable[] values = new Variable[] {new Integer32(Id),              // id
                                            new OctetString(itinerarioID),  // itinerarioID
                                            new Integer32(shid),            // short id
                                            new OctetString(name),          // nome
                                            new Integer32(version),         // versao
                                            new OctetString(paragens),      // lista de paragens
                                            new OctetString(hora),          // hora de inicio de servico
                                            new OctetString(pois),          // lista de pontos de interesse
                                            new OctetString(pontos)         // lista de pontos do percurso
                                           };

        proximoID = i+1;

        if(this.getAsString(new OID("1.3.6.1.3.999.1.3.1.1."+proximoID)).equals("noSuchInstance"))
        {
            continua = false;
        }

        map.put(i, values);
    }

    return map;

}
```

Este é apenas um exemplo, pois tal como faz o download dos itinerários, também faz o de todos os outros dados que devem ser guardados na *Autocarro-MIB*, como paragens, pontos de interesse, etc.

O manager servidor central realiza pedidos *SET* sempre que há informação em tempo real a ser atualizada, isto é, quando o manager local também faz atualizações na *Autocarro-MIB*. Os pedidos só são enviados se existir ligação à central (i.e. ligação internet).

```java
public boolean existeConexaoComACentral() throws UnknownHostException, IOException
{
    String ipAddress = "";

    if(this.address.equals("127.0.0.1/2001"))
    {
        ipAddress = "localhost";
    }
    else
    {
        ipAddress = this.address;
    }

    boolean ret = false;
    InetAddress inet = InetAddress.getByName(ipAddress);

    System.out.println("Sending Ping Request to " + this.address);

    if(inet.isReachable(5000))
    {
        ret=true;
        System.out.println("Host is reachable");
    }
    else
    {
        System.out.println("Host is not reachable");
    }

    return ret;
}
```

Sem ligação não vale a pena estar a enviar pois sabemos que os pacotes não vão sequer partir da origem. A atualização dos dados pode ser feita de duas formas. Por vezes, é apenas necessário atualizar por exemplo a paragem atual onde o autocarro se encontra.

```java
public void actualizaPosicaoActualDoAutocarroNaCentral(int idAutocarro, String latitude, String longitude) throws IOException
{
    OID oidLatitude = new OID("1.3.6.1.3.999.1.1.1.4."+idAutocarro);
    OID oidLongitude = new OID("1.3.6.1.3.999.1.1.1.5."+idAutocarro);

    this.set(oidLatitude, latitude);
    this.set(oidLongitude, longitude);
}

public void actualizaParagemActualNaCentral(int idAutocarro, int paragem) throws IOException
{
    OID oidParagemActual = new OID("1.3.6.1.3.999.1.1.1.10."+idAutocarro);
    this.setInt(oidParagemActual, paragem);
}
```

Noutras alturas surge a necessidade de alterar vários dados ao mesmo tempo, pelo que é mais vantajoso atualizar tudo, ao invés de estar fazer a atualização uma a uma, o que pode comprometer a performance do sistema.

```
public void actualizaDadosDeViagemNaCentral(int idAutocarro, int sentido, String latitude, String longitude,
                                            int velocidade, String estado, int motoristaActual, int itinerarioActual,
                                                int paragemActual, int capacidade, int lugaresLivres, int lugaresOcupados,
                                                    int poiMaisProximo, String pontoMaisProximo)
                                                        throws IOException
{
  System.out.println("------- actualizaDadosDeViagemNaCentral ---------");

  OID oidSentido          =  new OID("1.3.6.1.3.999.1.1.1.3."+idAutocarro);
  OID oidLatitude         =  new OID("1.3.6.1.3.999.1.1.1.4."+idAutocarro);
  OID oidLongitude        =  new OID("1.3.6.1.3.999.1.1.1.5."+idAutocarro);
  OID oidVelocidade       =  new OID("1.3.6.1.3.999.1.1.1.6."+idAutocarro);
  OID oidEstado           =  new OID("1.3.6.1.3.999.1.1.1.7."+idAutocarro);
  OID oidMotoristaActual  =  new OID("1.3.6.1.3.999.1.1.1.8."+idAutocarro);
  OID oidItinerarioActual =  new OID("1.3.6.1.3.999.1.1.1.9."+idAutocarro);
  OID oidParagemActual    =  new OID("1.3.6.1.3.999.1.1.1.10."+idAutocarro);
  OID oidCapacidade       =  new OID("1.3.6.1.3.999.1.1.1.12."+idAutocarro);
  OID oidLugaresLivres    =  new OID("1.3.6.1.3.999.1.1.1.13."+idAutocarro);
  OID oidLugaresOcupados  =  new OID("1.3.6.1.3.999.1.1.1.14."+idAutocarro);
  OID oidPoiMaisProximo   =  new OID("1.3.6.1.3.999.1.1.1.16."+idAutocarro);
  OID oidPontoMaisProximo =  new OID("1.3.6.1.3.999.1.1.1.17."+idAutocarro);

