Edna Eunice Mascarenhas Moreira

Licenciatura em Ciências de Engenharia
Electrotécnica e de Computadores

Desenvolvimento e Controlo Remoto de Sinais de Trânsito Interligados Utilizando IOPT-Tools

Dissertação para obtenção do Grau de Mestre em
Engenharia Electrotécnica e de Computadores

Orientador: Doutor Filipe de Carvalho Moutinho,

Professor Auxiliar, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Co-orientador: Doutor Rogério Alexandre Botelho Campos Rebelo,

Investigador, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Júri

Presidente: Doutor Paulo da Costa Luís da Fonseca Pinto
Arguente: Doutor João Paulo Mestre Pinheiro Ramos e Barros
Vogal: Doutor Filipe de Carvalho Moutinho

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Junho 2018

Desenvolvimento e Controlo Remoto de Sinais de Trânsito Interligados Utilizando IOPT-Tools

Copyright © Edna Eunice Mascarenhas Moreira, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa.

A Faculdade de Ciências e Tecnologia e a Universidade Nova de Lisboa têm o direito, perpétuo e sem limites geográficos, de arquivar e publicar esta dissertação através de exemplares impressos reproduzidos em papel ou de forma digital, ou por qualquer outro meio conhecido ou que venha a ser inventado, e de a divulgar através de repositórios científicos e de admitir a sua cópia e distribuição com objetivos educacionais ou de investigação, não comerciais, desde que seja dado crédito ao autor e editor.

À minha família e amigos…

Agradecimentos

Em primeiro lugar gostaria de agradecer aos meus orientadores Filipe Moutinho e Rogério Campos-Rebelo pela sua disponibilidade, paciência e conselhos. As suas sugestões foram essenciais para o desenvolvimento e escrita desta dissertação.

Um obrigado aos meus colegas e amigos da Faculdade de Ciências e Tenologia da pelo apoio que me deram ao longo destes anos, especialmente à Ana Ferreira, Bruno de Almeida, Carlos Silva, Gisela Seixas, Hussein Rassid, Miguel Vieira, Ricardo Belchior e Tiago Pereira.

Não poderia deixar de agradecer à minha família e restantes amigos pela sua paciência e suporte durante esta etapa da minha vida. O seu apoio e confiança permitiram que chegasse aqui.

Resumo

As condições de circulação das estradas encontram-se em constante alteração. Isto deve-se a vários fatores como condições climatéricas, ocupação ou ocorrência de acidentes. Apesar disso, grande parte dos sinais de trânsito possui informação estática. O desenvolvimento de sinalização não estática, como é o caso de sinais de mensagens variáveis, pode melhorar a qualidade da informação disponibilizada aos condutores, o que pode resultar numa diminuição do número de acidentes rodoviários, uma vez que os condutores passam a ter acesso a informação sobre as vias públicas em tempo real.

Neste trabalho é proposta uma arquitetura para sinais de trânsito que torna possível a partilha de informação entre eles, a comunicação destes com veículos, o seu controlo, monitorização e ainda a configuração da informação que disponibilizam. A partilha de informação é feita através da implementação de serviços acessíveis através de redes Wi-Fi difundidas pelos sinais de trânsito. É também proposta neste trabalho a utilização das ferramentas IOPT para suportar a monitorização, o controlo e o desenvolvimento de controladores para estes sinais.

Palavras-chave: *Sinais de trânsito, comunicação entre infraestruturas, sistemas de controlo e monitorização remotos, redes de Petri, ferramentas IOPT.*

Abstract

Road traffic conditions are constantly changing. This is due to several factors such as weather conditions, occupancy or occurrence of accidents. Despite this, most road signs have static information. The development of non-static road signs, such as variable message signs, can improve the quality of information provided to drivers, which can result in a reduction in the number of road accidents, since drivers are given real-time information on road condition.

In this work is proposed an architecture for road signs that allows their interaction, their interaction with vehicles, control and monitoring systems, and the configuration of the information they provide. Information sharing is done through the implementation of services accessible through Wi-Fi networks broadcasted by the road signs. It is also proposed the use of IOPT-Tools to support the development, monitoring and control of road sign controllers.

Keywords: *Road signs, Infrastructure-to-Infrastructure communication, remote control and monitoring systems, Petri nets, IOPT-Tools.*

Conteúdo

Agradecimentos v

Resumo vii

Abstract ix

Conteúdo xi

Lista de Figuras xv

Lista de Tabelas xvii

1. Introdução 1
   - 1.1. Contextualização 1
   - 1.2. Motivação 2
   - 1.3. Objectivos 3
   - 1.4. Estrutura do Documento 4
2. Trabalhos e Tecnologias Relacionados 7
   - 2.1. Trabalhos Relacionados 8
     - 2.1.1. SAFESPOT 8
     - 2.1.2. EAR-IT 10
     - 2.1.3. CVIS 11
     - 2.1.4. COOPERS 12
     - 2.1.5. CAPTIV 13

2.1.6. Outros Projetos ... 14

2.1.7. Análise dos Projetos e das Tecnologias Utilizadas ... 15

2.2. Sinalização de Mensagem Variável ... 16

2.3. Redes de Petri e Ferramentas de Automatização de Projecto ... 18

2.3.1. Redes de Petri IOPT ... 19

2.3.2. IOPT-Tools ... 20

3. Arquitectura Proposta ... 23

3.1. Descrição Geral da Arquitetura ... 23

3.2. Comunicação entre Sinais de Trânsito – Módulos SP e SC ... 27

3.3. Comunicação entre Sinais de Trânsito e Veículos – Módulo VP ... 28

3.4. Monitorização e Controlo dos Sinais de Trânsito – Módulos SDC e RCMSP ... 29

4. Desenvolvimento e Validação ... 31

4.1. Materiais ... 31

4.1.1. Raspberry Pi 3 ... 32

4.1.2. Adaptadores de Wi-Fi ... 32

4.2. Descrição ... 33

4.2.1. Desenvolvimento do sistema de comunicação ... 34

4.2.2. Funcionamento do Servidor TCP/IP ... 37

4.2.3. Funcionamento do Cliente TCP/IP ... 38

4.2.4. Estrutura das Mensagens ... 38

4.2.5. Desenvolvimento do sistema de monitorização e controlo ... 39

4.3. Testes e Resultados ... 45

5. Conclusões e Trabalho Futuro ... 49

Referências ... 51

Anexos ... 55

A. Sistema de Comunicação do Sinal de Trânsito - Código 55

B. Controlador do Sinal de Trânsito – Código.. 74

C. Controlador do Sinal de Trânsito – Rede de Petri 79

Servidor para Veículos.. 79

Servidor para Sinais de Trânsito... 80

Cliente para Sinais de Trânsito 81

Lista de Figuras

Figura 1- Principais causas de morte em 2015 (adaptado de [5]) 2

Figura 2 - Interação entre dois sinais de trânsito e entre um sinal e um veículo 4

Figura 3 - Diferentes tipos de painéis de mensagem variável, adaptado de [30] 17

Figura 4 - Exemplo de painéis luminosos de mensagem única [30] 17

Figura 5 - Sinais luminosos de afetação de vias [30] 17

Figura 6 - Demonstração do disparo de uma transição 19

Figura 7 - Página principal das ferramentas IOPT 20

Figura 8 - Editor de redes IOPT 21

Figura 9 - Simulador de redes IOPT 22

Figura 10 - Interacção entre os sinais de trânsito, veículos e sistemas de controlo e monitorização remotos 24

Figura 11 - Módulos da arquitetura proposta para os sinais de trânsito 26

Figura 12 – Topologias de rede utilizadas no projeto 27

Figura 13 - Esquema representativo do Raspberry Pi 3 Model B 32

Figura 14 - Esquema representativo de um adaptador USB de Wi-Fi 33

Figura 15 - Módulos criados para implementar o sistema de comunicação 34

Figura 16 - Estrutura de uma entrada no histórico do servidor 35

Figura 17 - Redes de Petri utilizadas na monitorização e controlo dos módulos de comunicação 40

Figura 18 - Rede de Petri utilizada para controlar o funcionamento do sinal de trânsito 42

Figura 19 - Ficheiros criados pelo gerador de código C 42

Figura 20- (a) Informação fornecida pelo sinal de trânsito; (b) Registo das últimas 10 ligações feitas aos servidores 46

Figura 21 - Monitorização remota do módulo SC 46

Figura 22 - (a) Resposta do sinal em manutenção; (b) Resposta do sinal em funcionamento normal 47

Figura 23 – Modelo do servidor para comunicação entre sinais de trânsito e veículos 79

Figura 24 - Modelo do servidor para comunicação entre sinais de trânsito 80

Figura 25 - Modelo do cliente utilizado para comunicação entre sinais de trânsito . 81

Lista de Tabelas

Tabela 1 - Estrutura da informação enviada pelos sinais de trânsito 39

Tabela 2 - Eventos associados aos módulos de comunicação do sinal de trânsito 41

Tabela 3 - Mensagens recebidas pelo controlador e eventos associados..................... 43

Tabela 4 - Eventos de saída do controlador e mensagens associadas.......................... 44

Introdução

1.1. Contextualização

De acordo com a OMS [1] (Organização Mundial de Saúde), em 2015 [2] (Figura 1), os acidentes rodoviários encontravam-se entre as 10 principais causas de morte no mundo e eram a principal causa de morte de indivíduos entre os 15 e 29 anos [3], [4], tendo sido responsáveis pela morte de cerca de 1.3 milhões de pessoas em 2015 [2]. Estes acidentes são causados por vários motivos, entre os quais se encontram: o excesso de velocidade, excesso de álcool no sangue e a distração dos condutores. Estima-se que, caso não sejam tomadas medidas para melhorar a segurança nas estradas, os acidentes rodoviários irão ocupar o 7º lugar na lista de causas de morte em 2030.

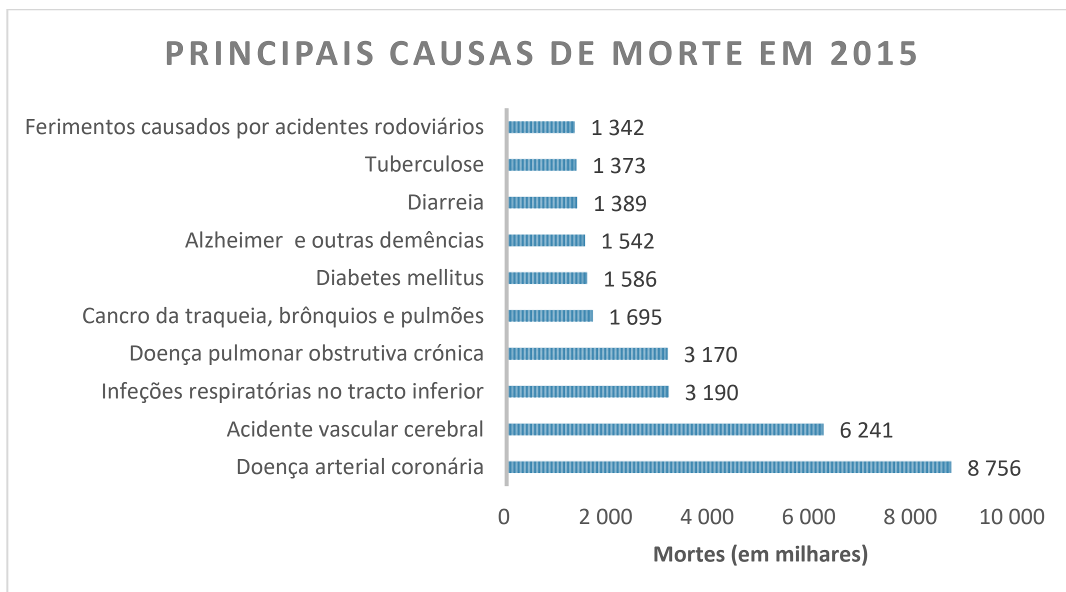


Figura 1- Principais causas de morte em 2015 (adaptado de [5])

Nos últimos anos, várias medidas têm sido tomadas de modo a melhorar a segurança dos vários utilizadores das estradas. Muitas dessas medidas enquadram-se no panorama dos Sistemas Inteligentes de Transporte (ITS, na sigla inglesa), cujo principal foco é o desenvolvimento de aplicações que permitam o aumento da comodidade e da segurança dos utilizadores de veículos [6]. Grande parte das aplicações que existem atualmente baseiam-se no reconhecimento de sinais de trânsito e obstáculos e/ou na comunicação entre veículos e outros elementos presentes nas estradas (ciclistas, peões, outros veículos e infraestruturas). Outra vertente de aplicações de ITS incide sobre o desenvolvimento das infraestruturas, de modo a tornar possível a comunicação entre estas e veículos.

1.2. Motivação

A existência de vários projetos [6] que envolvem a comunicação entre os veículos e as infraestruturas presentes nas estradas indica que estas poderão ter no futuro um papel essencial no desenvolvimento de medidas de prevenção de acidentes rodoviários, através da difusão de informação. Para tal, as infraestruturas deverão possuir uma arquitetura focada na recolha, processamento e transmissão de informação por parte dos

sinais de trânsito e, deste modo, fornecer dados aos veículos sobre o estado da estrada (ocupação, obras em curso, acidentes, …), regras de trânsito (limites de velocidade), estado de semáforos, entre outros.

Para permitir estas funcionalidades será necessária a existência de um sistema que permita a propagação de informação ao longo das vias públicas. A informação partilhada por este sistema, quando combinada com informação recebida por outros sistemas presentes no veículo, poderá ser utilizada para garantir que existe redundância de informação ou rejeitar informação errada. Deste modo o condutor/veículo poderá ter sempre acesso a informação fidedigna para o auxiliar a tomar decisões adequadas na estrada.

1.3. Objectivos

Com este trabalho pretende-se propor uma arquitetura para sinais de trânsito que possibilite o armazenamento e partilha de informação entre estes e disponibilizá-la aos veículos, tal como no exemplo ilustrado na Figura 2. Para tal, será necessário identificar e definir que informação deve ser partilhada pelos sinais de trânsito e o modo como esta é partilhada tanto com os veículos como entre diferentes sinais de trânsito. Deve ser feita a definição do formato das mensagens a enviar e a escolha de uma tecnologia e protocolos de comunicação que se revelem adequados para o sistema. A informação recebida deverá ser processada e armazenada de forma adequada e deve ser possível definir que dados serão enviados e com que frequência. Deverá ser possível configurar, para cada sinal de trânsito, de que forma os dados serão processados, armazenados e enviados.

Um outro objetivo deste trabalho consiste em verificar se ferramentas de automatização de projeto, nomeadamente as ferramentas IOPT, são adequadas para desenvolver, monitorizar e controlar o funcionamento dos sinais de trânsito. Para isso, o comportamento dos sinais será modelado através de redes de Petri IOPT, que depois

de simuladas e verificadas, serão usadas para gerar automaticamente o código do controlador e dos nós de comunicação que suportam a comunicação com o sistema de monitorização e controlo remoto.

Para validar as propostas serão desenvolvidos protótipos a instalar em sinais de trânsito e será testada a comunicação entre estes e o envio de informação tanto do próprio sinal como de outros sinais de trânsito aos veículos circundantes.

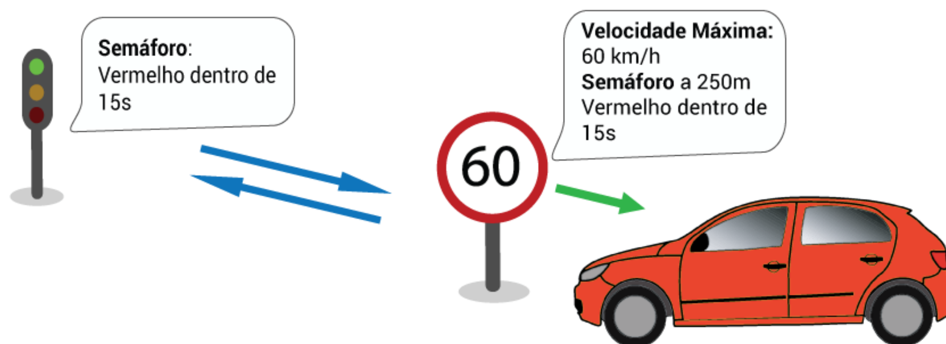


Figura 2 - Interação entre dois sinais de trânsito e entre um sinal e um veículo

1.4. Estrutura do Documento

Este documento encontra-se dividido em cinco capítulos. No primeiro capítulo é introduzido o tema desta dissertação, mencionando o contexto em que este se integra, a motivação e os objetivos desta.

No segundo capítulo são apresentados vários projetos relacionados com o desenvolvimento das infraestruturas rodoviárias, a sua comunicação com veículos e aplicações de segurança relacionadas. Seguidamente, é feita uma análise comparativa aos projetos e às tecnologias que estes utilizam. Por fim, são introduzidos os painéis de mensagem variável e métodos para definir a informação que estes disponibilizam. Neste capítulo é ainda apresentado um conjunto de ferramentas de automatização de projeto (*IOPT-Tools*) e o formalismo de modelação redes de Petri IOPT.

No terceiro capítulo é descrita a arquitetura proposta para os sinais de trânsito, introduzindo os vários módulos concebidos para comunicação, controlo e monitorização. É ainda proposto o uso de ferramentas de automatização de projeto, nomeadamente as IOPT-Tools, para suportar o desenvolvimento destes sinais.

No quarto capítulo são descritos os protótipos, feitos para validar a arquitetura proposta no capítulo anterior, e a sua abordagem de desenvolvimento. São ainda apresentados os testes realizados.

No quinto capítulo são apresentadas as conclusões ao trabalho realizado, e sugeridos alguns melhoramentos que poderiam ser realizados no futuro.

Nas últimas páginas deste documento é possível encontrar os anexos que contêm detalhes acerca da implementação da solução desenvolvida.

Trabalhos e Tecnologias Relacionados

Devido ao crescimento da população, à sua concentração em determinados locais e à urbanização [7], têm aparecido novos desafios na área de ITS [6] relativos à otimização do trânsito nas vias públicas e ao aumento da segurança destas. Com estes desafios em mente, têm surgido vários projetos onde são estudados vários sistemas cooperativos que, através de diferentes modos de comunicação envolvendo veículos e infraestruturas, permitem que condutores consigam obter informação relevante durante o seu percurso.

Neste capítulo são apresentados alguns dos projetos desenvolvidos nos últimos anos para promover a segurança nas estradas e as tecnologias utilizadas durante o desenvolvimento destes. Em primeiro lugar será introduzido o SAFESPOT, um projeto que envolve diferentes tipos de comunicação, dando enfâse aos subprojectos relacionados com o desenvolvimento das infraestruturas. Em seguida é apresentado o EAR-IT que apresenta um modo inovador de obter informação a partir de sensores acústicos instalados nas infraestruturas. Os projetos CVIS, COOPERS e CAPTIV são projetos com aplicações semelhantes às que se pretendem desenvolver, com diferenças que também serão apresentadas neste capítulo. Serão ainda apresentados outros projetos interessantes do ponto de vista do desenvolvimento de infraestruturas e, finalmente, uma análise aos projetos e tecnologias utilizadas.

Para além destes projetos, serão introduzidos os painéis de sinalização variável. Este tipo de sinalização permite que a sua informação seja alterada de acordo com as

condições da estrada em que se encontram. Neste capítulo são distinguidas as diferentes variantes deste tipo de sinalização dinâmica, tal como plataformas utilizadas para os controlar. Por fim, são introduzidas as ferramentas de automatização de projeto IOPT, o modo como podem ser utilizadas no desenvolvimento, controlo e monitorização de sistemas, e apresentadas as redes de Petri e as redes de Petri IOPT (*Input-Output Place-Transition*).

