Instituto Superior de Engenharia do Porto

Mestrado em Engenharia Informática
Tecnologias do conhecimento e decisão

Sistema multi-agente para gestão de tráfego em centros urbanos

Autor:
António Mota

Orientador:
Doutor António Silva

Júri:

Presidente:
Doutora Maria de Fátima Coutinho Rodrigues

Vogais:
Doutor José Avelino da Silva Marinho
Doutor Antonio Pinto de Sousa e Silva

Novembro de 2010

Agradecimentos

Gostaria de começar por agradecer ao **Dr. António Silva** pela disponibilidade e paciência que sempre teve comigo ao longo do tempo que partilhámos.

À **Dra. Fátima Rodrigues** pelo profissionalismo e compreensão que sempre demonstrou.

Ao meu colega e grande amigo **António Vaz**, pelas incontáveis horas de trabalho em conjunto ao longo do percurso académico (*porque as verdadeiras amizades são fruto da dedicação mútua*).

À minha **família**, porque nunca deixou de me apoiar.

À minha querida **Sandra**, pelo amor que sempre me deu.

E por fim, às minhas duas razões de vida, a **Lia** e a **Luz**.

Resumo

A circulação automóvel nos centros urbanos deixou de ser uma definição de dois estados - pode avançar ou não - para passar a considerar uma quantidade razoável de variáveis que determinam como, quando e o que fazer quanto à aplicação de alterações na gestão de tráfego, idealmente em tempo real.

O funcionamento destes sistemas depende da recolha de dados sobre o fluxo de veículos e a sua variação ao longo do tempo. Estas recolhas podem ser feitas de forma automática através da utilização de dispositivos electrónicos e electromagnéticos denominados detectores de presença. O controlo é feito recorrendo à utilização de sinais luminosos (semáforos) controlados por um sistema dedicado e instalado localmente.

Existem, no entanto, outros casos em que os referidos sistemas locais são controlados por um sistema central que procede a uma gestão baseada em recolhas de dados provenientes dos detectores de presença e na introdução de dados pertinentes por parte de operadores afectos ao sistema. A partir daqui, o processamento dos dados recolhidos pode provocar uma série de alterações na actuação dos controladores para que estes ajustem o seu funcionamento ao volume de tráfego existente naquele instante. Por fim, é realizado um processamento analítico posterior, feito por utilizadores e técnicos, actualmente de forma manual, do qual resulta uma série de decisões sobre o possível alargamento da rede de controladores ligados ao sistema centralizado.

Este tipo de implementação tem algumas limitações que passam desde logo pela dependência exclusiva e directa dos sistemas locais face ao controlador central, o que pode originar situações indesejáveis quando ocorrem por exemplo, cortes nas comunicações.

Com este trabalho, pretende-se realizar um estudo de um sistema análogo ao descrito acima e propôr adaptações do mesmo a um sistema baseado em multi-agentes com as seguintes características principais:

- Cada controlador local é representado por um agente
- Permitir a comunicação e partilha de conhecimento entre agentes
- Aumentar a capacidade de reacção face a ocorrências em tempo real
- Dotar os agentes de capacidades pró-activas
- Controlador central como agente coordenador que detém o conhecimento a nível global e que proporciona a ligação com o interveniente humano.

Abstract

Traffic circulation within the cities, is no longer a two state definition - you may proceed or not - as it began to consider a reasonable amount of variables which help determine how, when and what to do about possible changes in traffic management, ideally in real time.

These systems operate depending on the data gathered concerning the flow of vehicles and its variation through time. The data can be obtained automatically through the use of electronic and electromagnetic devices called presence detectors. Traffic control uses light based signals (traffic lights) connected to a dedicated system, installed locally in each intersection.

There is, however, another kind of traffic control in which these so called locally installed systems are controlled by a central system that processes data received from the presence detectors and also data entered directly by system operators. From this point on, data processing may produce a series of changes in the local systems operating routines, forcing them to adjust those routines at any given moment. Afterwards, an analytical data processing is manually performed by technicians and users, from which results a series of decisions about possible enlargements of the area controlled by the central system.

This type of implementation implies a few limitations such as the exclusive dependency of the local systems towards the central controller, which may provoque unwanted situations when faced with communications prob-

lems, among others.