  this.setInt(oidSentido, sentido);
  this.set(oidLatitude, latitude);
  this.set(oidLongitude, longitude);
  this.setInt(oidVelocidade, velocidade);
  this.set(oidEstado, estado);
  this.setInt(oidMotoristaActual, motoristaActual);
  this.setInt(oidItinerarioActual, itinerarioActual);
  this.setInt(oidParagemActual, paragemActual);
  this.setInt(oidCapacidade, capacidade);
  this.setInt(oidLugaresLivres, lugaresLivres);
  this.setInt(oidLugaresOcupados, lugaresOcupados);
  this.setInt(oidPoiMaisProximo, poiMaisProximo);
  this.set(oidPontoMaisProximo, pontoMaisProximo);

  System.out.println("-------------------------");
}
```

Embora nem sempre seja utilizado, caso alguma coisa corra mal e o agente central não se tenha apercebido que os dados da *Central-MIB* foram alterados pelo manager presente no autocarro, há sempre a possibilidade de avisá-lo remotamente.

```
public void enviaAvisoDeActualizacaoParaAgente() throws SocketException, IOException
{
    // cria um socket UDP
    DatagramSocket socket = new DatagramSocket();
    System.out.println("* Socket criado na porta: " + socket.getLocalPort());

    String msg = "actualizaAutocarro|";

    byte[] m = msg.getBytes(); // transforma arg em bytes
    InetAddress serv = InetAddress.getByName("localhost");
    int porta = 6789;
    DatagramPacket request = new DatagramPacket(m, msg.length(), serv, porta);

    // envia datagrama contendo a mensagem m
    socket.send(request);


    byte[] buffer = new byte[1000];
    DatagramPacket resposta = new DatagramPacket(buffer, buffer.length);
    socket.setSoTimeout(10000); // timeout em ms

    // recebe resposta do servidor - fica em wait ateh chegada
    socket.receive(resposta);
    System.out.println("* Resposta do servidor após pedido de actualização:" + new String(resposta.getData()));

    socket.close();
}
```

Como vimos na configuração do agente central, os pacotes recebidos são processados. Assim, o agente sabe que foram feitas atualizações e grava-as na base de dados SQL do servidor.

```
// mensagem enviada quando há informação a ser actualizada em tempo real quanto ao autocarro
if(recorte[0].equals("actualizaAutocarro"))
{
    this.readAndSaveAutocarroRows();
    resp = "Informação dos autocarros foi actualizada com sucesso!";
}
```

De forma a saber a última vez que o sistema central comunicou com o sistema do autocarro, é guardada uma etiqueta temporal para cada autocarro. Só assim é possível saber se os dados do autocarro que se encontram na central são dados em tempo real ou não.

```
public void actualizaUltimaLeituraNaCentral(int autocarro) throws IOException
{
  String data = new Date().toString();
  OID oid = new OID("1.3.6.1.3.999.1.1.1.11."+autocarro);
  this.set(oid, data);

}
```

Uma alternativa possível à criação de dois agentes distintos seria criar apenas um, e definir dois *targets* com endereços diferentes: um do agente central e outro do agente local. No entanto, fazer isto implicaria juntar os métodos todos numa só classe, gerando desta forma o *design pattern* denomindado de *blob* - técnica de programação não muito recomendada - o que tornaria o código muito confuso.

E. Gestor no Sistema do Passageiro

Por fim, foi definido o manager que corre no sistema do passageiro. Apesar de este manager apenas ter sido definido após ter o sistema de administração e o sistema do autocarro bem definidos, é apresentado nesta secção uma vez que diz respeito à instrumentação SNMP. Como já foi mencionado, no sistema do passageiro apenas existe um manager que comunica com o agente da central quando necessita de dados, estáticos ou dinâmicos. A configuração do manager segue o mesmo padrão dos managers definidos anteriores.