2.1. Trabalhos Relacionados

2.1.1. SAFESPOT

O SAFESPOT (*Cooperative systems for Road Safety*) [8] é um projeto que surgiu na União Europeia durante o FP6 (*6th Framework Program*). Este projeto surgiu com o objectivo de encontrar soluções cooperativas que permitissem combinar a informação obtida através de veículos e infraestruturas inteligentes e assim, promover a segurança dos condutores [8], [9].

O SAFESPOT foi dividido em oito subprojectos: SAFEPROBE (*In-vehicle sensing and platform*), INFRASENS (*Infrastructure sensing and platform*), SINTECH (*Innovative technologies*), SCOVA (*Cooperative systems applications vehicle based*), COSSIB (*Cooperative systems applications infrastructure based*), BLADE (*Business models, legal aspects, and deployment*), SCORE (SAFESPOT *core architecture*) e HOLA (*Horizontal activities*). Destes subprojectos, três eram focados na integração das infraestruturas numa arquitetura cooperativa: INFRASENS, SINTECH e COSSIB.

2.1.1.1. INFRASENS

No INFRASENS foi criada uma plataforma que permitisse que a informação proveniente dos sensores presentes nas estradas fosse integrada com a informação

disponibilizada pelos veículos – através da plataforma desenvolvida pelo subprojecto SAFEPROBE – e de maneira a obter uma solução cooperativa que promova a segurança dos condutores.

A plataforma desenvolvida era constituída por vários módulos capazes de detetar situações de risco que possam ocorrer, *software* adequado para processar os dados e gerar informação que possa ser utilizada para fornecer avisos em tempo real aos utilizadores. Estes avisos são comunicados aos condutores através de mensagens apresentadas pelas infraestruturas ou por mensagens que estas enviam para os veículos [9].

Um destes módulos é o *Data Fusion Module* [8] que trata do processamento dos dados através de dois processos: refinamento de objetos e refinamento de situações. Estes processos permitem aumentar a precisão dos dados obtidos quanto à presença de obstáculos e veículos, ou de alterações das condições das estradas (condições meteorológicas, trânsito, etc.), respetivamente. Os dados tratados pelo *Data Fusion Module* são obtidos através de vários sistemas de sensores como: *laser scanners*, câmaras, sistemas de RFID (*RadioFrequency IDentification*) e WSNs (*Wireless Sensor Networks*).

2.1.1.2. SINTECH

Este subprojecto tinha como objectivo adaptar tecnologias de comunicação e integrá-las nas arquiteturas desenvolvidas no SAFEPROBE e INFRASENS [8]. Algumas das tecnologias consideradas permitiam a localização cooperativa relativa, posicionamento baseado em GNSS (*Global Navigation Satellite System*), comunicações baseadas na localização, rede de comunicação entre veículos e infraestruturas. Esta última deveria ter em conta a necessidade de utilizar protocolos de comunicação seguros, rápidos e sem falhas para comunicação V2V (*Vehicle to Vehicle*) e V2I (*Vehicle to Infrastructure*), a necessidade de propagar a informação e da existência de situações prioritárias. Para realizar a comunicação, foi utilizada a tecnologia IEEE 802.11p [8], [10].

2.1.1.3. COSSIB

O COSSIB foi criado com o intuito de desenvolver um conjunto de aplicações que criam mensagens de aviso com base na informação obtida pelas infraestruturas e veículos e que depois utilizam as infraestruturas para as comunicar [11]. As aplicações desenvolvidas deveriam ser capazes de aumentar a margem de segurança entre peões, ciclistas e veículos, melhorar o modo como a informação relativa à segurança é apresentada aos vários veículos presentes nas estradas e a qualidade dessa informação [8].

Com os fatores anteriores em mente foram criadas aplicações que: indicam a velocidade recomendada aos condutores (*Dynamic Speed Alert*) a partir de dados como as condições meteorológicas e condições da estrada [11]; impedem que os veículos se despistem (*Road departure prevention*) – uma trajetória segura é criada a partir de informação como a geometria da estrada, velocidade e carga do veículo e sempre que o veículo saí da trajetória, uma mensagem de aviso é enviada ao condutor; informam todos os utilizadores de estradas sobre situações de risco que possam ocorrer em cruzamentos (*Intersection collision prevention*) [8]; e os veículos de emergência têm prioridade e são utilizados como infraestruturas móveis da estrada em que se encontram.

2.1.2. EAR-IT

O projeto europeu EAR-IT (*Experimenting Acoustics in Real environments using Innovative Test-beds*) [12] procura utilizar tecnologia acústica para aplicações domésticas e urbanas. A utilização de tecnologia acústica nas redes de sensores sem fios apresenta várias vantagens, visto que, ao contrário do que acontece com os sensores normalmente utilizados, os sensores acústicos não se encontram limitados por ângulos de visão, condições meteorológicas, presença de obstáculos e, quando equipados com dispositivos com capacidade de processamento, permitem a deteção de eventos específicos [13].

A deteção de eventos no EAR-IT passa por três etapas realizadas por uma APU (*Acoustic Processing Unit*). Na primeira etapa é feito o pré-processamento dos sinais (eliminação de ruído, filtragem do sinal, …) de modo a que estes cheguem à fase seguinte com a melhor qualidade possível. A etapa seguinte é formada por três fases: inicialmente, é feita a extração de características principais do sinal, ou seja, o menor conjunto de características que permitem identificar o evento; seguidamente, é feita a classificação do evento, recorrendo a algoritmos de *clustering* ou de *machine-learning*; e, por último, é feita a deteção do evento acústico através do resultado do classificador. Na última etapa é feita uma modelação estatística que permite verificar a evolução dos eventos identificados. Esta modelação é feita recorrendo a informação obtida em eventos anteriores.

Este sistema de deteção de eventos acústicos revelou ter bastantes aplicações tais como: a deteção de comandos de voz, o que permite controlar vários dispositivos; deteção da presença de pessoas em edifícios, que possibilita melhorar a eficiência energética de edifícios [14]; deteção e localização de veículos de emergência, que permite controlar sinais de trânsito de maneira a otimizar o seu percurso; e a obtenção de dados sobre a ocupação das estradas [12].

2.1.3. CVIS

Tal como o SAFESPOT, o CVIS (*Cooperative Vehicle-Infrastructure Systems*) [15] surgiu na União Europeia e teve como objectivo desenvolver e testar várias tecnologias para sistemas cooperativos que possibilitassem a comunicação entre veículos e as infraestruturas que os rodeiam [10].

Como cada veículo pode necessitar de vários endereços de IP e a mobilidade é uma característica essencial para arquitetura, a tecnologia utilizada no CVIS é baseada no IPv6. O acesso a diferentes tecnologias (3G, Wi-Fi, DSRC, entre outros) é controlado utilizando o *standard* de comunicação ISO (*International Organization for Standardization*) CALM [16].

O sistema desenvolvido é bastante flexível visto que não existe qualquer tipo de hierarquia definida: qualquer elemento do sistema pode comunicar com outro, podendo ser um provedor de serviços num instante e um consumidor noutro. Como os veículos e infraestrutura não têm papéis diferentes na arquitetura, as mesmas plataformas podem ser instaladas nos diferentes elementos do sistema. Isto torna o sistema escalável e flexível, e, deste modo, facilmente adaptável a diferentes aplicações.

Para comprovar a utilidade e eficácia da arquitetura na melhoria das condições de segurança e comodidade dos utilizadores, foram escolhidas 20 aplicações entre as quais encontram-se as aplicações desenvolvidas nos subprojectos CURB (*Cooperative Urban Applications*) e CINT (*Cooperative Inter-urban Applications*). O primeiro tipo de aplicações encontra-se focado na gestão de trânsito: cada utilizador tem informação personalizada para que possa otimizar o seu percurso (estado dos semáforos, velocidade recomendada, entre outros), reduzindo as áreas de congestionamento. As aplicações CINT focam-se em informar o condutor sobre o ambiente que o rodeia – desde alertá-lo acerca de acidentes a limites de velocidade ou condições de meteorologia.

2.1.4. COOPERS

O COOPERS (*COOPerative systEms for Intelligent Road Safety*) [10], [17] é um projeto que surgiu na União Europeia, durante o programa FP6-IST, com o intuito de criar uma rede sem fios onde fosse possível estabelecer a comunicação entre veículos e infraestruturas e, deste modo, melhorar a segurança nas estradas e a gestão de trânsito. As suas principais aplicações passam pela partilha de informação acerca de incidentes que possam ocorrer nas estradas (acidentes, buracos, …) e sobre regras de trânsito temporárias (obras na estrada, limites de velocidade, …), escolha de rotas mais rápidas e/ou seguras, pagamento de portagens e recolha de dados acerca da condução dos utilizadores [17].

Neste projeto existem dois tipos de comunicação: V2I (*Vehicle to Infrastructure*) e I2V (*Infrastructure to Vehicle*). A primeira permite obter dados sobre a viagem dos condutores,

como a velocidade, posição e condições meteorológicas que depois da sua análise permitem obter informação sobre o estado do trânsito. Já a comunicação I2V possibilita fornecer informação que promova a segurança dos condutores.

A validação do sistema foi feita através de testes em quatro países da União Europeia, utilizando diferentes meios de comunicação como: DSRC (*Dedicated-Short Range Communications*), GPRS (*General Packet Radio Service*), DAB (*Digital Audio Broadcast*) e CALM IR (*Continuous Air-interface for Long to Medium range InfraRed*) [10]. Os resultados em Itália demonstraram que a existência de sistemas cooperativos como o COOPERS têm uma influência benéfica no comportamento dos condutores.

2.1.5. CAPTIV

O CAPTIV (*Cooperative strAtegies for low Power wireless Transmissions between Infrastructure and Vehicles*) [7], [18], [19] é um projeto desenvolvido pelo grupo ITS-SIG (*Scientific Interest Group on Intelligent Transport Systems*) [20] e tem como objectivo a utilização das infraestruturas presentes nas estradas (sinais de trânsito) para transmissão de informação aos condutores a partir de uma rede de sensores sem fios (WSN – *Wireless Sensor Network*).

Este projeto procura encontrar soluções com baixo consumo de energia [18] que permitam transmitir informação acerca da estrada, através de infraestruturas já existentes [7]. O sistema foi desenvolvido de modo a poder ser implementado em qualquer local onde a velocidade máxima dos veículos não ultrapassasse os 90 km/h. A comunicação é feita por rádio na banda de frequência dos 2.4 GHz utilizando a tecnologia Zigbee e tem quatro modos distintos: V2V (*Vehicle to Vehicle*), V2R (*Vehicle to Road*), R2V (*Road to Vehicle*) e R2R (*Road to Road*). A rede tem uma cobertura mínima de 100 metros, suporta até 100 utilizadores e a transmissão de informação é rápida o suficiente para que o condutor tenha acesso a dados atualizados e tempo para os analisar [6], [7].

2.1.6. Outros Projetos

2.1.6.1. VII

Atualmente, nos Estados Unidos da América é utilizado um sistema, VII (*Vehicle Infrastructure Integration*), que permite a comunicação entre veículos e as infraestruturas presentes nas estradas e também entre veículos de maneira segura [6], [21]. Para tal, é utilizada comunicação rádio a curtas distâncias – DSRC.

2.1.6.2. Smartway

Tal como no projeto anterior, o projeto japonês *Smartway* utiliza a DSRC para estabelecer a comunicação entre as infraestruturas e os veículos. A arquitetura desenvolvida permite enviar aos condutores informação visual acerca do estado das estradas, informação sobre o estado do trânsito através de rádio e alertas sobre situações de perigo.

2.1.6.3. Deteção da Presença de Animais na Estrada

Em Itália, foi criado um sistema distribuído sem-fios que permite a deteção de eventos de animais a atravessar estradas, utilizando uma WSN, como método de prevenção de colisões entre animais e veículos [22]. A deteção é feita através de sensores equipados com radares Doppler que se encontram instalados nas estradas. Estes sensores detetam a presença de veados nas chamadas zona de segurança - zonas dentro do alcance dos sensores. A informação destes sensores é propagada até ao sistema de controlo, onde é processada e pode dar origem ao acionamento dos sinais luminosos mais próximos do local onde o evento foi detetado. A rede é constituída por vários nós instalados em posições fixas e conhecidas ao longo das estradas. Cada nó encontra-se equipado com dois radares Doppler. Com estes radares voltados para o exterior das estradas, é possível

verificar as variações dos sinais e se estas ultrapassam o *threshold* definido, detetando assim a aproximação de animais selvagens. Enquanto os nós da WSN são responsáveis pela transmissão de informação, uma unidade de controlo irá enviar comandos para ligar/desligar sinais luminosos, alterar o intervalo de amostragem dos sensores e iniciar o diagnóstico do sistema.

2.1.7. Análise dos Projetos e das Tecnologias Utilizadas

Como se pode verificar, todos estes projetos apresentam semelhanças, como a utilização de tecnologias sem fios para estabelecer a comunicação entre os diferentes elementos dos sistemas e o facto de as infraestruturas serem utilizadas como provedores de serviços e os veículos como consumidores, exceto no projeto CVIS onde não existe nenhuma estrutura definida. A informação partilhada varia consoante as aplicações desenvolvidas por cada projeto, mas em todas as aplicações de segurança existe a preocupação em enviar o mais rápido possível informação sobre os sinais de trânsito de que o condutor se está a aproximar e avisá-lo de situações que possam colocar a sua segurança em risco.

Quanto às tecnologias de comunicação, a mais utilizada é o DSCR. Esta tecnologia opera na banda dos 5.8 – 5.9 GHz, é utilizada em aplicações de segurança na comunicação entre veículos e infraestruturas e tem como base de funcionamento o protocolo 802.11p, desenvolvido especificamente para aplicações na área de ITS. O DSRC pode ser utilizado na recolha de informação por parte das infraestruturas e na partilha desta num raio não superior a 1km [6], [23].

Outra das tecnologias utilizadas é o Zigbgee, que tem bastante interesse em projetos onde é necessário fazer a recolha rápida de informação de sensores e estabelecer comunicações rápidas e eficazes. Outro modo de obter comunicações rápidas, eficazes e com velocidades de transmissão elevadas é através da utilização do Wi-Fi (IEEE 802.11a/b/g/n) [23]. Estas tecnologias permitem suportar a comunicação, tendo baixo custo e sendo de fácil instalação.

Testes realizados para validação dos diferentes projetos revelaram que a existência de sistemas cooperativos têm uma influência benéfica no comportamento dos condutores [17] e que grande parte dos participantes dos testes realizados no âmbito do projeto Smartway estaria interessada em comprar uma ITS-On-Board-Unit para comunicação com as infraestruturas [24]. Apesar de projetos como o CVIS, COOPERS e SAFESPOT terem surgido à cerca de 10 anos, esta área continua relevante e a despertar o interesse da comunidade científica como se pode ver nas seguintes secções especiais em revistas científicas [25], [26] e pelas contribuições realizadas desde então, como, por exemplo, [27]–[29].

2.2. Sinalização de Mensagem Variável

A Sinalização de Mensagem Variável (SMV) é definida como sinalização que contém elementos simbólicos e/ou textuais que podem ser alterados consoante a informação que se deseja transmitir [30]. Estes sinais podem ser classificados de acordo com a sua funcionalidade como: Painéis de Mensagem Variável (PMV), Painéis Luminosos de Mensagem Única (PLMU) ou Sinais Luminosos de Afetação de Vias (SLAV).

Os PMVs contêm áreas onde é possível mostrar informação textual tal como gráfica. Estes sinais são normalmente utilizados para transmitir informação relativa a acidentes, condições meteorológicas e sugestões de percursos [31], [32], quando não existe informação relevante para ser mostrada, apresentam apenas a horas ou uma mensagem que indica que o sinal se encontra ativo [30].

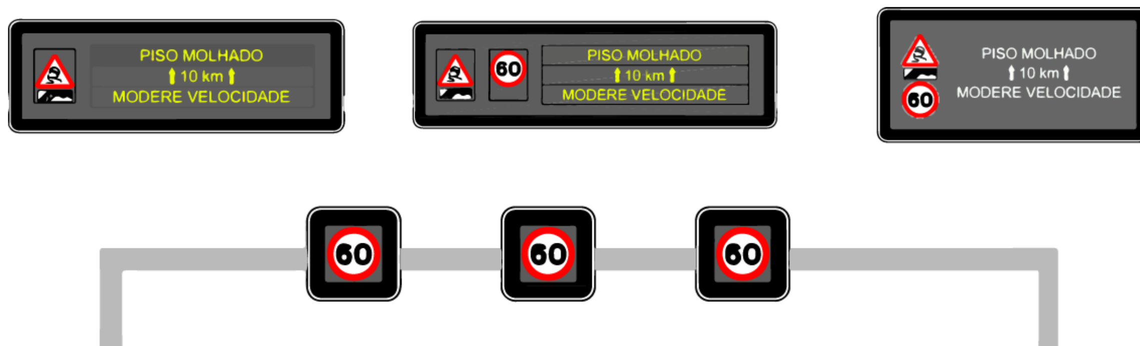


Figura 3 - Diferentes tipos de painéis de mensagem variável, adaptado de [30]

Os PLMUs são sinais transversais com área apenas para elementos gráficos e podem ser utlizados em situações em que é necessário alterar a sinalização normal das vias, por exemplo: limites de velocidade temporários, neve ou gelo na estrada. Por norma, estes sinais são colocados nos dois lados da via.

Figura 4 - Exemplo de painéis luminosos de mensagem única [30]

Os SLAVs são sinais luminosos que fornecem informação acerca de vias de trânsito, indicando se os condutores podem ou não circular nelas. Estes sinais podem apresentar três símbolos distintos: uma seta vertical verde, uma seta inclinada amarela ou uma cruz vermelha.

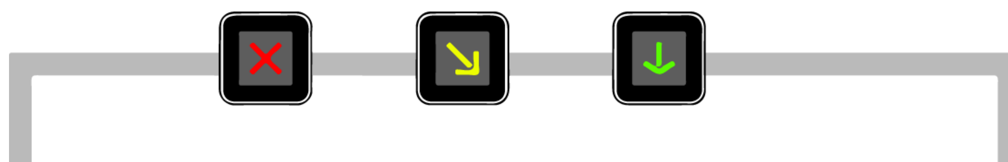


Figura 5 - Sinais luminosos de afetação de vias [30]

Em Portugal, a gestão da informação fornecida pelos SMVs encontra-se a cargo das concessionárias e subconcessionárias [30]. No caso da Brisa, existe uma plataforma *web-based*, introduzida em 2005, responsável pela gestão de infraestruturas – o ATLAS [31]. Com a informação obtida através do processamento de dados vindos de diversos

sistemas de aquisição como sistemas de contadores, CCTV, estações meteorológicas, etc., o ATLAS permite acompanhar a evolução do tráfego, criar alertas e estimar a duração de um percurso. Os alertas gerados pela plataforma são confirmados pelos operadores presentes nos centros de gestão de tráfego da Brisa Manutenção e Operação e, caso seja necessário, o ATLAS permite que sejam informados meios de assistência e/ou mostradas mensagens nos sinais de mensagens variáveis.

2.3. Redes de Petri e Ferramentas de Automatização de Projecto

Uma rede de Petri é um formalismo gráfico e matemático utilizado na modelação de sistemas [33]. Este conceito começou a ser desenvolvido por Carl Adam Petri [33], [34]. Estas redes são grafos orientados compostos por dois tipos de nós: lugares e transições.

Os lugares representam estados do sistema e são representados graficamente por círculos. Um lugar diz-se marcado quando contém pelo menos uma marca (*token*) [33], [34]. As transições representam eventos e são representados graficamente por barras ou quadrados. Os arcos deste tipo de grafo apenas permitem ligações entre nós distintos, ou seja, de lugar para transição ou de transição para lugar, e podem ter diferentes pesos.