In this paper, we intend to study a similar system to the one described above, while proposing its adaptation to a multi-agent based system with the following features:

- Each local controller is represented by an agent
- Enable communication and knowledge sharing among agents
- Increase the ability to react to events in real time
- Give the agents pro-active capabilities
- The central controller is a coordinator agent that holds global knowledge and provides a link with human users.

Conteúdo

Lista de Figuras

Lista de Algoritmos

Listings

Capítulo 1

Introdução

O crescimento das cidades e o consequente aumento no número de veículos têm como impacto natural o aumento dos congestionamentos e dos acidentes de tráfego. A solução mais habitual para este tipo de problemas passa pelo alargamento das vias que, apesar de criar um efeito positivo imediato, apresenta igualmente algumas desvantagens tais como a redução de espaços públicos ou o custo associado que é normalmente muito elevado.

Uma das soluções mais populares durante as décadas de 80 e 90, passou pela criação de sistemas automáticos de gestão de tráfego cujo objectivo passa não por renovar a rede viária mas antes por optimizar a rede existente maximizando a sua utilização. Estes sistemas cediam, de forma pré-estipulada, sequencial e numa rotina incessante, a passagem a cada uma das vias que lhes estavam afectas. Inicialmente, apenas uma avaria poderia interromper o ciclo eterno que era percorrido por estes sistemas sendo-lhes mais tarde adicionadas outras características, hoje consideradas essenciais, tais como a capacidade de ter múltiplas programações consoante as variações de tráfego ao longo do dia ou a simples possibilidade de ser colocado em modo intermitente durante os períodos em que o seu funcionamento não era considerado necessário (por ex-

emplo, durante o período nocturno). Numa fase posterior, foi criada aquela que ainda hoje é considerada a característica mais importante de um sistema luminoso de controlo de tráfego actual: a capacidade de detectar a presença de veículos numa via.

Ao detectar a presença de veículos, o sistema tem a possibilidade de mudar apenas para as vias que efectivamente precisam do verde para avançar (se não existem veículos em espera, não faz sentido abrir o verde numa via) mas, melhor do que isso, pode abrir o sinal verde apenas durante o tempo considerado necessário quando este não atinge o máximo estabelecido. A título de exemplo, se numa via está programado para que o verde esteja ligado durante um tempo máximo de 60 segundos, se esse tempo não foi esgotado e já não houverem mais veículos para passar, o sistema poderá antecipar a abertura do próximo verde e consequentemente da próxima via que possui veículos em espera, procedendo assim a uma gestão muito mais eficaz da intersecção.

Nos dias de hoje, os controladores electrónicos que são utilizados na gestão de tráfego são já bastante evoluídos, permitindo uma adaptação mais eficiente às variações do tráfego diário e permitindo igualmente múltiplas programações distribuídas por tabelas de tempo que podem ser activadas consoante as horas, dias do mês ou dias da semana. Mas, apesar de toda a evolução a que se tem assistido, existe um parâmetro que sempre foi considerado como importante mas cuja implementação nunca acompanhou a eficiência demonstrada pelos sistemas independentes e locais. A capacidade de dois ou mais sistemas independentes comunicarem e partilharem informação entre si com vista à obtenção de objectivos comuns. Esta comunicação já é possível mas de forma muito limitada.

Tudo isto mostrou-se manifestamente insuficiente mediante o estado actual do controlo de tráfego, que utiliza sistemas electrónicos, mais propriamente

nos ambientes urbanos das grandes cidades. Assim, chegamos ao patamar do controlo centralizado. Este tipo de controlo surgiu da necessidade de criação e sustentabilidade de um sistema de mobilidade urbana específico (mais adequado às grandes cidades) que realmente responda aos problemas que hoje são apresentados e que consistem numa variação constante e quase imprevisível das condições diárias de tráfego nestes locais.

Esta alternativa começou a ser estudada já na década de 70 quando se deu início a uma série de estudos por parte da empresa SFIM(5) que resultariam mais tarde naquilo que foi denominado por sistema GERTRUDE[1]. Depois, seria fundada uma empresa com o mesmo nome(6) e a SFIM foi posteriormente adquirida na totalidade pela SAGEM Avionics(8).