```
public String getAsString(String cmd) throws Exception
{

    String resultado = "";

    // Create TransportMapping and Listen
    Log.d(TAG, "-TransporteUDPMapping criado.");

    Log.d(TAG, "Create Target Address object");
    // Create Target Address object
    CommunityTarget comtarget = new CommunityTarget();
    comtarget.setCommunity(new OctetString(community));
    comtarget.setVersion(SNMP_VERSION);

    Log.d(TAG, "-address: " + ipAddress + "/" + port);

    comtarget.setAddress(new UdpAddress(ipAddress + "/" + port));
    comtarget.setRetries(2);
    comtarget.setTimeout(1500);

    Log.d(TAG, "Prepare PDU");
    // create the PDU
    PDU pdu = new PDU();
    pdu.add(new VariableBinding(new OID(cmd)));
    pdu.setType(PDU.GET);


    Log.d(TAG, "Sending Request to Agent...");

    // send the PDU
    ResponseEvent response = snmp.send(pdu, comtarget, transport);

    // Process Agent Response
    if (response != null) {
        // extract the response PDU (could be null if timed out)
        PDU responsePDU = response.getResponse();
        // extract the address used by the agent to send the response:
        Address peerAddress = response.getPeerAddress();
        Log.d(TAG, "peerAddress " + peerAddress);
        if (responsePDU != null) {
            int errorStatus = responsePDU.getErrorStatus();
            int errorIndex = responsePDU.getErrorIndex();
            String errorStatusText = responsePDU.getErrorStatusText();

            if (errorStatus == PDU.noError) {
                Log.d(TAG, "Snmp Get Response = " + responsePDU.getVariableBindings());
                Log.d(TAG, "Snmp GET = " + responsePDU.get(0).getVariable().toString());

                resultado = responsePDU.get(0).getVariable().toString();

            } else {
                Log.d(TAG, "Error: Request Failed");
                Log.d(TAG, "Error Status = " + errorStatus);
                Log.d(TAG, "Error Index = " + errorIndex);
                Log.d(TAG, "Error Status Text = " + errorStatusText);


            }
        } else {
            Log.d(TAG, "Error: Response PDU is null");
        }
    } else {
        Log.d(TAG, "Error: Agent Timeout... \n");
    }

    return resultado;
}
```

No arranque do sistema, caso não existam dados no sistema ou os dados existentes estejam desatualizados, o manager envia uma série de pacotes *GET* para o agente central, de modo a gravar localmente toda a informação estática necessária ao funcionamento do sistema.

```
public TreeMap loadItinerariosFromCentral() throws Exception
{
    TreeMap<Integer, Variable[]> map = new TreeMap<>();

    boolean continua = true;

    String BDid;
    String id;
    String shortID;
    String nome;
    String versao;
    String listaDeParagens;
    String horaInicioServico;
    String listaPOIS;
    String listaPontos;
    int proximoID;
    String horaFimServico;

    for(int i=1; i<100 && continua; i++)
    {
        BDid                = "1.3.6.1.3.999.1.3.1.1."+i;
        id                  = "1.3.6.1.3.999.1.3.1.2."+i;
        shortID             = "1.3.6.1.3.999.1.3.1.3."+i;
        nome                = "1.3.6.1.3.999.1.3.1.4."+i;
        versao              = "1.3.6.1.3.999.1.3.1.5."+i;
        listaDeParagens     = "1.3.6.1.3.999.1.3.1.6."+i;
        horaInicioServico   = "1.3.6.1.3.999.1.3.1.7."+i;
        listaPOIS           = "1.3.6.1.3.999.1.3.1.9."+i;
        listaPontos         = "1.3.6.1.3.999.1.3.1.10."+i;
        horaFimServico      = "1.3.6.1.3.999.1.3.1.8."+i;


        int Id              = Integer.parseInt(this.getAsString(BDid));
        String itinerarioID = this.getAsString(id);
        int shid            = Integer.parseInt(this.getAsString(shortID));
        String name         = this.getAsString(nome);
        String hora         = this.getAsString(horaInicioServico);
        int version         = Integer.parseInt(this.getAsString(versao));
        String paragens     = this.getAsString(listaDeParagens);
        String pois         = this.getAsString(listaPOIS);
        String pontos       = this.getAsString(listaPontos);
        String horaFim      = this.getAsString(horaFimServico);