Uma transição dispara quando contém os *tokens* necessários no lugar de entrada (origem do arco) e quando o evento associado com a transição ocorre. Após o disparo da transição, os *tokens* do lugar de origem são consumidos e os dos lugares de saída são criados [33], [34]. A distribuição destes *tokens* depende do peso dos arcos da transição para os lugares de saída (Figura 6).

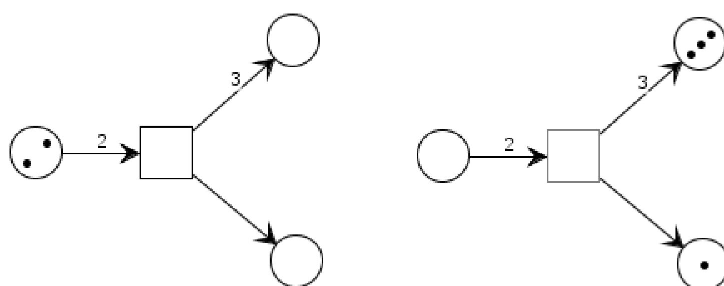


Figura 6 - Demonstração do disparo de uma transição

Uma rede de Petri pode ser utilizada como ferramenta de modelação de sistemas de monitorização e controlo. A receção e envio de mensagens representam eventos do sistema [34]. Quando uma mensagem é recebida, é procurada a transição a que a mensagem se encontra associada e a transição dispara e, desta forma, tem-se o estado atual do sistema. Quando um sinal de saída é gerado pela rede, é possível afetar o processo e assim controlá-lo [35].

2.3.1. Redes de Petri IOPT

Dentro da classe das redes de Petri é possível encontrar as redes IOPT [36], [37]. Estas redes distinguem-se das redes de Petri usuais pelo facto de permitirem definir sinais e eventos tanto de entrada como de saída, e ainda *arrays* com tabelas de informação. Estas características adicionais foram pensadas de modo a que a comunicação entre o controlador (modelado através da rede) e o sistema controlado fosse possível [38].

Os sinais de entrada permitem obter informação sobre o mundo exterior, já os sinais de saída permitem realizar operações como ativar atuadores e envio de mensagens para outros sistemas [37], [38]. Os eventos encontram-se normalmente associados a alterações nos sinais. Desse modo, temos que os eventos de entrada são influenciados pelos sinais de entrada e os eventos de saída afetam os sinais de saída. Numa situação em que se deseje realizar comunicação entre subsistemas, são utilizados eventos que não dependem de variações nas entradas e saídas do sistema, os chamados eventos autónomos.

2.3.2. IOPT-Tools

A criação de redes IOPT pode ser feita através das IOPT-Tools, um conjunto de ferramentas *web-based* disponíveis *online* (http://gres.uninova.pt), que permitem desenvolver e testar controladores para diversos sistemas digitais. Estas ferramentas incluem um editor [39], um simulador [40], ferramentas de verificação de propriedades [41], um depurador e geradores de código automáticos [42], [43] (Figura 7).

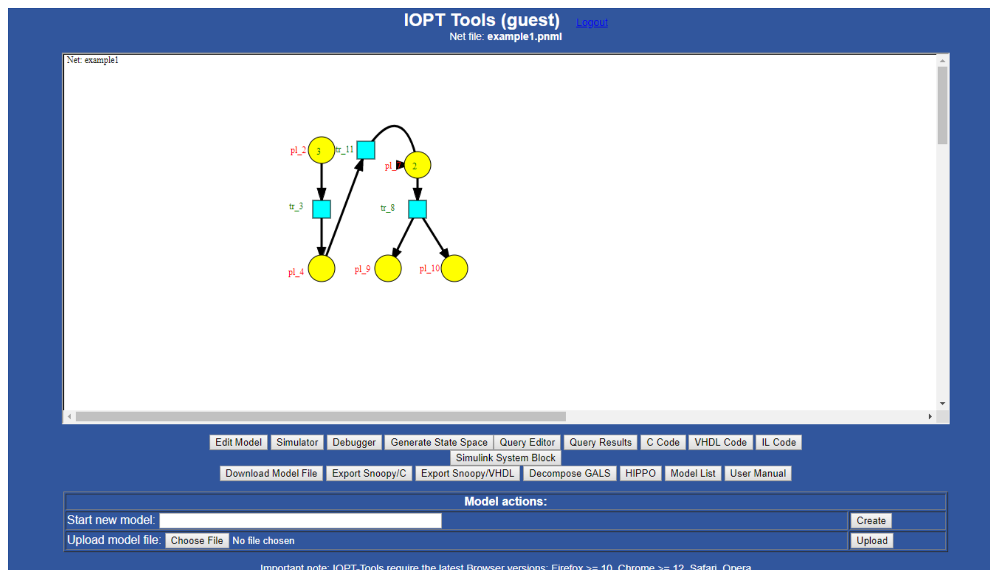


Figura 7 - Página principal das ferramentas IOPT

A interface do editor contém uma caixa de ferramentas no lado esquerdo, uma área de desenho no centro e um editor de propriedades no lado direito (Figura 8). Na caixa de ferramentas é possível encontrar todos os elementos gráficos necessários para desenhar a rede de Petri (arcos, lugares, transições, sinais de entrada/saída, ...) para além de outras ferramentas de edição. No editor de propriedades é possível realizar ações como alterar o nome dos elementos da rede, criar condições de guarda e alterar o tipo de sinais e eventos. A área de desenho, como o nome indica, é onde a rede de Petri é desenhada.

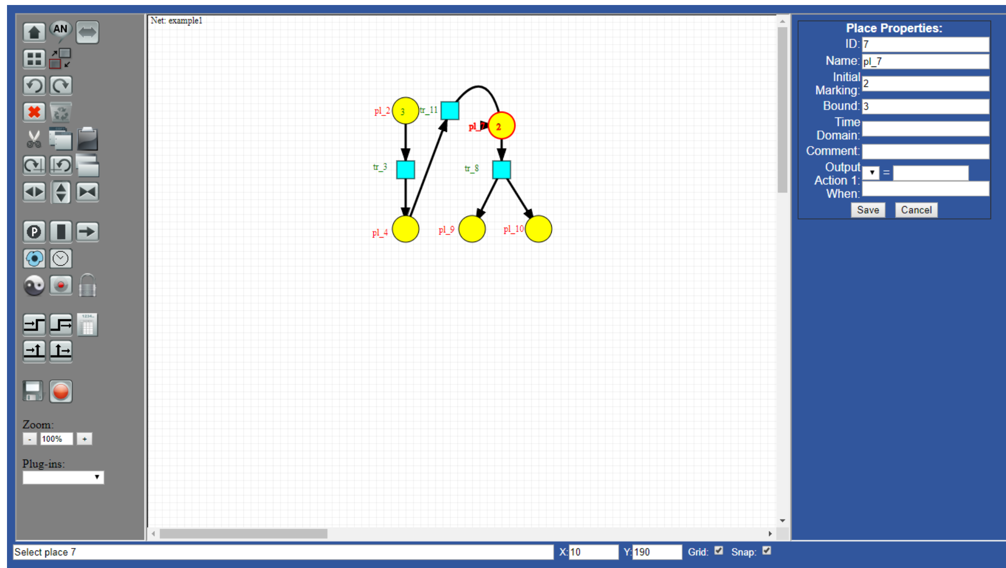


Figura 8 - Editor de redes IOPT

O simulador (Figura 9) permite testar a rede de Petri sem a implementar no *hardware*, tornando mais fácil e rápida a correção de erros de modelação que possam ocorrer. A sua interface inclui uma barra de ferramentas, onde é possível configurar a velocidade da simulação, reiniciá-la, repeti-la ou executá-la continuamente até atingir um *breakpoint* definido pelo utilizador. No centro encontra-se a área de visualização com a rede de Petri, onde é possível verificar o estado da rede em tempo real e ainda forçar o valor de sinais. Esta última ação também pode ser efetuada na barra de estados, localizada no lado direito da interface.

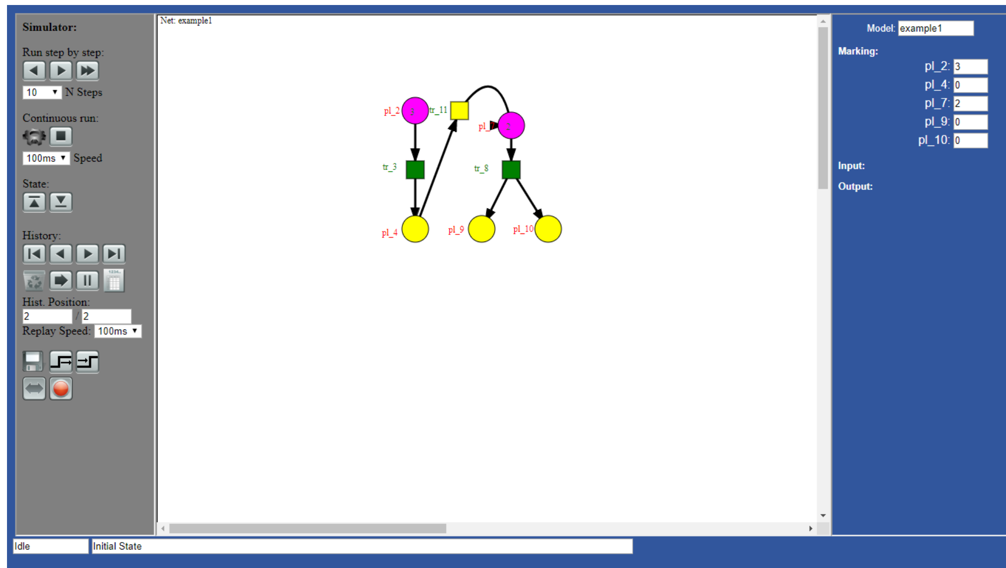


Figura 9 - Simulador de redes IOPT

A interface gráfica do depurador [44] é semelhante à do simulador e permite verificar o funcionamento em tempo real do controlador depois de implementado no *hardware*. A comunicação entre o depurador e o protótipo é feita através de uma ligação TCP/IP.

Os geradores de código automáticos criam todos os ficheiros necessários para implementar o controlador no *hardware* desejado. Neste momento, encontram-se disponíveis geradores de código C, VHDL (VHSIC Hardware Description Language), Javascript, IL (Instruction List) [45] e Simulink System Block.

Arquitectura Proposta

Neste capítulo é proposta uma arquitetura para sinais de trânsito e a utilização das ferramentas IOPT no seu desenvolvimento, monitorização e controlo remoto. As IOPT-Tools são um conjunto de ferramentas de automatização de projeto com capacidade de operação remota que permitem especificar, validar e implementar controladores. Pretende-se que com a arquitetura sugerida seja possível conseguir um sistema de informação que quando combinado com outros sistemas já disponíveis no mercado, permita aumentar a segurança dos utilizadores das vias públicas.

3.1. Descrição Geral da Arquitetura

O sistema a ser desenvolvido procura encontrar um método para partilhar informação entre sinais de trânsito e desta forma permitir que, mesmo que um veículo não tenha tempo suficiente para obter toda a informação que cada sinal possui, consiga obtê-las através da comunicação com outros sinais de trânsito na mesma área. Além disto, procura-se também encontrar um método de monitorizar e controlar o funcionamento dos sinais de trânsito. Com estes objetivos em mente, desenvolveu-se uma arquitetura para um sistema que permite que um sinal de trânsito interaja com outros sinais de trânsito, veículos e sistemas de controlo e monitorização remotos (RCMS – *Remote Control and Monitoring Systems*), como ilustrado na Figura 10.

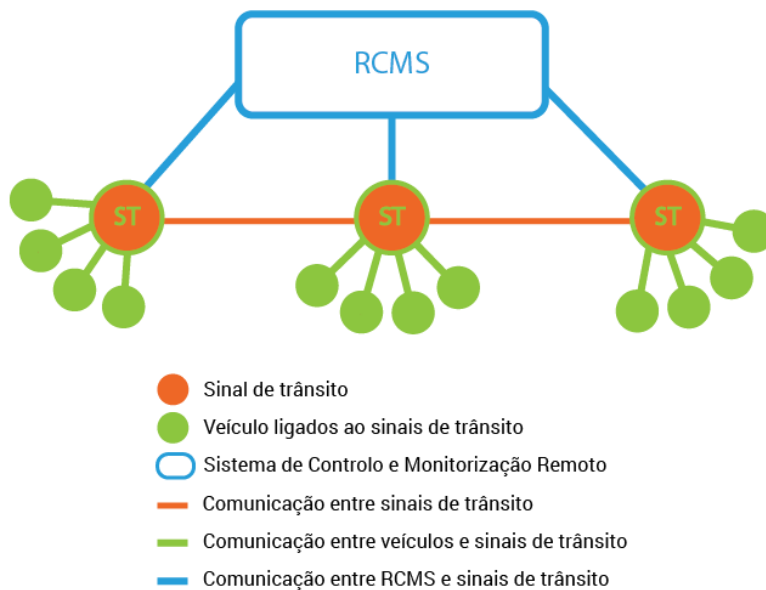


Figura 10 - Interação entre os sinais de trânsito, veículos e sistemas de controlo e monitorização remotos

A arquitetura proposta para este sistema pode ser definida como uma Arquitetura Orientada a Serviços (SOA – *Service-Oriented Arquitecture*, em inglês). Esta arquitetura é composta por serviços, provedores e consumidores dos mesmos. Neste caso, cada sinal de trânsito é simultaneamente provedor e consumidor, e a informação que este disponibiliza é o conjunto dos serviços fornecidos: um sinal de trânsito é provedor de serviços para outros sinais, veículos e RCMS; e um consumidor de serviços de outros sinais de trânsito.

A comunicação entre provedores e consumidores pode ser feita através de dois padrões de troca de mensagens: pedido/resposta e publicação/subscrição. O primeiro padrão é utilizado na interação entre sinais de trânsito e veículos, onde os últimos realizam pedidos de informação aos sinais; já a comunicação com outros sinais e sistemas de controlo e monitorização pode ser feita com qualquer um dos padrões de troca de mensagens, visto que para estas interações os objetivos são, respetivamente, a partilha de informação entre sinais e o envio de comandos e receção de informação do sistema por parte dos RCMS.

A arquitetura proposta para os sinais de trânsito é composta por seis módulos (Figura 11):

- Provedor de serviços para veículos (VP – *Vehicle Provider*) – este módulo é responsável por receber e enviar informação aos veículos.
- Provedor de serviços para sinais de trânsito (SP – *Sign Provider*) – este módulo encarrega-se de receber e enviar informação a outros sinais de trânsito.
- Consumidor de serviços de sinais de trânsito (SC – *Sign Consumer*) – este módulo é um cliente com a função de enviar e receber informação de sinais de trânsito vizinhos;
- Base de dados do sinal (SDB – *Sign DataBase*) – esta base de dados contém toda a informação partilhada com o sinal de trânsito por outros sinais e veículos, para além da informação do próprio sinal;
- Provedor de serviços para os RCMS (RCMSP – *RCMS Provider*) – este módulo é utilizado para receber comandos e enviar informação ao sistema de controlo e monitorização;
- Controlador dinâmico do sinal (SDC – *Sign Dynamic Controller*) – responsável por controlar o comportamento de elementos de sinais de trânsito dinâmicos como, por exemplo, as luzes de um sinal de trânsito ou a mensagem de um PMV. Os módulos VP, SP e SC também são controlados por este módulo, através das mensagens que este recebe do RCMSP, e, por sua vez, envia-lhe atualizações acerca do estado atual do controlador.

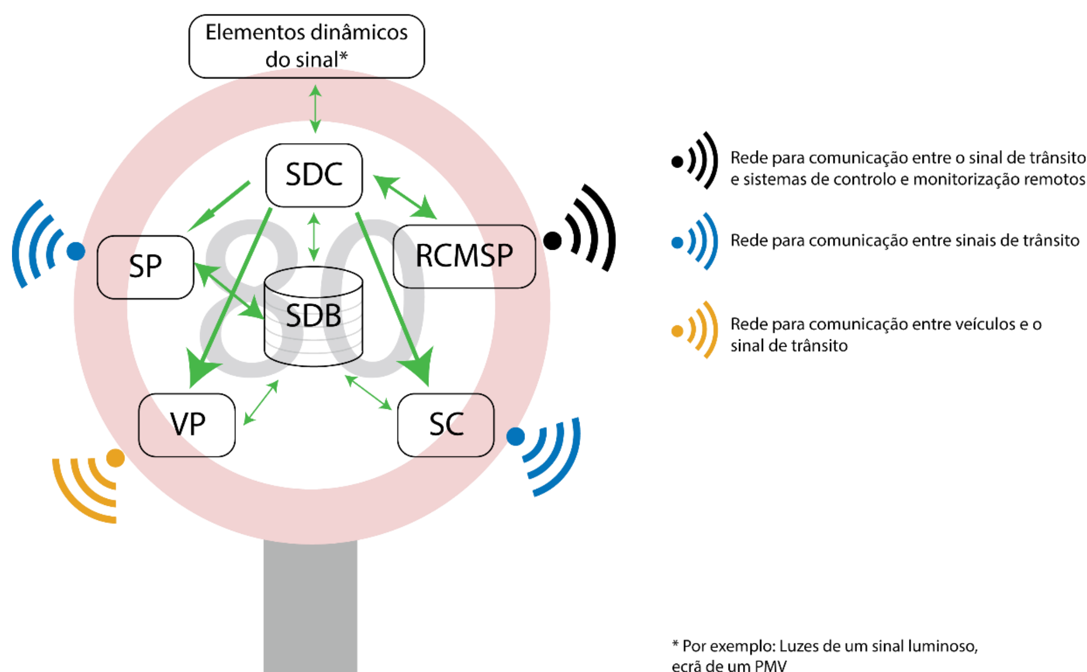


Figura 11 - Módulos da arquitetura proposta para os sinais de trânsito

Com esta arquitetura, todos os sinais de trânsito podem ter os mesmos módulos de comunicação independentemente da sua informação ou comportamento. Apenas no caso de sinais de trânsito com comportamento dinâmico é necessário adaptar o controlador dinâmico do sinal. Para desenvolver este controlador propõe-se a utilização das ferramentas IOPT.

Para que seja possível a interação com outros sinais, veículos e sistemas de controlo e monitorização é necessário escolher pelo menos uma tecnologia de comunicação sem fios. A tecnologia de comunicação sem fios escolhida para este sistema foi o Wi-Fi, dado que permite estabelecer ligações de confiança, atingir velocidades de transmissão de dados elevadas e ser de fácil implementação. Para além disso, como se pode ver no capítulo sobre Trabalhos e Tecnologias Relacionados, o Wi-Fi é uma das tecnologias já utilizadas em sistemas criados para promover a segurança de condutores. É ainda possível tirar partido do facto desta tecnologia poder ser utilizada tanto em redes estruturadas como em redes sem nenhum ponto de acesso (AP – *Access Point*) definido para criar duas redes diferentes para comunicação com veículos e outros sinais de trânsito. No entanto, é de notar que poderiam ter sido utilizadas outras tecnologias para os módulos de comunicação. No caso da comunicação entre sinais de trânsito, poderiam

ter sido utilizadas tecnologias com ou sem fios, enquanto para estabelecer a comunicação com os veículos continuaria a ser necessária uma tecnologia sem fios.