Existem outros exemplos de sistemas do género mas presentemente, e falando do caso particular português(9), este é o sistema que existe nas cidades de Lisboa(10) e Porto(14) embora neste último o sistema seja denominado por SIGA[2].

A grande diferença entre o sistema convencional e o centralizado é que no primeiro caso, os dados recolhidos pelos detectores são utilizados localmente para saber qual a via a abrir e durante quanto tempo, mediante uma ou várias programações possíveis e pré-determinadas. No segundo caso, toda a informação recolhida é enviada para um posto central que analisa os dados de forma global e envia instruções para os controladores locais com vista a melhorar não só o tráfego local mas também com o objectivo de melhorar o tráfego circundante pertencente a outras intersecções.

À primeira vista, esta solução parece que responde a todas as questões e problemas que poderão existir e em parte isto é verdade. Tanto o é que este

[1]Gestion Electronique de Régulation en Temps Réel pour l'Urbanisme, les Déplacements et l'Environnement

[2]Semáforos Inteligentes de Gestão Automática

tipo de controlo permite o acompanhamento de veículos de emergência durante o seu percurso, facilitando-o, permite a atribuição de prioridades às vias de transportes públicos, permite também o controlo do volume de veículos nas entradas e saídas do centro urbano e permite, com a conjugação de todos estes factores, a melhoria global do comportamento automóvel e da qualidade do ar através da diminuição da poluição atmosférica. Subsistem no entanto alguns problemas que continuam a carecer de solução: a capacidade de comunicação entre controladores (no sistema centralizado a comunicação é feita apenas entre controlador e posto central através de cabos rígidos, o que, em caso de quebra de comunicações, representa um problema grave) e a partilha de informações entre estes. A capacidade de reacção é limitada à análise dos dados por parte do posto central e não existem capacidades pró-activas com vista a assegurar eventos que possam ser eventualmente previstos.

Capítulo 2

Critérios de implantação de semáforos

2.1 Volume de tráfego

Um dos factores mais importantes para a avaliação da necessidade de semaforização, passa pelo volume de tráfego que circula nas vias que se intersectam no local. De uma forma geral existem duas situações mais comuns sendo a primeira quando existe uma via principal com um volume de veículos consideravelmente superior à via secundária, provocando uma diminuição de oportunidades de passagem para quem vem desta via e a segunda, quando as vias têm um volume de veículos igualmente elevado diminuindo as oportunidades de passagens entre as várias vias. Tudo isto tem como consequência o aumento de filas e atrasos na circulação.

Foram estabelecidos limites de volumes para justificar a implantação de semáforos(24) considerando o número de faixas de rodagem e o número de veículos especialmente em horas de ponta. De referir ainda que estes limites podem variar caso o nível populacional seja inferior a 10.000 habitantes.

2.2 Volume de tráfego pedonal

A quantidade de peões que circulam nas imediações do local a analisar e mais especificamente os que realizam travessias, são outro factor a ter em conta. Quando o tempo de espera para realizar a travessia de uma intersecção é superior a 30 segundos, a probabilidade de o peão assumir um comportamento tido como de risco (tenta atravessar utilizando os espaços entre veículos) aumenta consideravelmente(24). Existem países onde é estabelecido um número mínimo de peões por hora para implantar semáforos sendo que esse número pode variar entre 190 e 300 quando observado em horas críticas ou de ponta para os peões.

2.3 Acidentes

Os acidentes ocorridos numa intersecção e particularmente os que envolvem casos mais graves (feridos graves ou mortes) funcionam como o argumento mais forte para pedir a implantação se semáforos. Este pedido parte normalmente dos habitantes próximos ao local mas é importante avaliar a causa dos acidentes e qual a relação com a falta de semáforos. Está estabelecido que o número mínimo de acidentes com vítimas graves ou mortais é de 5 no espaço temporal de um ano, desde que tenham ocorrido pela falta de semáforos no local(24).

2.4 Outros critérios

Existem outros critérios tais como volumes mínimos de tráfego, número mínimo de vias, velocidade máxima permitida no local igual ou inferior à velocidade máxima dentro das localidades (será o mesmo que afirmar que apenas é equacionada a colocação de semáforos nestas zonas) e ainda a utilização frequente

de agentes da autoridade para controlar o tráfego numa intersecção durante um mínimo de duas horas diárias.