        Variable[] values = new Variable[] {
                new Integer32(Id),              // id
                new OctetString(itinerarioID),  // itinerarioID
                new Integer32(shid),            // short id
                new OctetString(name),          // nome
                new OctetString(hora),          // hora de inicio de servico
                new Integer32(version),         // versao
                new OctetString(paragens),      // lista de paragens
                new OctetString(pois),          // lista de pontos de interesse
                new OctetString(pontos),        // lista de pontos do percurso
                new OctetString(horaFim)        // hora de fim de serviço
        };

        proximoID = i+1;

        if(this.getAsString("1.3.6.1.3.999.1.3.1.1."+proximoID).equals("noSuchInstance"))
        {
            continua = false;
        }

        map.put(i, values);
    }

    return map;
}
```

Como podemos observar pelo código acima, no caso dos itinerários enviam-se os *GET requests* do agente para pedir a informação dos itinerários que se considera importante guardar localmente. Todas as restantes tabelas seguem exactamente a mesma estrutura.

O manager do sistema do passageiro apenas actua em duas situações:

1. quando o sistema é iniciado e é necessária a instalação ou atualização dos dados.
2. quando o utilizador pretende saber onde se encontra um dado autocarro em tempo real e o manager tem que pedir essa informação ao agente central.

```
public void actualizaPosicoesDoAutocarroNaBDLocal(int i) throws Exception
{

    String id;
    String latitude;
    String longitude;
    String itinerarioActual;
    String paragemActual;
    String lugaresLivres;
    String lugaresOcupados;
    String poiMaisProximo;
    String pontoActualMaisProximo;

    id                          = "1.3.6.1.3.999.1.1.1.1."+i;
    latitude                    = "1.3.6.1.3.999.1.1.1.4."+i;
    longitude                   = "1.3.6.1.3.999.1.1.1.5."+i;
    itinerarioActual            = "1.3.6.1.3.999.1.1.1.9."+i;
    paragemActual               = "1.3.6.1.3.999.1.1.1.10."+i;
    lugaresLivres               = "1.3.6.1.3.999.1.1.1.13."+i;
    lugaresOcupados             = "1.3.6.1.3.999.1.1.1.14."+i;
    poiMaisProximo              = "1.3.6.1.3.999.1.1.1.16."+i;
    pontoActualMaisProximo      = "1.3.6.1.3.999.1.1.1.17."+i;

    int autocarro       = Integer.parseInt(this.getAsString(id));
    String lat          = this.getAsString(latitude);
    String lon          = this.getAsString(longitude);
    int itinerario      = Integer.parseInt(this.getAsString(itinerarioActual));
    int paragem         = Integer.parseInt(this.getAsString(paragemActual));
    int livres          = Integer.parseInt(this.getAsString(lugaresLivres));
    int ocupados        = Integer.parseInt(this.getAsString(lugaresOcupados));
    int poi             = Integer.parseInt(this.getAsString(poiMaisProximo));
    String pontoActual  = this.getAsString(pontoActualMaisProximo);