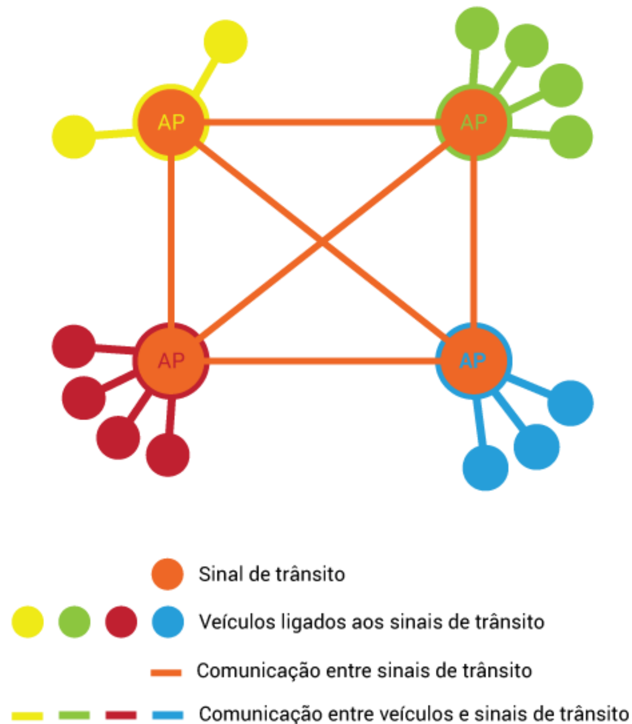


Figura 12 – Topologias de rede utilizadas no projeto

3.2. Comunicação entre Sinais de Trânsito – Módulos SP e SC

A partilha de informação entre os sinais de trânsito através de Wi-Fi requer que os sinais se encontrem numa rede em que seja possível comunicarem. Visto que podem existir várias situações que podem causar a alteração da sinalização de uma rodovia – alterações na legislação, condições meteorológicas, obras, entre outras – deve ser possível adicionar ou remover sinais de trânsito sem que seja necessário reconfigurar os restantes sinais. Para além disso, a falha de um dos sinais não deve impedir o funcionamento da rede.

Tendo estes fatores em conta, a interface utilizada para a comunicação entre os sinais de trânsito foi configurada como uma *mesh network*. Este tipo de rede permite conectar todos os dispositivos de uma rede através de múltiplos caminhos, sem que haja a necessidade de utilizar pontos de acesso. A existência de vários caminhos permite que mesmo que uma das ligações entre nós da rede deixe de funcionar, continue a ser possível estabelecer a comunicação entre todos os nós, via caminhos alternativos.

Um sinal de trânsito é um nó da rede, desta forma é possível estabelecer várias rotas de comunicação entre os sinais. Assim, mesmo que um dos sinais deixe de funcionar, os restantes continuem a trocar informação. Cada nó desta rede é composto pelos módulos SP e SC, que correspondem a um servidor e um cliente TCP/IP (*Transmission Control Protocol/Internet Protocol*). Estes módulos são responsáveis por responder e realizar pedidos a outros sinais, respetivamente.

3.3. Comunicação entre Sinais de Trânsito e Veículos – Módulo VP

A outra interface é utilizada durante a comunicação entre sinais de trânsito e veículos. Para esse efeito, é configurado um ponto de acesso (*access point*, AP) no sinal de trânsito que aguarda por ligações por parte dos veículos. Quando um veículo se liga à rede difundida pelo sinal, recebe a informação que o último tem armazenada, ou seja, tanto a informação do próprio sinal como a que este obteve durante a comunicação com outros sinais. Desta maneira, problemas que possam surgir pelo facto do Wi-Fi não lidar eficazmente com trocas rápidas de pontos de acesso, são contornados, pois mesmo que um veículo não se consiga ligar a todas as redes difundidas, conseguirá ter acesso à informação de todos os sinais que o rodeiam. Este ponto de acesso corresponde ao módulo VP da arquitetura.

3.4. Monitorização e Controlo dos Sinais de Trânsito – Módulos SDC e RCMSP

O controlador dinâmico de cada sinal de trânsito deve ser desenvolvido de acordo com o comportamento pretendido para esse sinal. Para desenvolver este módulo, propõe-se o uso das ferramentas IOPT. O desenvolvimento de controladores com estas ferramentas contém as seguintes etapas: desenvolvimento de uma rede IOPT capaz de representar o comportamento pretendido para o sinal de trânsito e para os módulos de comunicação com veículos e outros sinais de trânsito, que é criada no editor de modelos IOPT; verificação do funcionamento da rede através do simulador de redes IOPT e, caso seja necessário, atualizar o modelo criado; verificação das propriedades da rede através da criação de *queries*, geração do espaço de estados do modelo e análise ao resultado das *queries* realizadas e atualização do modelo, caso se verifique necessário; implementação do código do controlador criado pelas ferramentas de geração automática de código C, que inclui um servidor HTTP (*Hypertext Transfer Protocol*) que suporta a monitorização e controlo remoto do sinal, sendo utilizado como provedor dos sistemas de controlo e monitorização; e, por fim, é realizada a programação e execução do controlador do sinal de trânsito.

Desenvolvimento e Validação

Para validar a arquitetura proposta para sinais de trânsito e o uso das ferramentas IOPT-Tools no seu desenvolvimento, monitorização e controlo, foi desenvolvido e testado um conjunto de protótipos. Neste capítulo são descritos os materiais e as tecnologias utilizadas para desenvolvê-los, bem como o seu funcionamento, as mensagens que enviam, detalhes de implementação, a abordagem de desenvolvimento e os testes realizados aos protótipos.

4.1. Materiais

Para construir os protótipos é necessário *hardware* com capacidade de armazenamento de ficheiros e integração de Wi-Fi. Tendo estes fatores em conta, foram utilizados os seguintes materiais:

- 3 Raspberry Pi 3;
- Adaptadores de Wi-Fi;
- Fontes de alimentação.

4.1.1. Raspberry Pi 3

O Raspberry Pi 3 [46] (Figura 13) é um computador com dimensões de um cartão de crédito que permite desenvolver diversas aplicações. Este dispositivo tem o sistema operativo Raspian e o PIXEL [47] (***Pi Improved Xwindows Environment, Lightweight***) como ambiente gráfico.

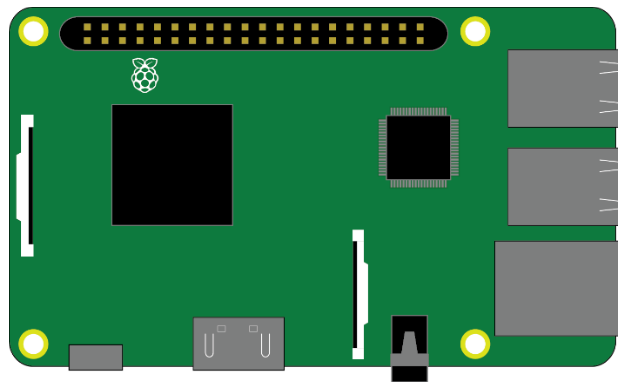


Figura 13 - Esquema representativo do Raspberry Pi 3 Model B

O modelo utilizado (B) tem as seguintes características:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
- 1 GB de RAM;
- 4 Portas USB;
- 1 Porta Ethernet.

4.1.2. Adaptadores de Wi-Fi

Como o Raspberry Pi 3 possui apenas uma interface de Wi-Fi, utilizada para criar o ponto de acesso para a comunicação com os veículos, é necessário adicionar um adaptador de Wi-Fi para a *mesh network*. Os adaptadores utilizados foram:

- EZ Connect N 150Mbps Wireless USB2.0 [48] da SMC Networks – este adaptador suporta os modos de Wi-Fi b/g/n e os *standards* de encriptação WPA e WPA2.

- Xavi Geral 54Mbps USB2.0 [49] da Clix – este adaptador suporta os modos de Wi-Fi b/g e encriptação WPA.
- Wireless G USB Adapter [50] da Belkian – este adaptador, tal como o anterior, suporta os modos de Wi-Fi b/g.

Figura 14 - Esquema representativo de um adaptador USB de Wi-Fi

4.2. Descrição

Cada protótipo de um sinal de trânsito foi feito com o Raspberry Pi 3 (RPi3) e um adaptador de Wi-Fi. A adição de um adaptador foi feita para configurar a *mesh network*, visto que o RPi3 possui apenas uma interface de Wi-Fi, que é utilizada para criar o ponto de acesso para a comunicação com os veículos.

A interface de Wi-Fi do Raspberry Pi foi configurada como ponto de acesso utilizando os programas hostapd [51], [52], para configurar a rede difundida – SSID, canal, modo de Wi-Fi, palavra-chave da rede, entre outros –, e isc-dhcp-server [53], que é responsável pela configuração do servidor DHCP (*Dynamic Host Configuration Protocol*) que se encarrega da configuração de rede aos terminais – atribuição de endereços de IP aos clientes, definição de máscaras de rede, servidores de DNS (*Domain Name System*).

A interface utilizada para comunicar com outros sinais de trânsito foi configurada como *mesh network*, recorrendo ao módulo B.A.T.M.A.N (*Better Approach To Mobile Ad-hoc Networking*) *advanced* [54] que implementa o protocolo de *routing* B.A.T.M.A.N. Com este protocolo cada nó tem apenas informação acerca do melhor *next-hop* para que

os dados cheguem aos restantes nós. A implementação do sistema de comunicação e do sistema de monitorização e controlo foram feitas separadamente.

O sistema de comunicação foi desenvolvido, em Python, utilizando vários módulos com diferentes funções, o que permitiu ter um sistema mais organizado e facilmente compreensível. O sistema de monitorização e controlo foi implementado através da adaptação do código do controlador gerado pelas ferramentas IOPT.

4.2.1. Desenvolvimento do sistema de comunicação

Os módulos do sistema de comunicação foram desenvolvidos em Python, visto que as ferramentas de desenvolvimento para esta linguagem já se encontram instaladas no sistema operativo deste dispositivo – Raspbian. Os módulos foram desenvolvidos recorrendo a bibliotecas standard [55] como *socket*, *threading*, *json*, *select* e *xml.etree*. Na Figura 15 encontram-se os módulos desenvolvidos para implementar o sistema de comunicação:

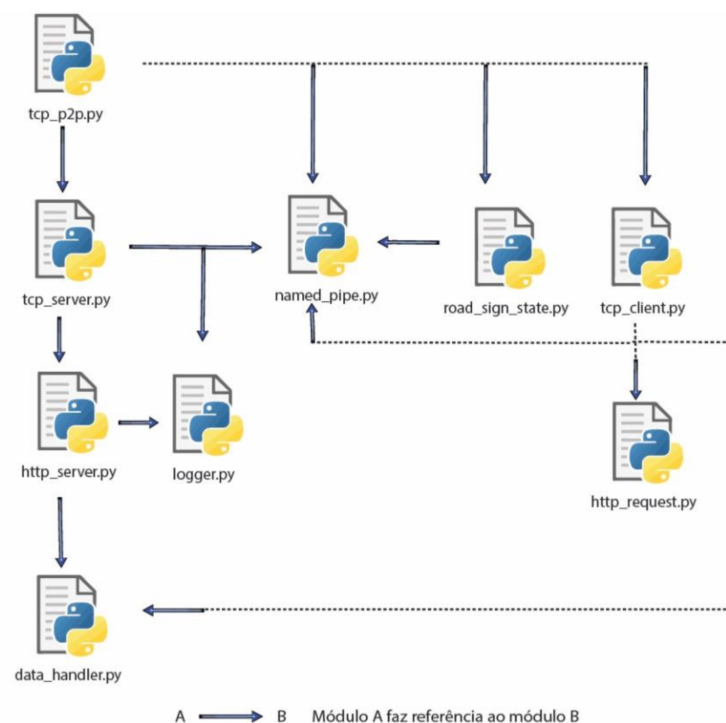
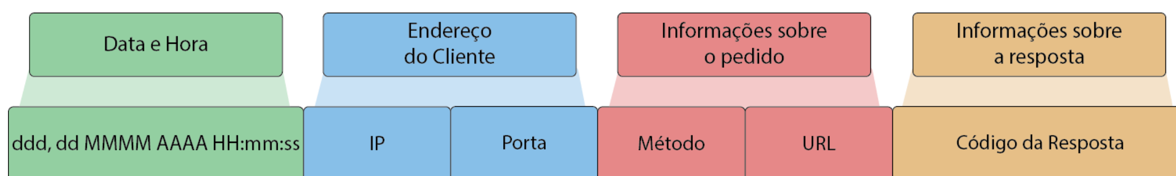


Figura 15 - Módulos criados para implementar o sistema de comunicação

O módulo ***data_handler*** é responsável pela consulta e armazenamento da informação no sinal de trânsito. Aqui encontram-se as funções que permitem ler a informação do sinal de trânsito, informação enviada por outros sinais de trânsito e atualizá-las ou adicionar nova informação. Já o módulo ***logger*** é responsável por atualizar o ficheiro com o histórico das ligações feitas aos servidores do sinal de trânsito. Este módulo permite também consultar a informação deste mesmo ficheiro. Cada entrada do histórico, como indicado na Figura 16, contém a data e hora a que foi recebido o pedido, o endereço do cliente, a URL (*Uniform Resource Locator*), o método utilizado e o código da resposta do servidor.



[Mon, 11 December 2017 02:57:18] 192.168.41.2:34020 POST /info 200

Figura 16 - Estrutura de uma entrada no histórico do servidor

O módulo ***http_request*** é responsável por criar uma classe que permite criar pedidos com os métodos POST e GET. Este módulo é utilizado pelo cliente TCP/IP para criar pedidos POST com a informação do sinal no corpo da mensagem.

O processamento dos pedidos feitos pelo cliente e a criação das respostas adequadas é feita através do módulo ***http_server***. Este módulo verifica se o pedido foi feito utilizando métodos válidos – POST e GET –, retira qualquer informação sobre o cliente que tenha sido enviada no corpo, ou cabeçalho do pedido e cria a resposta adequada à URL pedida. O servidor é capaz de criar respostas a pedidos de informação – URL: /info – por parte de sinais de trânsito com a informação do sinal, a pedidos de informação por parte de veículos – URL: /info – com toda a informação sobre sinais que tiver, a pedidos do histórico – URL: /log, /log10 ou /log20 – com o registo completo das comunicações, das últimas dez, ou vinte. Caso a URL do pedido não seja reconhecida, é criada uma resposta com uma mensagem de erro, o mesmo acontece se o método utilizado não for suportado. A informação fornecida aos clientes é obtida com as funções disponibilizadas

pelo módulo ***data_handler***, que também é utilizado para armazenar qualquer informação recebida com o pedido. Se o sinal de trânsito se encontrar em modo de manutenção, este servidor cria uma resposta com uma mensagem que informa o cliente desta situação.

No módulo ***tcp_client*** encontra-se a implementação da *thread* responsável por aceder ao ficheiro com a informação sobre os servidores dos sinais vizinhos e, utilizando o módulo ***http_request***, realiza pedidos de informação aos mesmos. Caso o sinal receba a resposta antes de atingir o tempo limite definido para cada comunicação, a informação do servidor é armazenada. Quando o sinal de trânsito está em manutenção, o cliente deixa de poder realizar pedidos. Já o módulo ***tcp_server*** permite criar um servidor responsável por gerir as ligações que lhe são feitas. Neste módulo são aceites as novas ligações, recebidos os pedidos, enviadas as respostas criadas pelo ***http_server*** e com o módulo ***logger*** é atualizado o histórico de ligações.

Os módulos ***named_pipe*** e ***road_sign_state*** são utilizados na comunicação entre o sistema desenvolvido para os sinais de trânsito e o sistema de controlo e monitorização. O primeiro contém as funções necessárias para criar, abrir, ler e escrever em *named pipes*. Um *named pipe* trata-se de um canal utilizado para estabelecer a comunicação entre processos, através de mensagens que são lidas pela mesma ordem em que foram escritas, e, devido a este comportamento, é também conhecido como FIFO (*First In, First Out*). Neste módulo são criados dois FIFOs para que seja possível obter comunicação bidirecional entre os sistemas.

O módulo ***road_sign_state*** é responsável por criar a *thread* encarregue de receber mensagens provenientes do sistema de monitorização e controlo e reencaminhá-las para as *threads* dos servidores e cliente TCP/IP, informando-as assim acerca de comandos enviados ao sistema através da rede IOPT. Por fim, tem-se o módulo ***tcp_p2p*** onde são inicializados os servidores, o cliente, criados os FIFOs para comunicação com o controlador IOPT e a *thread* definida no módulo ***road_sign_state***.

4.2.2. Funcionamento do Servidor TCP/IP

O servidor tem um endereço definido por um IP e uma porta de comunicação e fica à espera de ligações de clientes. Quando um cliente tenta estabelecer uma ligação com o servidor, este último é responsável por aceitá-la, atribuir um endereço ao cliente e criar uma *socket* que represente essa ligação. Depois da ligação ser aceite, o servidor recebe o pedido do cliente, cria uma resposta e atualiza o ficheiro com o registo das ligações feitas ao servidor – "log.txt". Após a resposta ter sido enviada, a ligação é encerrada. Visto que o sinal pode receber vários pedidos em simultâneo, o servidor funciona em modo assíncrono, isto significa que todas as operações realizadas pelos clientes – ligação, envio de mensagens e encerramento de ligações - originam eventos que são processados rapidamente pelo servidor.

O processamento de eventos passa por obter uma lista de todas as *sockets* que originaram os eventos que ocorreram desde a última vez que o processamento ocorreu e separá-las em dois grupos distintos: um para as *sockets* prontas para serem lidas (que contêm informação para ser lida, representam novas ligações ou ligações que foram terminadas do lado do cliente) e outro para as ligações que estão prontas para ser "escritas", ou seja, para as que estão à espera de resposta.

As novas ligações são aceites e adicionadas a uma lista com as ligações que têm (potencialmente) pedidos que ainda não foram recebidos; aquelas que enviaram informação têm os seus pedidos lidos, as respetivas respostas criadas e são adicionadas a uma lista com ligações à espera de resposta; as ligações à espera de resposta têm-nas parcial ou totalmente enviadas (dependendo do seu tamanho): se a resposta tiver sido totalmente enviada, a ligação é encerrada e a *socket* é removida da lista, caso contrário, a *socket* permanece na lista; e por fim, as ligações que foram encerradas do lado do cliente, são também encerradas do lado do servidor e eliminadas de todas as listas em que se encontravam.

4.2.3. Funcionamento do Cliente TCP/IP

Do lado do cliente, a maior dificuldade encontra-se em saber quais os endereços dos servidores de outros sinais de trânsito. Para tal, foi criado um ficheiro "peers.txt" que contém a lista de endereços dos servidores a que o sinal se pode conectar. Cada linha do ficheiro contém o IP e a porta de comunicação de um servidor, estando estes separados por um espaço. Antes de iniciar os pedidos, o cliente obtém os endereços de outros sinais a partir do ficheiro, e começa a realizar pedidos sequenciais aos diferentes sinais.

Para evitar que o sinal fique bloqueado ao tentar estabelecer a ligação a um servidor, o cliente tem um limite de tempo para se ligar ao servidor, caso não consiga, tenta ligar-se ao servidor seguinte. Despois de tentar comunicar com todos os servidores da lista, o cliente é desligado durante um intervalo de tempo, já que a informação dos sinais de trânsito não muda regularmente, e por isso não existe necessidade de atualizar a informação constantemente.

4.2.4. Estrutura das Mensagens

As mensagens enviadas pelos sinais de trânsito devem conter informação relevante para o condutor ou sistema de bordo responsável por interpretá-las. Neste sistema são disponibilizados o tipo do sinal (*type*), a sua informação (*sign info*), a localização do sinal (*lat* e *long*) e um identificador (*id*).