Pode ser ainda considerada a utilização de semáforos em tempo parcial, colocando-os em modo inactivo durante o resto do tempo se durante pelo menos duas horas diárias o local apresentar motivos válidos para colocar a semaforização mas não atinja os mínimos exigidos fora desse período.

2.5 Programa base

Depois de estabelecer a necessidade de semaforizar uma intersecção, é necessário definir qual o tipo de programa base para o controlador. O programa base funciona como a principal referência para o controlador quando este não está inserido num sistema centralizado ou, estando inserido, perde comunicação com a central de controlo. Existem três tipos de programação:

- Totalmente actuado
- Semi actuado
- Totalmente fixo

No primeiro caso, não é possível distinguir uma via principal de entre todas as que se intersectam no local ou seja, o volume de tráfego é razoavelmente distribuído por todas as vias. Outra maneira de definir uma via principal é quando nesta existem carreiras de transportes públicos. Caso isto seja verdade para mais do que uma via, considera-se o número de passageiros.

No segundo caso, é possível distinguir uma via principal (utilizando os critérios já referidos) e esta terá prioridade em termos de ordem de abertura, atribuição da maior percentagem de tempo de verde e da menor de vermelho

e ainda destino preferencial após satisfação da abertura das vias onde foi detectada a presença de veículos.

O terceiro caso apenas é indicada a sua existência já que a sua utilização é muito rara e não faz muito sentido face à realidade viária actual.

Uma outra questão ponderada na definição da programação base e que tem a ver com o funcionamento particular das instalações independentes, é a distância para a intersecção semaforizada mais próxima ser inferior a 300 metros para obrigar à sincronização entre os controladores. Isto implica imediatamente que a programação base é do segundo tipo ou seja, semi actuada.

Capítulo 3

Sistema convencional

Um controlador de semáforos isolado possibilita a criação de uma ou várias programações a serem utilizadas numa dada intersecção consoante os parâmetros introduzidos pelo utilizador. O caso mais habitual será a utilização de programações diferentes consoante as horas e os dias da semana devido às variações do fluxo de tráfego ao longo do tempo.

A evolução deste tipo de sistemas atingiu o seu pico com a integração de componentes que permitem a detecção de veículos nas diferentes vias (fig.3.1), permitindo assim a evolução do funcionamento através de uma ou mais programações introduzidas na memória que, após a análise dos dados e fazendo a respectiva confrontação com os cenários possíveis, traduz-se numa série de estados (aberto para uns, fechado para outros).

3.1 Equipamentos

Os equipamentos ou dispositivos utilizados na gestão do tráfego podem ser divididos em grupos de acordo com a função que executam no sistema. Existem dispositivos de recolha, dispositivos de armazenamento, dispositivos de proces-

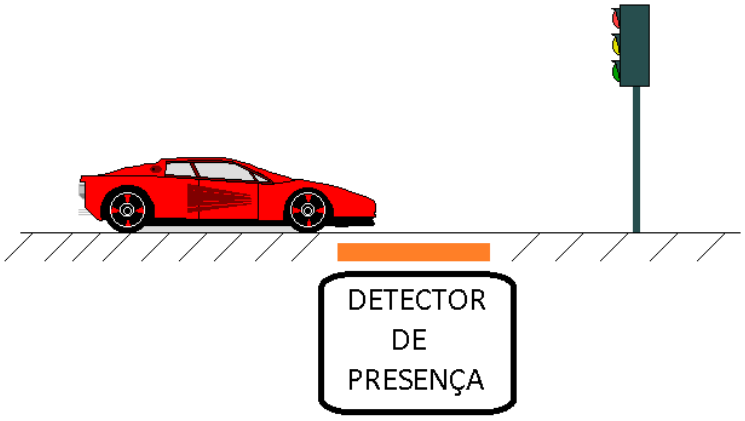

Figura 3.1: Sistema convencional de controlo de tráfego - Os veículos são detectados e a evolução do sistema depende exclusivamente destas ocorrências

samento e dispositivos de interface e comunicação. Para além destes, existem os equipamentos ditos exteriores ou de superfície tais como os semáforos. Para este trabalho, o tipo de dispositivo mais relevante é o de recolha de dados pelo que entraremos mais em pormenor neste tipo de equipamento.