    guardaNaBDultimoItinerarioRegistado(i, lat, lon, itinerario, paragem, livres, ocupados, poi, pontoActual);
}
```

No caso de precisar da informação de um dado autocarro em tempo real, o manager envia um *request* ao agente para que lhe envie toda a informação acerca desse autocarro, caso a ligação ao servidor central seja válida. A informação recebida é posteriormente guardada na base de dados local.

REFERÊNCIAS

[1] “Google Maps”, [Online]. Available: https://www.google.pt/maps/

[2] “Moovit”, [Online]. Available: https://moovitapp.com/

[3] “Trafi”, [Online]. Available: https://www.trafi.com/

[4] “Citymapper”, [Online]. Available: https://citymapper.com/

[5] “Transit”, [Online]. Available: https://transitapp.com/

[6] PCMag, “Transit App Face-Off: Citymapper vs. Transit vs. Moovit”, [Online]. Available: https://www.pcmag.com/article/338575/transit-app-face-off-citymapper-vs-transit-vs-moovit

[7] “Sapo Transportes”, [Online]. Available: http://m.transportes.sapo.pt/

[8] “MOVE-ME”, [Online]. Available: http://www.move-me.mobi/

[9] “SMS/ Email ao Minuto”, [Online]. Available: http://www.carris.pt/pt/informacao-ao-passageiro/

[10] “TUB Mobile”, [Online]. Available: http://mobile.tub.pt/

[11] “Automatic Vehicle Location”, [Online]. Available: https://www.liveabout.com/automatic-vehicle-location-avl-2798823

[12] “Global Positioning System”, [Online]. Available: https://www8.garmin.com/aboutGPS/

[13] “SQL - Structured Query Language”, [Online]. Available: https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_71/db2/rbafzsqlcon.htm

[14] “Simple Network Management Protocol (SNMP)”, [Online]. Available: http://www.net-snmp.org/

[15] “Internet Engineering Task Force (IETF)”, [Online]. Available: https://www.ietf.org/

[16] https://en.wikipedia.org/wiki/Simple_Network_Management_Protocol#/media/File:SNMP_communication_principles_diagram.PNG

[17] “Structure of Management Information Version 2 (SMIv2)”, [Online]. Available: https://tools.ietf.org/html/rfc2578

[18] https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc783142(v=ws.10)

[19] “Abstract Syntax Notation One (ASN.1)”, [Online]. Available: https://www.itu.int/en/ITU-T/asn1/Pages/introduction.aspx

[20] https://www.networkmanagementsoftware.com/snmp-tutorial-part-2-rounding-out-the-basics/

[21] “MIB Designer”, [Online]. Available: https://agentpp.com/tools/mibdesigner.html

[22] “Crowdsourcing”, [Online]. Available: https://en.wikipedia.org/wiki/Crowdsourcing

[23] “Identificação por Radiofrequência”, [Online]. Available: https://www.ncontrol.com.pt/o-que-e-rfid.html

[24] “Radio-frequency Identification”, [Online]. Available: https://www.atlasrfidstore.com/rfid-beginners-guide/

[25] “Active RFID vs. Passive RFID: What’s the Difference?”, [Online]. Available: http://blog.atlasrfidstore.com/active-rfid-vs-passive-rfid

[26] http://newsimg.bbc.co.uk/media/images/40091000/jpg/_40091293_rfid_tag203.jpg

[27] http://www.ajautomacao.com/tags-etiquetas-de-rfid/

[28] http://www.trustagsrfids.com/Content/File_Img/S_Product/2016-08-30/20160830113922552642O.png

[29] https://sc01.alicdn.com/kf/HTB1w9o5nYsTMeJjSszhq6AGCFXaY/High-quality-rfid-nfc-dry-wet-inlay.jpg_350x350.jpg

[30] https://blog.atlasrfidstore.com/active-rfid-vs-passive-rfid (https://blog.atlasrfidstore.com/wp-content/uploads/2013/06/passive-rfid-tag.jpg

[31] https://en.wikipedia.org/wiki/Radio-frequency_identification#/media/File:TransCore_RFID_reader_and_antenna.jpg

[32] https://en.wikipedia.org/wiki/Radio-frequency_identification#/media/File:Microchip_rfid_rice.jpg

[33] “Dead Recknoning”, [Online]. Available: https://en.wikipedia.org/wiki/Dead_reckoning.

[34] “Global System for Mobile Communications (GSM)”, [Online]. Available: https://searchmobilecomputing.techtarget.com/definition/GSM

[35] “Differential Global Positioning System (DGPS)” [Online]. Available: https://www.esri.com/news/arcuser/0103/differential1of2.html

[36] “European Geostationary Navigation Overlay Service (EGNOS)”, [Online]. Available: https://www.gsa.europa.eu/egnos/what-egnos
[37] “Wi-fi” [Online]. Available: https://www.cisco.com/c/en/us/products/wireless/what-is-wifi.html
[38] “IEEE 802.11” [Online]. Available: https://www.infowester.com/wifi.php
[39] “Terrestrial Turked Radio (TETRA)” [Online]. Available: https://www.etsi.org/technologies/tetra
[40] “Near Field Communication”, [Online]. Available: http://nearfieldcommunication.org
[41] https://pt.wikipedia.org/wiki/Near_Field_Communication#/media/File:Nfc_func.jpg
[42] http://techshakes.com/wp-content/uploads/2013/10/nfc.jpg