Os campos *type* e *sign info* possuem a informação que o condutor deve retirar do sinal para poder tomar decisões acertadas. A localização do sinal também é disponibilizada, visto que pode ser utilizada por outras aplicações, como o SAFESPOT [8], para criar mapas dinâmicos [9] onde é possível visualizar os diferentes sinais e veículos presentes na estrada e a sua informação. É necessário ainda que cada mensagem tenha um identificador do sinal que a enviou para que o sinal de destino possa atualizar a sua informação sobre o sinal de origem e não possua informação contraditória. Na tabela

abaixo encontram-se os diferentes campos das mensagens enviadas pelos sinais de trânsito:

Tabela 1 - Estrutura da informação enviada pelos sinais de trânsito

Campo	Descrição
Id	Valor numérico que permite identificar o sinal de trânsito
Type	Indica o tipo de sinal (informação, perigo, …)
Sign info	A informação que o condutor deve retirar do sinal
Lat *Long*	Indicam a localização do sinal através das suas coordenadas geográficas: latitude e longitude, respetivamente

A troca de informação entre sinais de trânsito é realizada através do protocolo HTTP e é utilizado o método POST. Estes pedidos terão no corpo da mensagem uma *string* com o formato JSON (*JavaScript Object Notation*) e deste modo informação estruturada, mas com um tamanho significativamente menor do que teriam num formato mais rigoroso como o XML (*eXtensible Markup Language*).

Sempre que o sinal receber um pedido, verifica se a URL pedida corresponde àquela que contém a informação do sinal – "/info" –, e, se isto se verificar, cria uma resposta com a sua informação. A informação enviada pelos outros sinais são guardadas num ficheiro. A informação que receber como resposta aos seus pedidos serão guardadas no mesmo ficheiro que as recebidas pelo sinal, enquanto atua como servidor.

4.2.5. Desenvolvimento do sistema de monitorização e controlo

A monitorização e controlo do sinal de trânsito são feitas com o auxílio das ferramentas IOPT. Os modelos de rede de Petri criados permitem ter uma representação gráfica dos módulos de comunicação do sinal e, no caso de sinais dinâmicos, do seu comportamento, facilitando a sua perceção e a deteção de eventuais erros. Dado que os módulos funcionam de modo independente podem ser modelados separadamente.

Os modelos apresentados na Figura 17 foram criados com o editor disponibilizado pelas ferramentas IOPT e têm como base o funcionamento de cada módulo introduzido anteriormente. Durante o desenvolvimento destes módulos foram utilizadas as ferramentas de simulação e de geração de espaços de estados, para corrigir eventuais erros de modelação e verificar que as redes não teriam locais onde pudessem ocorrer situações de *deadlock*.

Destes modelos, é possível retirar várias informações sobre o sinal de trânsito: o estado de cada um dos módulos de comunicação – inicializado ou não, o número de ligações realizadas pelo cliente, o número de ligações feitas aos servidores e o estado de cada ligação.

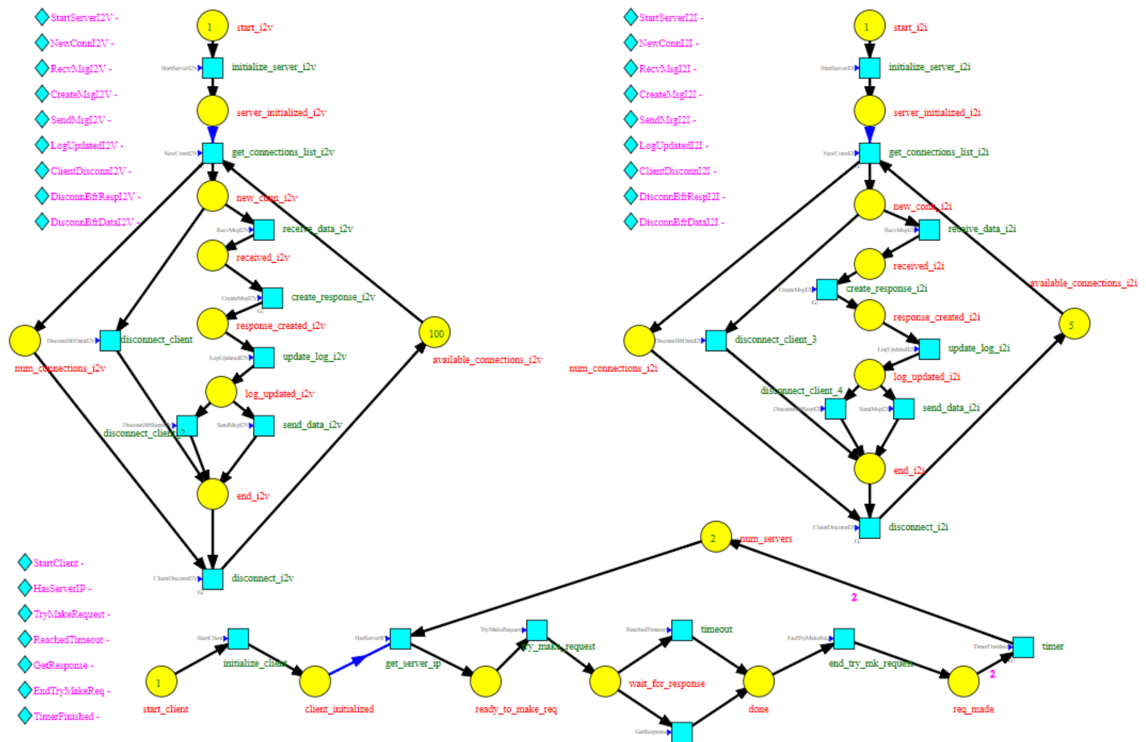


Figura 17 - Redes de Petri utilizadas na monitorização e controlo dos módulos de comunicação

Para monitorizar o sistema, os módulos desenvolvidos comunicam com o controlador gerado a partir da rede IOPT, que por sua vez, geram eventos associados com a ação realizada pelo sistema de comunicação. Estes eventos, descritos na Tabela 2, são utilizados para disparar a transição que irá colocar as marcas no lugar correspondente ao estado atual do sistema.

Tabela 2 - Eventos associados aos módulos de comunicação do sinal de trânsito

	Eventos	
Servidores	StartServerI2I/V	A iniciar servidor
	NewConnI2I/V	Nova conexão
	RecvMsgI2I/V	Pedido recebido
	CreateMsgI2I/V	Mensagem criada
	SendMsgI2I/V	Mensagem Enviada
	LogUpdatedI2I/V	Registo atualizado
	ClientDisconnI2I/V	Cliente desconectado
	DisconnBfrRespI2I/V	Cliente desconectado antes de resposta enviada
	DisconnBfrDataI2I/V	Cliente desconectado antes de pedido recebido
Cliente	StartClient	A iniciar cliente
	HasServerIP	Obteve endereço do servidor
	TryMakeRequest	A tentar fazer um pedido
	GetResponse	Resposta recebida
	ReachedTimeout	Pedido atingiu o limite de tempo
	EndTryMakeReq	Acabou a tentativa de fazer o pedido
	TimerFinished	Temporizador atingiu o limite

Por outro lado, o controlo dos sinais de trânsito é feito de duas formas distintas: através da limitação do número de ligações permitidas a cada módulo: 100 ao provedor de serviços para veículos, 5 para o provedor de serviços para outros sinais e 2 para o consumidor de serviços de outros sinais de trânsito; e através de sinais de entrada, cujo valor irá habilitar o disparo de transições, que por sua vez criam eventos de saída que são utilizados pelo controlador para criar e enviar mensagens para o sistema de comunicação e influenciar o seu funcionamento. Neste caso, foi criado um sinal que permite ligar e desligar o sistema de partilha de informação. Se o sistema for desligado o cliente deixa de poder realizar novos pedidos e os servidores respondem a todas as novas ligações com uma mensagem que indica que o sinal de trânsito se encontra em manutenção – "*Road sign in maintenance*". Enquanto o sinal se encontra em manutenção,

pode ser feita a sua reconfiguração, acedendo ao ficheiro com a informação do sinal ou ao ficheiro com os endereços dos sinais vizinhos. Quando o sistema se encontra ligado realiza e responde a pedidos normalmente.

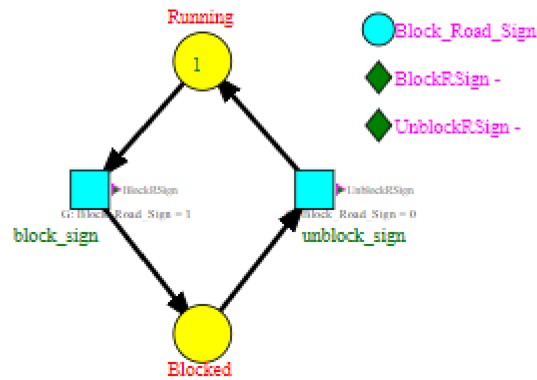


Figura 18 - Rede de Petri utilizada para controlar o funcionamento do sinal de trânsito

O código utilizado para implementar o monitor/controlador foi criado através do gerador automático de código C [42], disponível no *website* das ferramentas IOPT. O gerador cria um arquivo com os seguintes ficheiros:

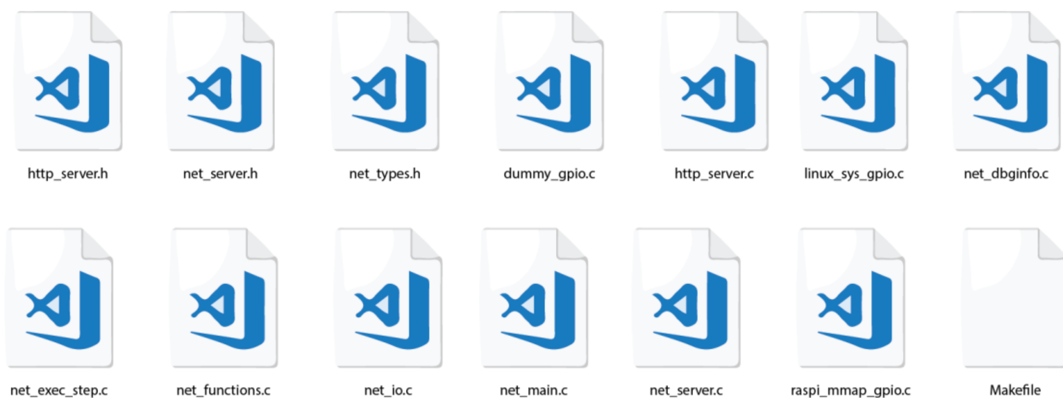


Figura 19 - Ficheiros criados pelo gerador de código C

Os ficheiros gerados podem ser compilados em grande parte dos sistemas, sendo sempre necessário adaptar a interface no ficheiro ***net_io.c*** para se conseguir a arquitetura desejada [37]. Para este sistema, para além de ser necessário alterar o ficheiro ***net_io.c***, também é alterado o ***Makefile***. O primeiro ficheiro contém, inicialmente, cinco funções:

- *InitializeIO* – esta função é executada uma única vez, no início do programa e pode ser utilizada para definir o valor inicial de sinais de entrada e de saída [37]. Neste caso, a função é utilizada para inicializar uma variável de controlo, que indica o último modo de funcionamento do sinal de trânsito – normal ou em manutenção. Esta variável é usada para detetar quando é necessário enviar uma nova mensagem de controlo ao sistema de comunicação.
- *GetInputSignals* – esta função é executada no início de cada ciclo de execução e permite verificar os valores de todos os sinais de entrada, eventos autónomos e ainda dados de outras aplicações [37]. Esta função é utilizada durante a monitorização do sinal de trânsito para ler o FIFO onde são escritas as mensagens vindas do sistema de comunicação. Sempre que é lida nova informação no FIFO, é identificada a mensagem e ativado o evento a que esta se encontra associada.

Tabela 3 - Mensagens recebidas pelo controlador e eventos associados

Mensagem Recebida	*Evento Associado*
"I2I/V_Server started"	StartServerI2I/V
"I2I/V_New connection"	NewConnI2I/V
"I2I/V_Received request"	RecvMsgI2I/V
"I2I/V_Response created"	CreateMsgI2I/V
"I2I/V _Sending response"	SendMsgI2I/V
"I2I/V _Log updated"	LogUpdatedI2I/V
"I2I/V _Client disconnected"	ClientDisconnI2I/V
"I2I/V _Disconnected bfr response"	DisconnBfrRespI2I/V
"I2I/V _Disconnected bfr sending data"	DisconnBfrDataI2I/V
"Client initialized"	StartClient
"Server IP available"	HasServerIP
"Making new request"	TryMakeRequest
"Response available"	GetResponse
"Reached response timeout"	ReachedTimeout
"End try request"	EndTryMakeReq

“Timer finished”	TimerFinished

- *PutOutputSignals* – esta função é executada em todas as etapas do ciclo de execução do controlador duas vezes: uma depois de serem lidos os valores das entradas do sistema e de serem atualizadas as marcações dos lugares da rede, e outra no fim do ciclo para atualizar as saídas [37]. Esta função é utilizada para verificar o estado dos eventos de saída, caso um deles esteja ativo, é verificado se existiu uma alteração no modo de funcionamento do sinal. Se se verificar uma alteração, é enviada uma mensagem ao sistema de comunicação, através de um *named pipe*.

Tabela 4 - Eventos de saída do controlador e mensagens associadas

Evento de saída	*Mensagem enviada*
UnblockRSign	“Maint. mode: 0”
BlockRSign	“Maint. mode: 1”

- *LoopDelay* – esta função é utilizada para definir um tempo de execução do ciclo fixo [37]. Esta função não foi utilizada neste sistema.
- *FinishExecution* – esta função é utilizada para terminar a execução do programa [37]. Para este sistema, esta função tal como a anterior, não foi implementada.

Para além destas funções, foi adicionada a função *resetAllInputEvents* que é utilizada, na função *GetInputSignals*, para desativar todos os eventos autónomos do sistema antes verificar a ocorrência de novos eventos. No ficheiro ***Makefile*** é apenas necessário descomentar a linha “-DHTTP_SERVER” para permitir que o depurador das ferramentas IOPT possa ser utilizado remotamente.

4.3. Testes e Resultados

Após o desenvolvimento dos protótipos foi feita a validação da arquitetura para sinais de trânsito proposta nesta dissertação através de vários testes. A interação com veículos é feita através da ligação à rede difundida pelo sinal, cujo SSID tem o formato "rs_latitude_longitude", e fazendo um pedido ao servidor para veículos, cujo endereço é idêntico para todos os sinais – 192.168.42.1. A interação entre os sinais de trânsito é feita através de uma rede *ad-hoc*, "road_sign_mesh", onde cada sinal tem um IP entre os valores 192.168.41.1 e 192.168.41.254.

Para testar o sistema de comunicação foram efetuados pedidos de informação aos diferentes sinais de trânsito utilizando um computador, utilizado para representar o sistema de comunicação de um veículo. A realização de pedidos ao servidor para veículos permite também verificar o funcionamento dos módulos para comunicação com outros sinais de trânsito, visto que, caso estes não se encontrem configurados corretamente, não é disponibilizada a informação de todos os sinais.

Na figura abaixo, é possível verificar a informação disponibilizada por um dos protótipos de um sinal de trânsito, tal como o registo dos últimos 10 pedidos realizados aos servidores do sinal:

```
{
    "id": "1",
    "info": "120 km/h",
    "lat": "3.87",
    "long": "-9.27",
    "type": "Speed Limit"
}
{
    "id": "3",
    "info": "Saida - Cascais",
    "lat": "3.187",
    "long": "-9.127",
    "type": "Information"
}
{
    "id": "2",
    "info": "stop",
    "lat": "3.875",
    "long": "-9.274",
    "type": "Stop Sign"
}
```

(a)

```
192.168.42.11:4741 GET /info 200
192.168.42.11:4742 GET /log 200
192.168.41.2:51660 POST /info 200
192.168.42.11:5393 GET /info 200
192.168.42.11:5400 GET /info 200
192.168.41.3:50018 POST /info 200
192.168.41.3:50086 POST /info 200
192.168.42.11:5406 GET /info 200
192.168.42.11:5407 GET /info 200
192.168.42.11:5409 GET /info 200
```

(b)

Figura 20- (a) Informação fornecida pelo sinal de trânsito; (b) Registo das últimas 10 ligações feitas aos servidores

O controlador desenvolvido para o sinal de trânsito foi testado utilizando o depurador das ferramentas IOPT. Mais uma vez, através da ligação ao servidor para veículos, é possível aceder remotamente ao controlador do sinal de trânsito e verificar o estado dos seus módulos.

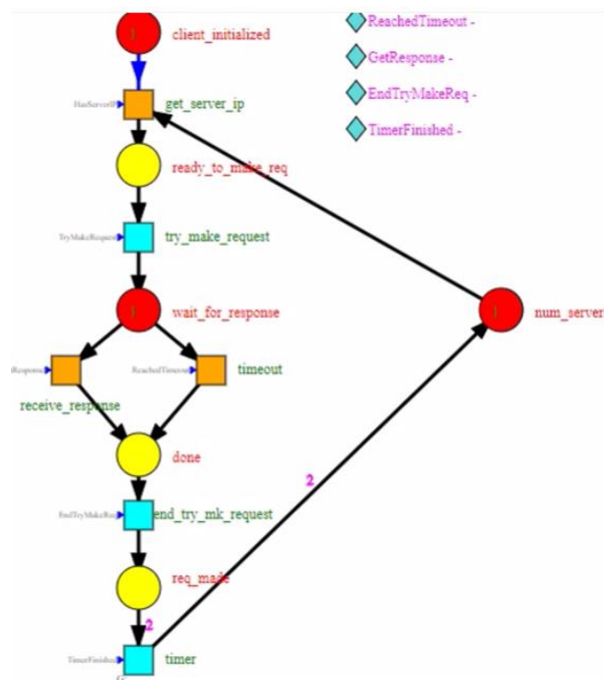


Figura 21 - Monitorização remota do módulo SC

O controlo do sinal de trânsito também foi testado, verificando como a alteração do valor do sinal de entrada responsável por controlar o modo de funcionamento do sinal influencia a resposta a novos pedidos feitas aos servidores do sinal de trânsito. Na figura abaixo, é possível observar as diferentes respostas fornecidas pelo servidor.

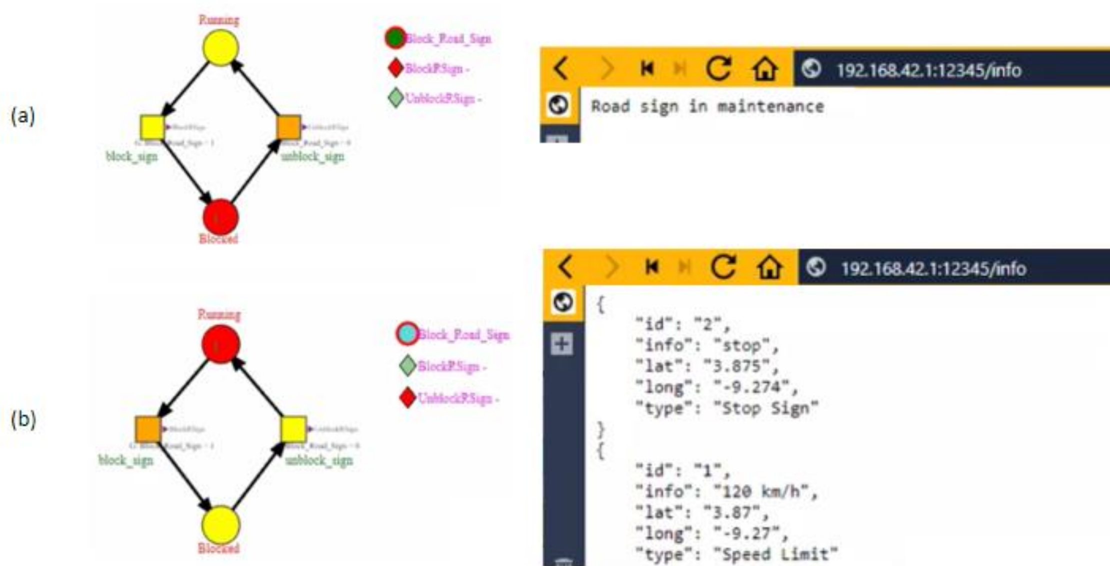


Figura 22 - (a) Resposta do sinal em manutenção; (b) Resposta do sinal em funcionamento normal

Conclusões e Trabalho Futuro

Durante o desenvolvimento deste trabalho verificou-se que existe uma escassez de projetos focados no desenvolvimento das infraestruturas rodoviárias para Sistemas Inteligentes de Transportes, apesar do desenvolvimento de sistemas de comunicação para veículos se encontrar avançado.