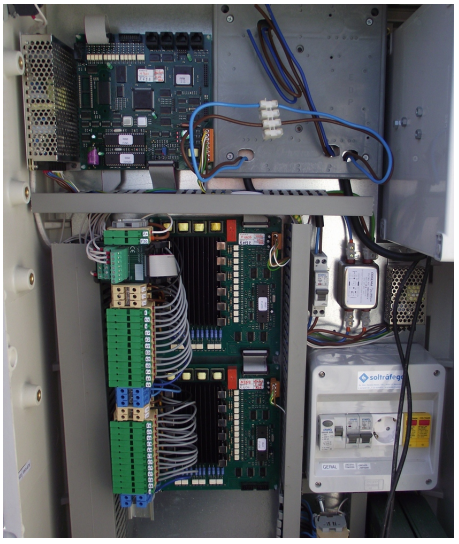

Figura 3.2: Controlador de semáforos convencional

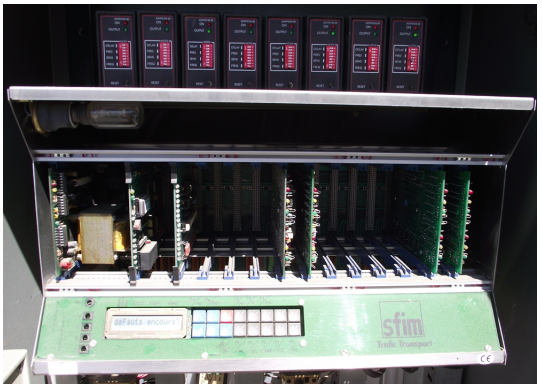

Figura 3.3: Controlador de semáforos convencional

3.1.1 Controladores

Denominam-se por controladores de semáforos os equipamentos com capacidade de armazenamento de dados (instruções) e processamento dos mesmos. Inicialmente, estes equipamentos eram electromecânicos, evoluindo posteriormente para unidades totalmente electrónicas (fig.3.2 e fig.3.3) com as reconhecidas capacidades de processamento que lhes estão associadas.

A função básica de um controlador, seja este de que tipo for, será sempre a de percorrer uma determinada rotina e executá-la. Esta rotina tende a ser maioritariamente estática e as únicas alterações à mesma apenas são decorrentes de alterações realizadas à geometria da intersecção ou ao sentido das vias que lhe estão afectas. O mesmo será dizer que, depois de estabelecida a rotina de uma intersecção, apenas se farão alterações ao funcionamento da mesma ao nível de tempos. As rotinas são frequentemente denominadas por etapas de funcionamento e definem quais as vias que abrem o sinal verde e quais as vias incompatíveis entre si.

Será de referir que, durante a execução das rotinas, são avaliadas variáveis que informam sobre a existência de veículos nas várias vias e apenas nas situações em que isto seja verdade é que a respectiva linha de rotina é executada. Caso contrário, avança para a linha seguinte.

Existem várias soluções no mercado mas de forma genérica, um controlador de semáforos é composto pelos seguintes módulos:

- **Módulo de alimentação** - Possui um transformador duplo 220V⇒12V e 220V⇒5V. Alberga um *watch-dog*[1] e um comutador electrónico para fazer a transição cores - intermitente. Fornece, como seria de esperar,

[1] O *watch-dog* analisa continuamente as variações na tensão de alimentação principal para, no caso desta ser demasiado baixa ou alta para manter os parâmetros normais de funcionamento, colocar o controlador em modo de avaria, ou seja, as saídas para os semáforos em amarelo intermitente

toda a energia para os restantes módulos do controlador.

- **Módulo CPU** - Composto por um processador e memórias do tipo RAM, apoiadas por uma bateria interna para suster os dados em caso de falha de energia.

- **Módulo de entradas** - Este módulo permite uma ligação directa com o *output* dos detectores electromagnéticos, recebendo assim os dados independentemente do tipo de detecção realizada.