[43] http://1.bp.blogspot.com/-umtkghyARyM/TzycEgZSo-I/AAAAAAAAAWY/AzYQ-hvwV1w/s320/paywave2.gif
[44] "Capture more consumers with NFC Technology in your vending machines", http://www.parlevelsystems.com/2013/10/01/nfc-and-vending/
[45] “Intelligent Transportation System (ITS)”, [Online]. Available: https://www.geospatialworld.net/blogs/what-is-intelligent-transport-system-and-how-it-works/
[46] Victor Nassar e Milton Luiz Horn Vieira, “A Tecnologia no auxílio ao transporte urbano: Projecto Smart Bus NFC RFID”, 2013.
[47] “Long-Term Evolution (LTE)”, [Online]. Available: https://www.sciencedirect.com/topics/engineering/long-term-evolution
[48] “Ultra high frequency (UHF)“ [Online]. Available: https://www.britannica.com/technology/UHF
[49] “Very high frequency (VHF) “ [Online]. Available: https://www.britannica.com/technology/VHF
[50] “Dedicated Short Range Communications (DSRC)”, [Online]. Available: https://www.fluidmesh.com/dsrc-dedicated-short-range-communications/
[51] “Wimax” [Online]. Available: http://wimaxforum.org/
[52] “Mesh Networks” [Online]. Available: https://www.sciencedirect.com/topics/computer-science/wireless-mesh-network

[53] “Método MoSCoW” [Online]. Available: https://www.toolshero.com/project-management/moscow-method/
[54] Suzanne Robertson e James Robertson, ” Mastering the Requirements Process: Getting Requirements Right”, 2006.

S. Sociedade de Transportes Colectivos do Porto, "Sociedade de Transportes Colectivos do Porto, SA," [Online]. Available: http://www.stcp.pt.

P. Kelly, "Quality bus transit systems," em International Conference on Public Transport Electronic Systems, 1996.

Android Central, "Best Transit App", [Online]. Available: http://www.androidcentral.com/best-transit-app.

Luís Eduardo Cachulo, "SITREPA – Sistema de Informação em Tempo-Real em Paragens de Autocarro", 2012.

Quora, "How do bus tracking apps work? Where do they get the GPS data from?", [Online]. Available: https://www.quora.com/How-do-bus-tracking-apps-work-Where-do-they-get-the-GPS-data-from.

Diário de Notícias, "Transit App Moovit. A aplicação para quem anda de transportes públicos", [Online]. Available: http://www.dn.pt/sociedade/interior/transit-app-moovit-a-aplicacao-para-quem-anda-de-transportes-publicos-5102688.html.

Which? "Top five best transport apps", [Online]. Available: https://blogs.which.co.uk/technology/app-review/top-five-best-transport-apps/.

Alexandra António, "Gestão de Operações nos Transportes Públicos Rodoviários de Passageiros", 2008.

The Telegraph, “Urban navigation app Citymapper raises $40m”, [Online]. Available: http://www.telegraph.co.uk/technology/news/12111388/Urban-navigation-app-Citymapper-raises-40m.html.

Ricardo Gomes Rodrigues, “Gestão em Tempo Real de Serviços de Transporte de Passageiros”, 2008.

Miguel Gomes Zenha, “SNMP VS NETCONF”, em Gestão de Redes, Mestrado Integrado em Engenharia Informática, Universidade do Minho, 2015.

O Público, “Informática a bordo para gestão à distância”, [Online]. Available: https://www.publico.pt/noticias/jornal/informatica-a-bordo-para-gestao-a-distancia-172016

Domingos Fernando Peixoto da Silva, “Sistemas de Informação Geográfica para Transportes: uma aplicação aos transportes urbanos de Guimarães.”, 2006.

Áquila Neves Chaves, Fernando Constantino Barceleini e Lucas Weng, “Sistema de Monitorização de Tráfego por meio de RFID”, 2008.

YourStory, “Lithuania-based TRAFI aims to make public transport and travel smarter in India”, [Online]. Available: https://yourstory.com/2016/05/trafi/.

“The Tech Behind Traffic Apps: How (Well) Do They Work?”, [Online]. Available: http://www.npr.org/sections/alltechconsidered/2015/05/18/407658702/the-tech-behind-traffic-apps-how-well-do-they-work.

Techilicious, “Best Navigation Apps”, [Online]. Available: http://www.techlicious.com/tip/best-navigation-apps/.

Quora, “What knowledge is needed to create a bus tracking system using GPS on a board of a bus integrated with Google Maps for public use?”, [Online]. Available: https://www.quora.com/What-knowledge-is-needed-to-create-a-bus-tracking-system-using-GPS-on-a-board-of-a-bus-integrated-with-Google-Maps-for-public-use.

Youtube, “Controlo de Transportes Urbanos com Tags RFID”, [Online]. Available: https://www.youtube.com/watch?v=SCAxOr-dhbw.

Youtube, “Montreal - controle de transporte urbano com RFID”, [Online]. Available: https://www.youtube.com/watch?v=-QO04olXZvI.

Shifter, “Se andas de transportes públicos em Lisboa, tens que instalar o Citymapper no teu telemóvel”, [Online]. Available: http://shifter.pt/2015/07/se-andas-de-transportes-publicos-em-lisboa-instala-o-citymapper-no-teu-telemovel/.