Através da revisão bibliográfica foram encontradas arquiteturas interessantes do ponto de vista da interação entre veículos e infraestruturas, e do desenvolvimento destas últimas. Os projetos encontrados tiram partido de tecnologias sem fios para obter e transmitir informação relevante aos veículos, e desenvolver aplicações que permitam não só promover a segurança dos condutores como também retirar dados estatísticos sobre a utilização das vias públicas. O DSCR é a tecnologia mais utilizada nos projetos mencionados. No entanto, tanto o Zigbee como o Wi-Fi (IEEE 802.11a/b/g/n) foram apresentadas como tecnologias rápidas e de confiança para a implementação de arquiteturas que dependam de partilhas de informação entre infraestruturas, ou infraestruturas e veículos.

Despois de estudados os diferentes projetos, foi desenvolvida uma arquitetura que torna os sinais de trânsito capazes de comunicar entre si, com veículos e com sistemas de controlo e monitorização remotos. Nesta arquitetura, como em grande parte das arquiteturas dos projetos encontrados, as infraestruturas – sinais de trânsito, neste caso – são utilizadas como provedores/servidores para os veículos que serão consumidores dos serviços disponibilizados. Um sinal de trânsito será ainda provedor para sistemas

de controlo e monitorização remotos, e pode ter tanto o papel de provedor como de consumidor na interação com outros sinais.

O trabalho desenvolvido permitiu obter uma arquitetura que possibilita a utilização de sinais de trânsito para propagar informação e fornecê-la aos veículos. Desta forma, os veículos têm acesso à informação disponibilizada por vários sinais de trânsito, mesmo que não comuniquem diretamente com eles. Este sistema de partilha de informação pode ser utilizado em conjunto com outros sistemas de aquisição de dados que os veículos e estradas possuem, sendo mais uma forma de obter nova informação e/ou corroborar informação obtida previamente. A utilização das ferramentas IOPT revelou-se um método eficaz para desenvolver controladores para os sinais de trânsito, para além de fornecerem um método para controlá-los e monitorizá-los remotamente. A utilização de eventos de entrada para habilitar transições nos modelos dos módulos de comunicação permitiu acompanhar o seu funcionamento, ao passo que a criação de eventos de saída associados a sinais de entrada permite controlar o funcionamento dos módulos.

Como trabalho futuro, seria de interesse integrar no sistema uma base de dados para armazenamento dos diferentes tipos de informação adquirida; a criação de alertas a partir da informação obtida pelos sinais de trânsito; a integração de outras infraestruturas rodoviárias no sistema como contadores, radares e câmaras; e o estudo de serviços e aplicações de segurança e informativas que possam ser desenvolvidas a partir da informação fornecida pelos veículos.

Referências

[1] «WHO | World Health Organization», *WHO*. [Em linha]. Disponível em: http://www.who.int/en/. [Acedido: 30-Jul-2017].

[2] «WHO | The top 10 causes of death», *WHO*. [Em linha]. Disponível em: http://www.who.int/mediacentre/factsheets/fs310/en/. [Acedido: 25-Fev-2017].

[3] «WHO | Road traffic injuries», *WHO*. [Em linha]. Disponível em: http://www.who.int/mediacentre/factsheets/fs358/en/. [Acedido: 24-Fev-2017].

[4] World Health Organization, *Global status report on road safety 2015*. 2015.

[5] World Health Organization, «Global Health Estimates 2015: Deaths by Cause, Age, Sex, by Country and by Region, 2000-2015». 2016.

[6] S. h An, B. H. Lee, e D. R. Shin, «A Survey of Intelligent Transportation Systems», em *2011 Third International Conference on Computational Intelligence, Communication Systems and Networks*, 2011, pp. 332–337.

[7] O. Berder *et al.*, «Cooperative communications between vehicles and intelligent road signs», em *2008 8th International Conference on ITS Telecommunications*, 2008, pp. 121–126.

[8] «Safespot». [Em linha]. Disponível em: http://www.safespot-eu.org. [Acedido: 31-Jan-2017].

[9] A. Spence, Y. Pavlis, T. Schendzielorz, e S. Zangherati, «Final Report: Needs and Requirements of Infrastructure-based Sensing – Part A», Nov. 2006.

[10] G. Toulminet, J. Boussuge, e C. Laurgeau, «Comparative synthesis of the 3 main European projects dealing with Cooperative Systems (CVIS, SAFESPOT and COOPERS) and description of COOPERS Demonstration Site 4», em *Intelligent Transportation Systems, 2008. ITSC 2008. 11th International IEEE Conference on*, 2008, pp. 809–814.

[11] T. Schendzielorz e F. Bonnefoi, «Application Scenarios and System Requirements», Jun. 2007.

[12] «EAR-IT: Using sound to picture the world in a new way», *Digital Single Market*. [Em linha]. Disponível em: https://ec.europa.eu/digital-single-market/en/news/ear-it-using-sound-picture-world-new-way. [Acedido: 31-Jan-2017].

[13] D. Hollosi, G. Nagy, R. Rodigast, S. Goetze, e P. Cousin, «Enhancing Wireless Sensor Networks with Acoustic Sensing Technology: Use Cases, Applications amp; Experiments», em *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, 2013, pp. 335–342.

[14] B. Kelly, D. Hollosi, P. Cousin, S. Leal, B. Iglár, e A. Cavallaro, «Application of Acoustic Sensing Technology for Improving Building Energy Efficiency», *Procedia Comput. Sci.*, vol. 32, pp. 661–664, 2014.

[15] «Cooperative Vehicle-Infrastructure Systems», *TRIP*, 21-Mai-2015. [Em linha]. Disponível em: http://www.transport-research.info/project/cooperative-vehicle-infrastructure-systems. [Acedido: 31-Jan-2017].

[16] P. Kompfner, «Final Activity Report Period: 01/02/2006 to 30/06/2010», Ago. 2010.

[17] J. Piao, M. McDonald, e N. Hounsell, «Cooperative vehicle-infrastructure systems for improving driver information services: an analysis of COOPERS test results», *IET Intell. Transp. Syst.*, vol. 6, n. 1, p. 9, 2012.

[18] P. Bosc, O. Sentieys, F. Peyret, C. Ray, J. m Bonnin, e Y. m L. Roux, «GIS ITS Bretagne: status and perspectives», em *2006 6th International Conference on ITS Telecommunications*, 2006, pp. 898–900.

[19] T. D. Nguyen, O. Berder, e O. Sentieys, «Cooperative strategies comparison for infrastructure and vehicle communications in CAPTIV», em *2009 9th International Conference on Intelligent Transport Systems Telecommunications, (ITST)*, 2009, pp. 420–424.

[20] «ITS-SIG». [Em linha]. Disponível em: http://gis-its.ifsttar.fr/index-en.php. [Acedido: 08-Fev-2017].

[21] G. Karagiannis *et al.*, «Vehicular Networking: A Survey and Tutorial on Requirements, Architectures, Challenges, Standards and Solutions», *IEEE Commun. Surv. Tutor.*, vol. 13, n. 4, pp. 584–616, Fourth 2011.

[22] F. Viani, A. Polo, E. Giarola, F. Robol, G. Benedetti, e S. Zanetti, «Performance assessment of a smart road management system for the wireless detection of wildlife road-crossing», em *2016 IEEE International Smart Cities Conference (ISC2)*, 2016, pp. 1–6.

[23] K. Dar, M. Bakhouya, J. Gaber, M. Wack, e P. Lorenz, «Wireless communication technologies for ITS applications [Topics in Automotive Networking]», *IEEE Commun. Mag.*, vol. 48, n. 5, pp. 156–162, 2010.

[24] «Smartway - FOT-Net WIKI». [Em linha]. Disponível em: http://wiki.fot-net.eu/index.php/Smartway. [Acedido: 15-Mar-2018].

[25] E. Ngai, F. Dressler, V. Leung, e M. Li, «Guest Editorial Special Section on Internet-of-Things for Smart Cities and Urban Informatics», *IEEE Trans. Ind. Inform.*, vol. 13, n. 2, pp. 748–750, Abr. 2017.

[26] L. L. Bello, M. Behnam, P. Pedreiras, e T. Sauter, «Guest Editorial Special Section on Communications in Automation #8211;Innovation Drivers and New Trends», *IEEE Trans. Ind. Inform.*, vol. 13, n. 2, pp. 841–845, Abr. 2017.

[27] S. Lee e A. Lim, «An Empirical Study on Ad Hoc Performance of DSRC and Wi-Fi Vehicular Communications», *Int. J. Distrib. Sens. Netw.*, vol. 9, n. 11, p. 482695, Nov. 2013.

[28] S. Mubeen, J. Mäki-Turja, e M. Sjödin, «Communications-oriented development of component-based vehicular distributed real-time embedded systems», *J. Syst. Archit.*, vol. 60, n. 2, pp. 207–220, Fev. 2014.

[29] A. Bhawiyuga, R. A. Sabriansyah, W. Yahya, e R. E. Putra, «A Wi-Fi based Electronic Road Sign for Enhancing the Awareness of Vehicle Driver», *J. Phys. Conf. Ser.*, vol. 801, p. 012085, Jan. 2017.

[30] «Instrução Técnica sobre a utilização da Sinalização de Mensagem Variável», Jul-2010. [Em linha]. Disponível em: http://www.imt-ip.pt/sites/IMTT/Portugues/InfraestruturasRodoviarias/InovacaoNormalizacao/Divulgao%20Tcnica/InstrucaoTecnicaUtilizacaoSinalizacaoMensagemVariavel.pdf. [Acedido: 18-Out-2017].

[31] R. M. Dias e T. D. Dias, «Melhoria dos Níveis de Serviço através da Monitorização de Tráfego», apresentado na 7º Congresso Rodoviário Português, 2013.

[32] A. Amador *et al.*, «Traffic Management Systems», *Intell. Transp. Syst. Technol. Appl.*, p. 249, 2015.

[33] T. Murata, «Petri nets: Properties, analysis and applications», *Proc. IEEE*, vol. 77, n. 4, pp. 541–580, 1989.

[34] R. Valette, «Petri nets for control and monitoring: specification, verification, implementation», em *Workshop on Analysis and and Design of Event-Driven Operations in Process Systems (ADEDOPS), Imperial College, London*, 1995, pp. 10–11.

[35] G. Frey e M. Minas, «Editing, visualizing, and implementing signal interpreted petri nets», em *Proceedings of the AWPN 2000*, 2000, pp. 57–62.

[36] L. Gomes, F. Moutinho, F. Pereira, J. Ribeiro, A. Costa, e J. P. Barros, «Extending input-output place-transition Petri nets for distributed controller systems development», em *2014 International Conference on Mechatronics and Control (ICMC)*, 2014, pp. 1099–1104.

[37] F. Pereira, F. Moutinho, e L. Gomes, «IOPT Tools User Manual», 2014. [Em linha]. Disponível em: http://gres.uninova.pt/iopt_usermanual.pdf. [Acedido: 28-Dez-2017].

[38] F. Pereira, F. Moutinho, e L. Gomes, «IOPT-tools—Towards cloud design automation of digital controllers with Petri nets», em *Mechatronics and Control (ICMC), 2014 International Conference on*, 2014, pp. 2414–2419.

[39] F. Pereira, F. Moutinho, J. Ribeiro, e L. Gomes, «Web based IOPT Petri net Editor with an extensible plugin architecture to support generic net operations», em *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, 2012, pp. 6151–6156.

[40] F. Pereira e L. Gomes, «Cloud Based IOPT Petri Net Simulator to Test and Debug Embedded System Controllers», em *Technological Innovation for Cloud-Based Engineering Systems*, 2015, pp. 165–175.

[41] F. Pereira, F. Moutinho, e L. Gomes, «Model-checking framework for embedded systems controllers development using IOPT Petri nets», em *2012 IEEE International Symposium on Industrial Electronics*, 2012, pp. 1399–1404.

[42] R. Campos-Rebelo, F. Pereira, F. Moutinho, e L. Gomes, «From IOPT Petri nets to C: An automatic code generator tool», em *2011 9th IEEE International Conference on Industrial Informatics*, 2011, pp. 390–395.

[43] F. Pereira e L. Gomes, «Automatic synthesis of VHDL hardware components from IOPT Petri net models», em *IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society*, 2013, pp. 2214–2219.

[44] F. Pereira, A. Melo, e L. Gomes, «Remote operation of embedded controllers designed using IOPT Petri-nets», 2015, pp. 572–579.

[45] R. Feio, J. Rosas, e L. Gomes, «Translating IOPT Petri net models into PLC ladder diagrams», em *2017 IEEE International Conference on Industrial Technology (ICIT)*, 2017, pp. 1211–1216.

[46] «Raspberry Pi 3 Model B», *Raspberry Pi*. .

[47] «Introducing PIXEL», *Raspberry Pi*, 28-Set-2016. .

[48] «SMC». [Em linha]. Disponível em: http://www.smc.com/en-global/products/product/45/0. [Acedido: 10-Ago-2017].

[49] «Clix - Área de Cliente :: Apoio Técnico». [Em linha]. Disponível em: http://cliente.clix.pt/apoiotecnico/adsl/penwireless/Dongle/especificacao/especificacaodongle.html. [Acedido: 10-Ago-2017].

[50] «Wireless G USB Network Adapter F5D7050 Version 3000 - Product Manual», 2004. .

[51] «hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator». [Em linha]. Disponível em: http://w1.fi/hostapd/. [Acedido: 18-Mar-2018].

[52] «hostapd Linux documentation page». [Em linha]. Disponível em: https://wireless.wiki.kernel.org/en/users/documentation/hostapd#wireless_interface. [Acedido: 18-Mar-2018].

[53] «DHCP_Server - Debian Wiki». [Em linha]. Disponível em: https://wiki.debian.org/DHCP_Server. [Acedido: 18-Mar-2018].
[54] «Wiki - batman-adv - Open Mesh». [Em linha]. Disponível em: https://www.open-mesh.org/projects/batman-adv/wiki/Wiki. [Acedido: 19-Fev-2018].
[55] «The Python Standard Library — Python 3.6.4 documentation». [Em linha]. Disponível em: https://docs.python.org/3/library/index.html. [Acedido: 19-Fev-2018].

Anexos

A. Sistema de Comunicação do Sinal de Trânsito - Código

data_handler.py

```
#!/usr/bin/env python
"""Deals with the sign information"""
import json

#-----------------------------------------------------------------------
def sign_info():
    """ Returns the information the signs conveys"""
    json_str = ''
    with open('sign_info.json') as info:
        json_data = json.load(info)
    json_str = json.dumps(json_data, indent=4, sort_keys=True)
    return json_str
#end sign_info

#-----------------------------------------------------------------------
def all_signs_info():
    """Returns the info of all signs that communicated with the sing and the sign's
own info"""
    json_str = sign_info()
    json_str += '\r\n'
    with open('roadsigns_info.txt') as client_info:
        for line in client_info:
            json_data = json.loads(line)
            json_str += json.dumps(json_data, indent=4, sort_keys=True)
            json_str += '\r\n'
    return json_str
```

```
#end_all_signs_info

#-----------------------------------------------------------------------
def average_velocity():
    """Allows to calculate the average speed all cars that sent information to the
sign"""
    total = 0
    car_number = 0
    with open('car_info.txt', 'r') as client_data:
        for line in client_data:
            if 'vel' in line:
                json_data = json.loads(line)
                if isinstance(json_data['vel'], list):
                    total += int(json_data['vel'][0])
                else:
                    total += int(json_data['vel'])
                car_number += 1
    if car_number != 0:
        average = '{0:.2f}'.format(total/car_number)
    else:
        return "There's no car data available"
    return average
#end_average_velocity

#-----------------------------------------------------------------------
def update_sign_data(all_data, updated_info, line):
    """Updates the information of a sign"""
    all_data[line] = json.dumps(updated_info, sort_keys=True) + '\n'
    with open('roadsigns_info.txt', 'w') as rs_info:
        rs_info.writelines(all_data)
#end_update_sign_data

#-----------------------------------------------------------------------
def add_info(file_name, new_data):
    """Adds new information to a file"""
    with open(file_name, 'a') as info_file:
        json.dump(new_data, info_file, sort_keys=True)
        info_file.write('\n')
#end_add_info

#-----------------------------------------------------------------------
def update_data_files(json_data, comm_type):
    """
    update files with car or sign info:
        if the data received comes from a sign:
            - check if there is already data from that sign
                - if it exists, delete it
            - add updated data to file
        if the data received comes from a car:
            - add new data to file
    """
    #if it's info from a sign
    found_data = False
    if "I2I" in comm_type:
        with open('roadsigns_info.txt', 'r') as rs_info:
```

```
            lines = rs_info.readlines()
        if not lines:
            add_info('roadsigns_info.txt', json_data)
        else:
            for line in lines:
                #if id of the sign in file = id new sign
                if line.split()[1].split('"')[1] == json_data['id']:
                    found_data = True
                    line_index = lines.index(line)
                    #update_sign_data(lines, json_data, lines.index(line))
                    break
                else:
                    #add_info('roadsigns_info.txt', json_data)
                    continue
            if found_data:
                update_sign_data(lines, json_data, line_index)
            else:
                add_info('roadsigns_info.txt', json_data)
    else:
        add_info('car_info.txt', json_data)
#end_update_data_files
```

logger.py

```
#!/usr/bin/env python
"""Logger module"""
def add_entry(data):
    """Adds a new entry to the log file"""
    with open('log.txt', 'a') as log_file:
        log_file.write(data)

#----------------------------------------------------------------------
def load_log():
    """Loads the content of the log file"""
    with open('log.txt', 'r') as log_file:
        log = log_file.read()
    if not log:
        return "There's no data to display"
    return log

#----------------------------------------------------------------------
def last_n_entries(nth):
    """Loads the last n lines of the log file
    Arguments:

    nth (int): number of lines to be loaded"""
    with open('log.txt', 'r') as log_file:
        text = log_file.readlines()
    if text:
        lines = len(text)
        if lines > nth:
            log = ''.join(text[lines-nth:])
        else:
```

```
            log = ''.join(text)
    else:
        return "There's no data to display"
    return log
```

http_request.py

```
#!/usr/bin/env python
"""HTTP Request Class"""

class HTTPRequest:
    """Creates a class that implements HTTP POST and GET requests"""
    def __init__(self, req_type, host_address, url, data):
        self.req_type = req_type
        self.host = host_address[0]
        self.port = host_address[1]
        self.url = url
        self.data = data

    def get(self):
        """creates a GET request"""
        request = 'GET ' + self.url + ' HTTP/1.1\r\n'
        request += 'Host: ' + self.host + ':' + str(self.port) + '\r\n'
        request += 'Connection: close\r\n'
        request += 'Accept:
text/html,text/plain,application/json,application/xml\r\n'
        request += '\r\n'
        return request

    def post(self):
        """creates a POST request with a JSON body"""
        request = None
        if '?' in self.url:
            raise ValueError('Invalid url')
        else:
            request = 'POST ' + self.url + ' HTTP/1.1\r\n'
            request += 'Host: ' + self.host + ':' + str(self.port) + '\r\n'
            request += 'Content-Type: application/json\r\n'
            request += 'Content-Length: ' + str(len(self.data)) + '\r\n'
            request += 'Connection: close\r\n'
            request += 'Accept:
text/html,text/plain,application/json,application/xml\r\n'
            request += '\r\n'
            request += self.data
            request += '\r\n'
        return request

    def create_request(self):
        """creates a request with the selected method"""
        request = None

        if self.req_type == 'GET':
            request = self.get()
        elif self.req_type == 'POST':
```

```
            request = self.post()
        else:
            raise ValueError('Method not supported')
        return request
```

http_server.py

```
#!/usr/bin/env python
"""HTTP Server Class"""

import xml.etree.ElementTree as ET
import datetime as dt
from urllib import parse
import json
import re
import logger
import data_handler
#-----------------------------------------------------------
HTTP_OK_CODE = 200
HTTP_BAD_REQ_CODE = 400
HTTP_NOT_FOUND_CODE = 404

REASON_PHRASE = {
    HTTP_OK_CODE: 'OK',
    HTTP_BAD_REQ_CODE: 'Bad Request',
    HTTP_NOT_FOUND_CODE: 'Not Found'
}

MAINT_ON = 'Maint. mode: 1\n'
MAINT_OFF = 'Maint. mode: 0\n'
#-----------------------------------------------------------
class HTTPServer:
    """
    Http server class
    creates responses to clients' requests and updates log file
    """
    def __init__(self, client_addr, req_msg, mode):
        self.client_addr = client_addr
        self.req_msg = req_msg
        self.mode = mode

    #---------------------------------------------------------------------
    @classmethod
    def response_header(cls, code, content_type):
        """Creates the response header"""
        header = 'HTTP/1.1 ' + str(code) + ' ' + REASON_PHRASE[code] + '\r\n'
        header += 'Connection: close\r\n'
        header += 'Content-Type: ' + content_type + '\r\n'
        header += '\r\n'
        return header

    #---------------------------------------------------------------------
    def not_found(self):
        """Creates response for page/file not found"""
        #--------------------HTML PAGE--------------------
```

```
        html_page = ET.Element("html")
        head = ET.SubElement(html_page, "head")
        title = ET.SubElement(head, "title")
        title.text = "404 Not Found"
        body = ET.SubElement(html_page, "body")
        header = ET.SubElement(body, "h1")
        header.text = "Not Found"
        parag = ET.SubElement(body, "p")
        parag.text = "The server couldn't find the requested url."
        #-------------------------------------------------
        #----------------SERVER RESPONSE------------------
        #HEADER
        response = self.response_header(HTTP_NOT_FOUND_CODE, 'text/html')
        #BODY
        response += "<!DOCTYPE HTML>"
        response += ET.tostring(html_page, encoding='unicode')
        #-------------------------------------------------
        return response

    #----------------------------------------------------------------------
    def bad_request(self):
        """Creates a response for bad requests"""
        #--------------------HTML PAGE--------------------
        html_page = ET.Element("html")
        head = ET.SubElement(html_page, "head")
        title = ET.SubElement(head, "title")
        title.text = "400 Bad Request"
        body = ET.SubElement(html_page, "body")
        header = ET.SubElement(body, "h1")
        header.text = "Bad Request"
        parag = ET.SubElement(body, "pre")
        parag.text = "The server couldn't understand the request.\r\n"
        parag.text = "Supported Requests: GET and POST\r\n"
        #-------------------------------------------------
        #----------------SERVER RESPONSE------------------
        #HEADER
        response = self.response_header(HTTP_BAD_REQ_CODE, 'text/html')
        #BODY
        response += "<!DOCTYPE HTML>"
        response += ET.tostring(html_page, encoding='unicode')
        #-------------------------------------------------
        return response

    #----------------------------------------------------------------------
    def info_page_i2i(self):
        """Creates response with the road sign information"""
        #-------------------JSON MESSAGE-------------------
        json_str = data_handler.sign_info()
        #-------------------------------------------------
        #----------------SERVER RESPONSE------------------
        #HEADER
        response = self.response_header(HTTP_OK_CODE, 'application/json')
        #BODY
        response += json_str
        #-------------------------------------------------
```

```python
        return response

    #----------------------------------------------------------------------
    def info_page_i2v(self):
        """Creates response with information from all the signs"""
        #-------------------JSON MESSAGE-------------------
        json_str = data_handler.all_signs_info()
        #-------------------------------------------------
        #---------------SERVER RESPONSE------------------
        #HEADER
        response = self.response_header(HTTP_OK_CODE, 'application/json')
        #BODY
        response += json_str
        #-------------------------------------------------
        return response

    #----------------------------------------------------------------------
    def log_page(self):
        """Creates a response with the log info"""
        #---------------SERVER RESPONSE------------------
        #HEADER
        response = self.response_header(HTTP_OK_CODE, 'text/plain')
        #BODY
        response += logger.load_log()
        #-------------------------------------------------
        return response

    #----------------------------------------------------------------------
    def log_n_page(self, numbr):
        """Creates a response with the log's last n entries

        Arguments:

        numbr (int): number of entries to display
        """
        #---------------SERVER RESPONSE------------------
        #HEADER
        response = self.response_header(HTTP_OK_CODE, 'text/plain')
        #BODY
        response += logger.last_n_entries(numbr)
        #-------------------------------------------------
        return response

    #----------------------------------------------------------------------
    def sign_in_maintenance(self):
        """Creates a response with the message 'Road sign in maintenance'"""
        #---------------SERVER RESPONSE------------------
        #HEADER11
        response = self.response_header(HTTP_OK_CODE, 'text/plain')
        #BODY
        response += 'Road sign in maintenance'
        #-------------------------------------------------
        return response

    #----------------------------------------------------------------------
```

```
    def handle_req_page(self, req_page, comm_type):
        """Creates a response with the requested page, if available"""
        if req_page == '/info':
            if "I2V" in comm_type:
                response = self.info_page_i2v()
            else:
                response = self.info_page_i2i()
        elif req_page == '/log':
            response = self.log_page()
        elif req_page == '/log10':
            response = self.log_n_page(10)
        elif req_page == '/log20':
            response = self.log_n_page(20)
        else:
            response = self.not_found()
        return response

    #----------------------------------------------------------------------
    def handle_request(self, comm_type):
        '''
        Creates a response to the client
        Updates the data files and creates a log entry

        Arguments:

        comm_type(str): type of comnunication (i2v or i2i)

        Returns:

        response(str): servers's response to the request
        log_entry(str): entry to be added to log file
        '''
        json_data = {}
        date = dt.datetime.now().strftime('[%a, %d %B %Y %H:%M:%S]') #define date
format
        req_line = self.req_msg.split('\r\n')[0] #get request's first line
        (method, url, _) = req_line.split()

        if self.mode == MAINT_OFF: #if the sign is not in maintenance
            if method != 'GET' and method != 'POST':
                response = self.bad_request()
                return response

            if '?' in url:
                (req_page, query_str) = url.split('?')
                json_data = parse.parse_qs(query_str)
            else:
                req_page = url

            if method == 'POST':
                query_str = self.req_msg.split('\r\n\r\n')[1]
                if query_str:
                    json_data = json.loads(query_str)

            response = self.handle_req_page(req_page, comm_type)
```

```
            if json_data:
                data_handler.update_data_files(json_data, comm_type)
        else:
            response = self.sign_in_maintenance()

        resp_code = response.split('\r\n')[0].split()[1] #Get code from status line
        log_entry = (date + ' ' + self.client_addr[0] + ':' +
str(self.client_addr[1])
                     + ' ' + method + ' ' + url + ' ' + resp_code + "\n")
        return response, log_entry
```

cp_client.py

```
#!/usr/bin/env python
"""Implements the TCP/IP client class"""
import socket
import threading
import time
import json
import errno
from http_request import HTTPRequest
import data_handler
import named_pipe
#------------------------------------------------------------------
BUFFER_SIZE = 1024
SLEEP_TIME = 1 #minutes
REQ_TIMEOUT = 100 #seconds
MAINT_ON = 'Maint. mode: 1\n'
MAINT_OFF = 'Maint. mode: 0\n'
#------------------------------------------------------------------

class Client(threading.Thread):
    """TCP client capable of making POST and GET requests
    and displaying the result in the console"""
    def __init__(self, q):
        threading.Thread.__init__(self)
        self.running = 1
        self.peer_list = [] #list of strings with the format: "ssid pwd ip port"
        self.queue = q
        self.soc = None
        named_pipe.write("Client initialized")
    #end __init__

    @classmethod
    def recv_data(cls, sock, timeout=1):
        """Receives data from server

        Returns the received message"""

        sock.setblocking(0) #non blocking mode
        msg = ''
        start = time.time()
```

```
        while True:
            if msg and time.time() - start > timeout:
                break
            elif time.time() - start > 2 * timeout:
                break
            try:
                data = sock.recv(BUFFER_SIZE).decode('utf-8')
                if data:
                    msg += data
                    start = time.time()
                else:
                    time.sleep(0.1)
            except socket.error as err:
                if err.args[0] == errno.EWOULDBLOCK:
                    pass
        #end while
        return msg
    #end recv_data

    def load_peer_info(self):
        """Loads the information about the servers"""
        with open('peers.txt', 'r') as peerfile:
            self.peer_list = peerfile.readlines()
    #end load_peer_info

    def request_info(self, connected, peerip, peerport):
        """Makes requests to the server until it has a response or
        reaches a timeout"""

        received_data = False
        if not connected:
            try:
                self.soc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                self.soc.connect((peerip, peerport))
                print('>> Connected to ' + peerip + ':' + str(peerport))
                connected = True
            except socket.error:
                pass
        else:
            #load sign_info
            json_info = data_handler.sign_info()
            #create request
            request = HTTPRequest('POST', [peerip, peerport], '/info',
                                  json_info).create_request()
            if not request:
                print('Method not supported')
                self.soc.close()
                connected = False

            print('>> Requested: \r\n', request.split('\r\n')[0])
            self.soc.send(request.encode('utf-8'))
            response = self.recv_data(self.soc)
            if response:
                if 'Road sign in maintenance' not in response:
                    print('>> Server response: \r\n', response)
```

```
                    json_data = json.loads(response.split('\r\n\r\n')[1])
                    data_handler.update_data_files(json_data, "I2I_")
                received_data = True
            else:
                print('>> No server response')
            self.soc.close()
            connected = False
        return connected, received_data
    #end request_info

    @classmethod
    def update_mode(cls, old_mode, queue):
        """Checks if there are changes in the sign state

        Returns mode (normal or monitor)"""

        try:
            new_mode = queue.get(block=False)
            print('MODE:', new_mode)
            if new_mode:
                mode = new_mode
            else:
                return old_mode
        except queue.Empty:
            pass
        return mode
    #end_update_mode

    def run(self):
        connected = False
        self.load_peer_info()
        mode = MAINT_OFF

        while self.running == 1:

            for peer in self.peer_list:
                mode = self.update_mode(mode, self.queue)

                if  mode == MAINT_OFF:
                    peerip, peerport = peer.split()
                    named_pipe.write('Server IP available')
                    print('>> Making request to server')
                    #try to get sign's info
                    start_time = time.time()
                    named_pipe.write('Making new request')
                    while True:
                        if time.time() - start_time < REQ_TIMEOUT:
                            (connected, received_data) = self.request_info(connected,
peerip,

int(peerport))
                        else:
                            named_pipe.write('Reached response timeout')
                            break
```

```
                        if received_data:
                            named_pipe.write('Response available')
                            break
                    #end while
                    named_pipe.write('End try request')
                #end if
            #end for
            time.sleep(60*SLEEP_TIME)
            named_pipe.write('Timer finished')
        #end while
    #end run

    def kill(self):
        """Stops thread"""
        self.running = 0
```

tcp_server.py

```
#!/usr/bin/env python
"""Implements the TCP/IP server class"""
import socket
import errno
import threading
import select
import time
import queue
import logger
import named_pipe
from http_server import HTTPServer
#------------------------------------------------------------------
BUFFER_SIZE = 1024
TIMEOUT = 60
MAINT_ON = 'Maint. mode: 1\n'
MAINT_OFF= 'Maint. mode: 0\n'
#------------------------------------------------------------------

class Server(threading.Thread):
    '''docstring for Server'''
    def __init__(self, serverhost='', serverport=None, prefix='', q=None):
        threading.Thread.__init__(self)
        self.running = 1
        self.serverhost = serverhost
        self.serverport = serverport
        self.prefix = prefix
        self.server_socket = None
        self.queue = q
        named_pipe.write(prefix + "Server started")
    #end __init__

    def setup_server(self):
        '''Creates server socket, binds it to the server port and waits for
connections
```

```
    Returns the created socket
    '''
    tcp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcp_socket.setblocking(0) # make socket non blocking
    #allows ip addresses to be reused before socket timeout
    tcp_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    print('>> Socket created.')
    try:
        tcp_socket.bind((self.serverhost, self.serverport))
        tcp_socket.listen(5)
    except socket.error as msg:
        raise msg
    print('>> Bind completed.')
    print('>> Listening for clients')
    return tcp_socket
#end setup_Server

@classmethod
def handle_connection(cls, tcp_socket):
    '''Returns the connection and the client's address'''
    try:
        conn, address = tcp_socket.accept()
    except socket.error as ex:
        raise ex
    print('>> Connected to: ' + address[0] + ':' + str(address[1]))
    return conn, address
#end handle_connection

@classmethod
def recv_data(cls, sock, timeout=1):
    """Reads data from socket"""
    sock.setblocking(0) #non blocking mode
    msg = ''
    start = time.time()

    while True:
        if msg and time.time() - start > timeout:
            break
        elif time.time() - start > 2*timeout:
            break
        try:
            data = sock.recv(BUFFER_SIZE).decode('utf-8')
            if data:
                msg += data
                start = time.time()
            else:
                time.sleep(0.1)
        except socket.error as err:
            if err.args[0] == errno.EWOULDBLOCK:
                pass
    #end while
    return msg
#end recv_data

def handle_client(self, clientsock, clientaddr, mode):
```

```
        '''
        Receives the client's request, returns a response
        If no data was received the function returns None
        '''
        recvdata = self.recv_data(clientsock)
        if not recvdata:
            print('>> No data received')
            return None
        print('>> Request from: ' + clientaddr[0] +':' + str(clientaddr[1]) +
              ' : ' + recvdata.split('\r\n')[0])
        named_pipe.write(self.prefix + 'Received request')
        response, log_entry = HTTPServer(clientaddr, recvdata,
mode).handle_request(self.prefix)
        named_pipe.write(self.prefix + 'Response created')
        logger.add_entry(log_entry)
        named_pipe.write(self.prefix + 'Log updated')
        return response.encode('utf-8')
    #end handle_client

    def handle_new_conn(self, sock_read, clients_addrs):
        """Handles new connections by adding the new socket to the socket list
        and saves the address of the new client"""
        named_pipe.write(self.prefix + 'New connection')
        (conn, addr) = self.handle_connection(self.server_socket)
        sock_read.append(conn) #add new socket to list
        clients_addrs[conn] = addr
    #end_handle_new_conn

    def handle_client_disconn(self, sock, responses, sock_lists, clients_addrs):
        """Checks when client closed the connection to the server

        Removes the socket from all lists and closes the connection on the server
side"""

        #sock_read = sock_lists[0] and sock_write = sock_lists[1]

        if sock in responses:
            named_pipe.write(self.prefix + 'Disconnected bfr response')
            print('Disconnected before sending response')
        elif sock in sock_lists[0]:
            named_pipe.write(self.prefix + 'Disconnected bfr sending data')
            print('Disconnected after new conn')

        named_pipe.write(self.prefix + 'Client disconnected')
        print('>>Disconnnected from: '+ clients_addrs[sock][0] + ':' +
              str(clients_addrs[sock][1]) + '\r\n')
        del clients_addrs[sock]
        try:
            sock_lists[0].remove(sock)
            sock_lists[1].remove(sock)
        except ValueError:
            pass
        sock.close()
    #end_handle_client_disconn
```

```
    def send_response(self, sock, responses, sock_lists, clients_addrs):
        """Sends a response to the request made

        Closes the connection when there is no more data to send
        """
        if sock in sock_lists[1]: #sock_write
            datatosend = responses[sock]
            if datatosend:
                named_pipe.write(self.prefix + 'Sending response')
                datasize = len(datatosend)
                sent = sock.send(datatosend)
                datatosend = datatosend[sent:]
                remaining = datasize - sent
            if remaining > 0:
                responses[sock] = datatosend
            else:
                print('There is no more data to send to ' + clients_addrs[sock][0] +
':'
                      + str(clients_addrs[sock][1]))
                try:
                    named_pipe.write(self.prefix + 'Client disconnected')
                    del responses[sock]
                    sock_lists[0].remove(sock)
                    sock_lists[1].remove(sock)
                    sock.close()
                    print('>>Disconnnected from: '+ clients_addrs[sock][0] + ':'
                          + str(clients_addrs[sock][1]) + '\r\n')
                except (KeyError, ValueError):
                    pass
    #end_send_response
    @classmethod
    def update_mode(cls, old_mode, _queue):
        """Checks if there are changes in the sign state

        Returns mode (normal or monitor)"""
        mode = old_mode
        try:
            new_mode = _queue.get(block=False)
            print('MODE:', new_mode)
            if new_mode:
                mode = new_mode
        except queue.Empty:
            pass
        return mode
    #end_update_mode

    def run(self):
        self.server_socket = self.setup_server()
        sock_read = [self.server_socket] #list of sockets ready to be read
        sock_write = [] #list of sockets ready to be written to
        clients_addrs = {} #dict with socket : address
        responses = {} #data do send : socket
        mode = MAINT_OFF

        try:
```

```
            while self.running == 1:
                mode = self.update_mode(mode, self.queue)

                readytoread, readytowrite, _ = select.select(sock_read, sock_write,
[], TIMEOUT)

                for sock in readytoread:
                    if sock is self.server_socket: #if theres a new connection
                        self.handle_new_conn(sock_read, clients_addrs)
                    else:
                    #there's new data to receive or client disconnected
                        response = self.handle_client(sock, clients_addrs[sock],
mode)
                        #if client disconnected
                        if not response:
                            #check when client disconnected
                            self.handle_client_disconn(sock, responses, [sock_read,
sock_write],
                                                       clients_addrs)
                        else:
                            responses[sock] = response
                            if sock not in sock_write:
                                sock_write.append(sock)

                for sock in readytowrite:
                    #send responses to available clients
                    self.send_response(sock, responses, [sock_read, sock_write],
clients_addrs)
            self.server_socket.close()
        except Exception as ex:
            raise ex

    def kill(self):
        """Stops thread"""
        self.running = 0
```

named_pipe.py

```
#!/usr/bin/env python
"""Named Pipe module"""
import os
import errno
import time
import select

WR_FIFO = '/tmp/pytoc'
RD_FIFO = '/tmp/ctopy'
FD_IN = None
FD_OUT = None

#------------------------------------------------------------
def create_pipe():
```

```
    """Creates two FIFOs for bidirectional communication between processes and opens
them"""
    try:
        os.mkfifo(WR_FIFO)
        os.mkfifo(RD_FIFO)
    except OSError as oe:
        if oe.errno != errno.EEXIST:
            raise oe

    global FD_IN
    #os.open(myfifo, os.O_WRONLY)
    FD_IN = os.open(WR_FIFO, os.O_RDWR)

    global FD_OUT
    FD_OUT = os.open(RD_FIFO, os.O_RDWR)
    print('File descriptor:', FD_OUT)

#------------------------------------------------------------
def write(message):
    """Writes message in WR_FIFO
    Arguments:

    message(str): data to send

    """
    global FD_IN
    if FD_IN is not None:
        try:
            message = message+'\n'
            time.sleep(0.01)
            os.write(FD_IN, message.encode())
            print("::::", message)
        except OSError as ose:
            if ose != errno.EPIPE:
                raise ose

#------------------------------------------------------------
def read():
    """Reads message written in RD_FIFO"""
    global FD_OUT
    if FD_OUT is not None:
        fifo_read = [FD_OUT]
        timeout = 10
        data = b''

        readytoread, _, _ = select.select(fifo_read, [], [], timeout)
        for fifo in readytoread:
            data = os.read(fifo, 15)
    return data.decode()

#------------------------------------------------------------
def close():
    """Closes the FIFOs and deletes them from the system"""
    os.close(WR_FIFO)
    os.remove(WR_FIFO)
```

```
    os.close(RD_FIFO)
    os.remove(RD_FIFO)
    print('FIFO closed')
```

road_sign_state.py

```
#!/usr/bin/env python
"""Module responsible for checking for changes in the sign state"""
import threading
import named_pipe

class RoadSignState(threading.Thread):
    """Creates a class with queues that send the actual sign state to the servers
    and client threads"""
    def __init__(self, queueI2I, queueI2V, queueClient):
        threading.Thread.__init__(self)
        self.queue_i2i = queueI2I
        self.queue_i2v = queueI2V
        self.queue_client = queueClient
        self.running = 1

    def run(self):
        while self.running == 1:
            data = named_pipe.read()
            if data:
                self.queue_i2i.put(data)
                self.queue_i2v.put(data)
                self.queue_client.put(data)

    def kill(self):
        """Stops thread"""
        self.running = 0
```

tcp_p2p.py

```
#!/usr/bin/env python
"""Main module"""
import queue
from road_sign_state import RoadSignState
from tcp_client import Client
from tcp_server import Server
import named_pipe
#-----------------------------------------------------------------
BUFFER_SIZE = 1024
#-----------------------------------------------------------------
Q_I2I = queue.Queue()
Q_I2V = queue.Queue()
Q_CLIENT = queue.Queue()
#-----------------------------------------------------------------
def main():
    """Starts all threads"""
    ap_serverhost = '192.168.42.1'
```

```
    #ap_serverhost = 'localhost'
    mesh_serverhost = '192.168.41.1'
    serverport = 12345
    try:
        named_pipe.create_pipe()
        rs_state = RoadSignState(Q_I2I, Q_I2V, Q_CLIENT)
        rs_state.start()
        ap_server = Server(ap_serverhost, serverport, 'I2V_', Q_I2V)
        ap_server.start()

        mesh_server = Server(mesh_serverhost, serverport, 'I2I_', Q_I2I)
        mesh_server.start()
        print('>> Starting servers...')

        client = Client(Q_CLIENT)
        client.start()
        print('>> Starting wifi client...')
    except Exception as ex:
        print('>> Error: turning off peer')
        named_pipe.write('Failed to start server')
        named_pipe.close()
        ap_server.kill()
        mesh_server.kill()
        client.kill()
        raise ex

if __name__ == '__main__':
    main()
```

B. Controlador do Sinal de Trânsito – Código

net_io.c

```
/* Net RoadSignPN - IOPT */
/* Automatic code generated by IOPT2C XSLT transformation. */
/* Please fill the necessary code to perform hardware IO. */


#include <stdlib.h>
#include "net_types.h"

/**********Added libraries**********/
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
/***********************************/

#ifdef ARDUINO
#include <Arduino.h>
#define ANALOG_IN_MAX  1023
#define ANALOG_OUT_MAX 255
#else
#define INPUT    0
#define OUTPUT   1
#define ANALOG_IN_MAX  1023
#define ANALOG_OUT_MAX 1023
extern void pinMode( int, int );
extern int  digitalRead( int );
extern void digitalWrite( int, int );
extern int  analogRead( int );
extern void analogWrite( int, int );
#endif


// Remote IcE/Debug forced values:
#ifdef HTTP_SERVER
iopt_param_info *input_fv = NULL, *output_fv = NULL;
#endif

/************************CONSTANTS************************/
#define MAX_BUF 1024
#define TIMEOUT 10
const char* PIPE_IN = "/tmp/pytoc";
const char* PIPE_OUT = "/tmp/ctopy";
const char* I2V_SERVER_ON = "I2V_Server started";
const char* I2V_NEW_CONN = "I2V_New connection";
const char* I2V_RECV_MSG = "I2V_Received request";
const char* I2V_CREAT_MSG = "I2V_Response created";
const char* I2V_UPDT_LOG = "I2V_Log updated";
const char* I2V_SEND_MSG = "I2V_Sending response";
const char* I2V_CLOSED_CONN = "I2V_Client disconnected";
```

```c
const char* I2V_CLOSED_BFR_RESP = "I2V_Disconnected bfr response";
const char* I2V_CLOSED_BFR_SEND = "I2V_Disconnected bfr sending data";

const char* I2I_SERVER_ON = "I2I_Server started";
const char* I2I_NEW_CONN = "I2I_New connection";
const char* I2I_RECV_MSG = "I2I_Received request";
const char* I2I_CREAT_MSG = "I2I_Response created";
const char* I2I_UPDT_LOG = "I2I_Log updated";
const char* I2I_SEND_MSG = "I2I_Sending response";
const char* I2I_CLOSED_CONN = "I2I_Client disconnected";
const char* I2I_CLOSED_BFR_RESP = "I2I_Disconnected bfr response";
const char* I2I_CLOSED_BFR_SEND = "I2I_Disconnected bfr sending data";

const char* CLIENT_ON = "Client initialized";
const char* SERVER_IP_AV = "Server IP available";
const char* NEW_REQUEST = "Making new request";
const char* RESPONSE_READY = "Response available";
const char* RESP_TIMEOUT = "Reached response timeout";
const char* END_REQUEST = "End try request";
const char* TIMER_FINISH = "Timer finished";
const char* BLOCK_ON = "Maint. mode: 1\n";
const char* BLOCK_OFF = "Maint. mode: 0\n";
/************************************************************/

/***********************GLOBAL VARS***********************/
char last_mode[15];
/************************************************************/
/***********************FUNCTIONS***********************/
void resetAllInputEvents(RoadSignPN_InputSignalEvents* events) {
    events->NewConnI2V = 0;
    events->RecvMsgI2V = 0;
    events->SendMsgI2V = 0;
    events->NewConnI2I = 0;
    events->RecvMsgI2I = 0;
    events->SendMsgI2I = 0;
    events->LogUpdatedI2V = 0;
    events->ClientDisconnI2V = 0;
    events->LogUpdatedI2I = 0;
    events->ClientDisconnI2I = 0;
    events->StartServerI2V = 0;
    events->StartServerI2I = 0;
    events->CreateMsgI2V = 0;
    events->CreateMsgI2I = 0;
    events->StartClient = 0;
    events->HasServerIP = 0;
    events->TryMakeRequest = 0;
    events->ReachedTimeout = 0;
    events->GetResponse = 0;
    events->EndTryMakeReq = 0;
    events->TimerFinished = 0;
}
/************************************************************/

/* Executed just once, before net execution starts: */
void RoadSignPN_InitializeIO()
```

```
{
    printf("Monitoring process...\n");
    strcpy(last_mode, BLOCK_OFF);
}



/* Read all hardware input signals and fill data-structure */
void RoadSignPN_GetInputSignals(
            RoadSignPN_InputSignals* inputs,
            RoadSignPN_InputSignalEvents* events )
{
    int fd;
    char buf[MAX_BUF];
    fd_set readfs;
    FILE *fp;
    struct timeval timeout = {0,TIMEOUT};//{sec, usec}

    if( events != NULL ) {
        resetAllInputEvents(events);
        //open FIFO (named pipe) in read only mode
        fp = fopen(PIPE_IN, "r");
        fd = fileno(fp);
        if (fp != NULL && fd != 0) {
            //clear set
            FD_ZERO(&readfs);
            //add file descriptor to set
            FD_SET(fd, &readfs);
            //check if there's input available
            select(fd+1, &readfs, NULL, NULL, &timeout);
            //if pipe has new input
            if(FD_ISSET(fd,&readfs)){
                // read FIFO
                if (fgets(buf, MAX_BUF, fp)!=NULL){
                    int size = strcspn(buf, "\n");
                    char message[size];

                    strncpy(message, buf, size);
                    message[size] = '\0';
                    //I2V message
                    if (message[2] == 'V')
                    {
                        if (strcmp(message, I2V_SERVER_ON) == 0)
                            events->StartServerI2V = 1;
                        else if (strcmp(message, I2V_NEW_CONN) == 0)
                            events->NewConnI2V = 1;
                        else if (strcmp(message, I2V_RECV_MSG) == 0)
                            events->RecvMsgI2V = 1;
                        else if (strcmp(message, I2V_CREAT_MSG) == 0)
                            events->CreateMsgI2V = 1;
                        else if (strcmp(message, I2V_UPDT_LOG) == 0)
                            events->LogUpdatedI2V = 1;
                        else if (strcmp(message, I2V_SEND_MSG) == 0)
                            events->SendMsgI2V = 1;
                        else if (strcmp(message, I2V_CLOSED_CONN) == 0)
```

```
                            events->ClientDisconnI2V = 1;
                        else if (strcmp(message, I2V_CLOSED_BFR_RESP) == 0)
                            events->DisconnBfrRespI2V = 1;
                        else if (strcmp(message, I2V_CLOSED_BFR_SEND) == 0)
                            events->DisconnBfrDataI2V = 1;
                    //I2I message
                    } else if (message[2] == 'I')
                    {
                        if (strcmp(message, I2I_SERVER_ON) == 0)
                            events->StartServerI2I = 1;
                        else if (strcmp(message, I2I_NEW_CONN) == 0)
                            events->NewConnI2I = 1;
                        else if (strcmp(message, I2I_RECV_MSG) == 0)
                            events->RecvMsgI2I = 1;
                        else if (strcmp(message, I2I_CREAT_MSG) == 0)
                            events->CreateMsgI2I = 1;
                        else if (strcmp(message, I2I_UPDT_LOG) == 0)
                            events->LogUpdatedI2I = 1;
                        else if (strcmp(message, I2I_SEND_MSG) == 0)
                            events->SendMsgI2I = 1;
                        else if (strcmp(message, I2I_CLOSED_CONN) == 0)
                            events->ClientDisconnI2I = 1;
                        else if (strcmp(message, I2I_CLOSED_BFR_RESP) == 0)
                            events->DisconnBfrRespI2I = 1;
                        else if (strcmp(message, I2I_CLOSED_BFR_SEND) == 0)
                            events->DisconnBfrDataI2I = 1;
                    //client message
                    } else
                    {
                        if (strcmp(message, CLIENT_ON) == 0)
                            events->StartClient = 1;
                        else if (strcmp(message, SERVER_IP_AV) == 0)
                            events->HasServerIP = 1;
                        else if (strcmp(message, NEW_REQUEST) == 0)
                            events->TryMakeRequest = 1;
                        else if (strcmp(message, RESPONSE_READY) == 0)
                            events->GetResponse = 1;
                        else if (strcmp(message, RESP_TIMEOUT) == 0)
                            events->ReachedTimeout = 1;
                        else if (strcmp(message, END_REQUEST) == 0)
                            events->EndTryMakeReq = 1;
                        else if (strcmp(message, TIMER_FINISH) == 0)
                            events->TimerFinished = 1;
                    }

                    printf("Message: %s\n", message);
                }
            }
            fclose(fp);
        } else {
            perror("PIPE_IN");
        }
    }

#ifdef HTTP_SERVER
```

```
    if( input_fv != NULL ) force_RoadSignPN_Inputs( input_fv, inputs );
#endif
}


/* Write all output values to physical hardware outputs */
void RoadSignPN_PutOutputSignals(
            RoadSignPN_PlaceOutputSignals* place_out,
            RoadSignPN_EventOutputSignals* event_out,
            RoadSignPN_OutputSignalEvents* events )
{
    if( events != NULL ) {
        int fd;

        fd = open(PIPE_OUT, O_WRONLY);
        if (fd > 0) {
            if (events->UnblockRSign || events->BlockRSign) {
                char message[15];
                if (events->UnblockRSign == 1 && strcmp(BLOCK_ON, last_mode) == 0) {
                    strcpy(message, BLOCK_OFF);
                    strcpy(last_mode, message);
                    write(fd, message, sizeof(message));
                }

                if (events->BlockRSign == 1 && strcmp(BLOCK_OFF, last_mode) == 0){
                    strcpy(message, BLOCK_ON);
                    strcpy(last_mode, message);
                    write(fd, message, sizeof(message));
                }
            }
            close(fd);
        } else {
            perror("PIPE_OUT");
        }

    }

#ifdef HTTP_SERVER
    if( output_fv != NULL )
        force_RoadSignPN_Outputs( output_fv, place_out, event_out );
#endif

}


/* Delay between loop iterations to save CPU and power consumption */
void RoadSignPN_LoopDelay()
{
}

/* Must return 1 to finish net execution */
int RoadSignPN_FinishExecution( RoadSignPN_NetMarking* marking )
{
    return 0;
}
```

C. Controlador do Sinal de Trânsito – Rede de Petri

Servidor para Veículos

Figura 23 – Modelo do servidor para comunicação entre sinais de trânsito e veículos

Servidor para Sinais de Trânsito

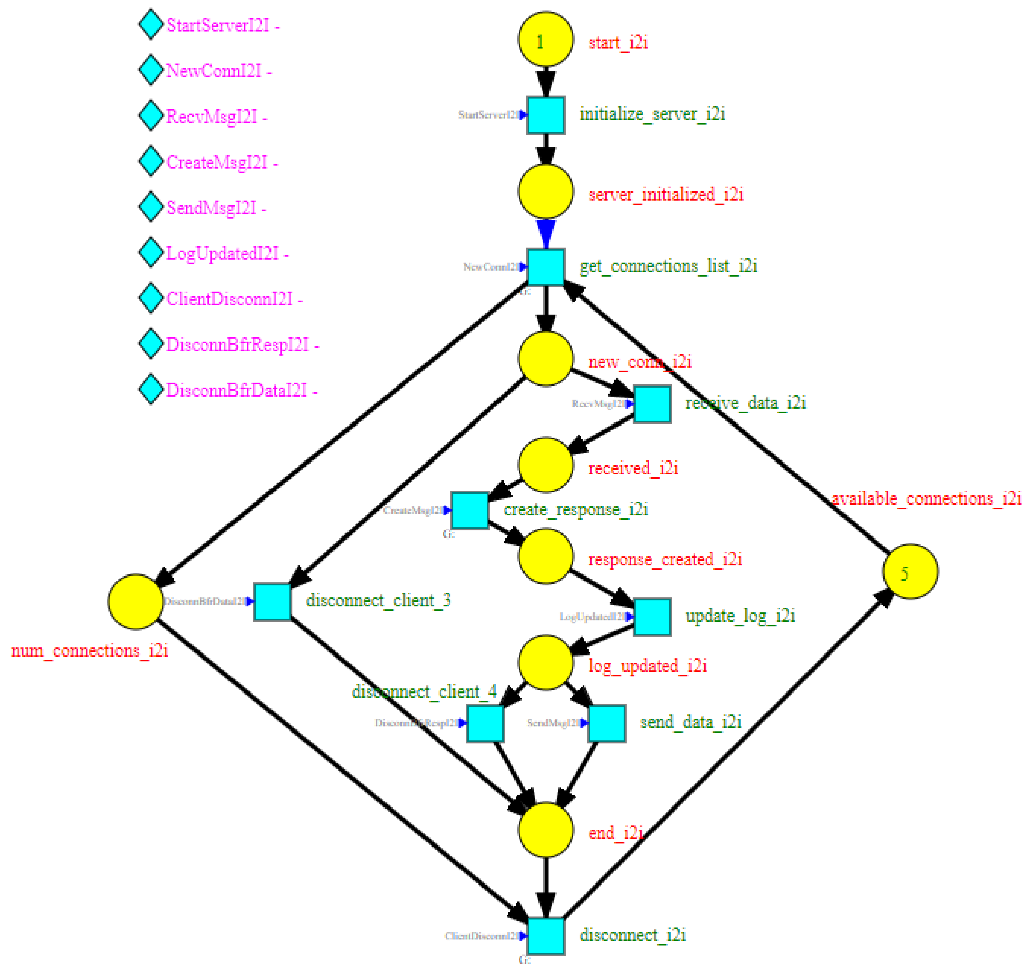


Figura 24 - Modelo do servidor para comunicação entre sinais de trânsito

Cliente para Sinais de Trânsito

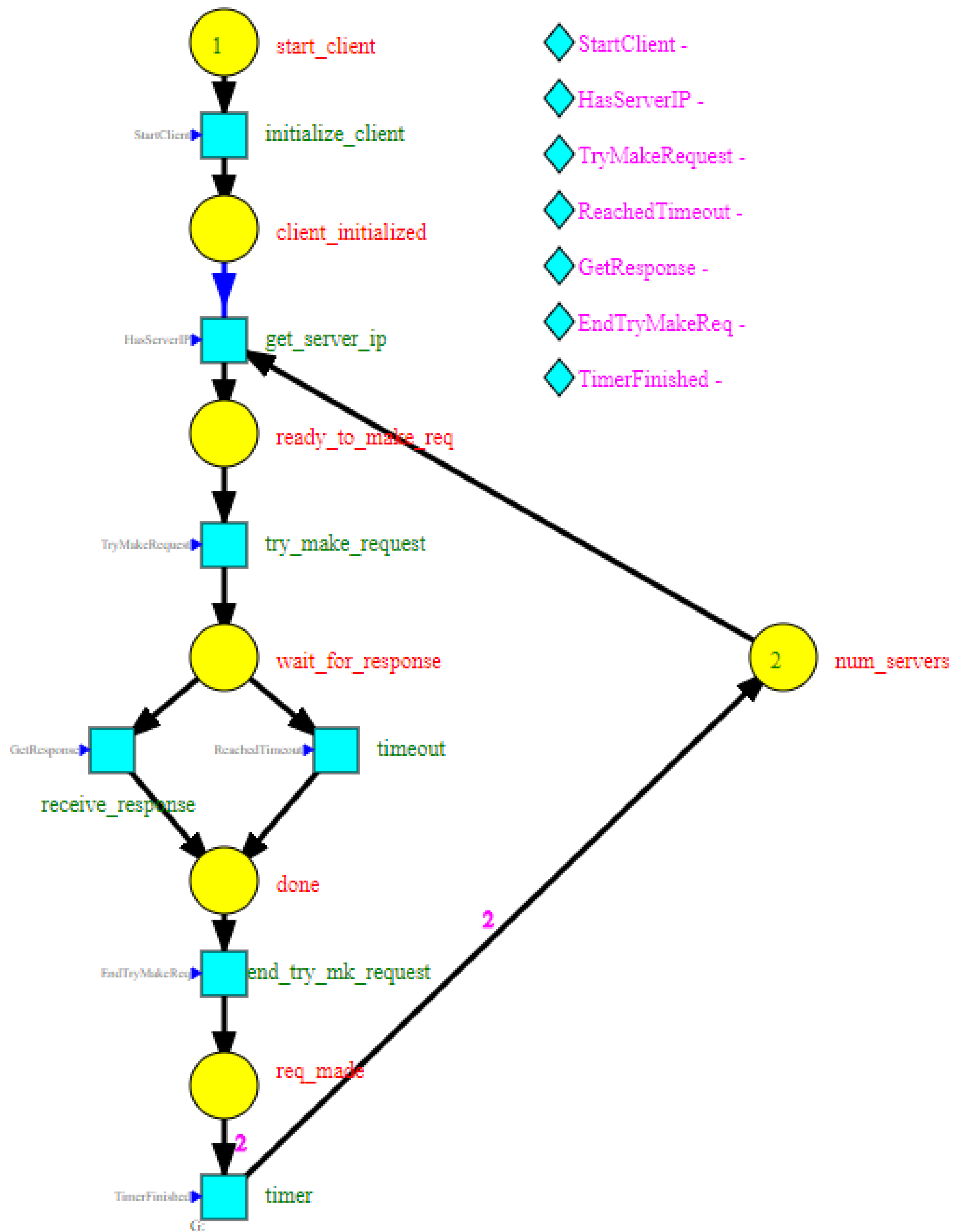


Figura 25 - Modelo do cliente utilizado para comunicação entre sinais de trânsito