- **Módulo de saídas** - Este módulo funciona como um conjunto de interruptores que abrem e fecham de acordo com as instruções provenientes da CPU. É composto por componentes electrónicos denominados por TRIAC's que, ao receber um impulso de baixa tensão na Gate, permite a passagem de corrente eléctrica entre os dois Ânodos. Esta passagem de corrente é bidireccional (corrente alternada) e a tensão máxima admitida entre os Ânodos ronda os 600 Volts.

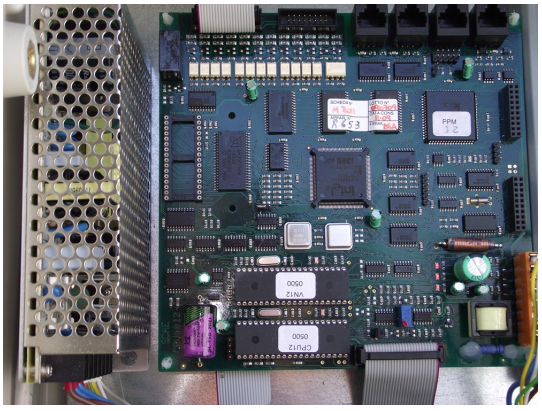

Figura 3.4: Módulo de alimentação (à esquerda, protegido por uma grelha metálica) e módulo CPU com bateria de suporte de memória

Figura 3.5: Módulos montados em rack - da esquerda para a direita: Alimentação, CPU, entradas e 2 saídas duplas. Display e teclado incorporados

Um controlador, quando instalado numa intersecção, possui um determi-

nado número de entradas e saídas e este permite a inserção de uma programação que poderá ser mais ou menos complexa consoante a intersecção em causa assim o exija. Normalmente e como já foi referido, a programação inicial é ajustada ao longo do tempo consoante são realizadas observações ao comportamento do tráfego no local sendo que os ajustes são maioritariamente ao nível dos tempos de verdes e vermelhos. Na maior parte dos casos, os ajustes são considerados mínimos já que a existência de detectores permite alguma elasticidade ao comportamento do controlador mas existem sempre limites a essa elasticidade e daí os referidos ajustes.

Ao longo dos anos, as consequentes evoluções destes sistemas permitiram aumentar cada vez mais a capacidade de adaptação às variações de tráfego mas a evolução natural foi, no caso de zonas extremas, a passagem para sistemas centralizados que mais não são do que os mesmos controladores aqui referidos com a diferença de estarem ligados a um sistema central (o controlador central é normalmente um computador com maiores capacidades de armazenamento e processamento de dados) do qual falaremos no capítulo seguinte.

Ao contrário do que se poderia supor, a implantação de sistemas centralizados não determinou o fim dos controladores locais. Ao invés disso, permitiu um reaproveitamento dos mesmos e a respectiva adaptação a uma realidade diferente em que as capacidades de detecção e processamento locais são utilizadas em benefício do funcionamento global do sistema.

3.1.2 Detectores electromagnéticos

Quanto aos detectores de veículos, existem vários tipos(15), tais como detectores por espiras de indução, detectores electromagnéticos, sensores eléctricos, sensores por processamento de imagens de vídeo, radares de efeito doppler,

sensores de infravermelhos, sensores ultra-sónicos e detectores acústicos passivos.

Os diferentes tipos de detectores poderão permitir obter diferentes tipos de dados(16). Qualquer um deles permite saber, para uma determinada via ou local, o volume de tráfego, a existência de veículos e a percentagem de ocupação da via mas apenas alguns permitem registar a velocidade dos veículos e são ainda menos os que permitem registar o tipo de veículos que circulam. O caso mais comum neste último tipo de detecção passa pela utilização de sensores por processamento de imagens de vídeo.

Figura 3.6: Conjunto de detectores electromagnéticos

Figura 3.7: Conjunto de detectores electromagnéticos

A configuração mais habitual e presente nos casos nacionais (Lisboa e Porto) passa pela instalação de câmaras de vídeo apenas para acompanhamento do comportamento do sistema ao longo do dia e ainda pela instalação de detectores electromagnéticos dispostos estrategicamente para permitir três leituras principais:

- Detecção de presença de veículos
- Detecção de fim de fila(17)

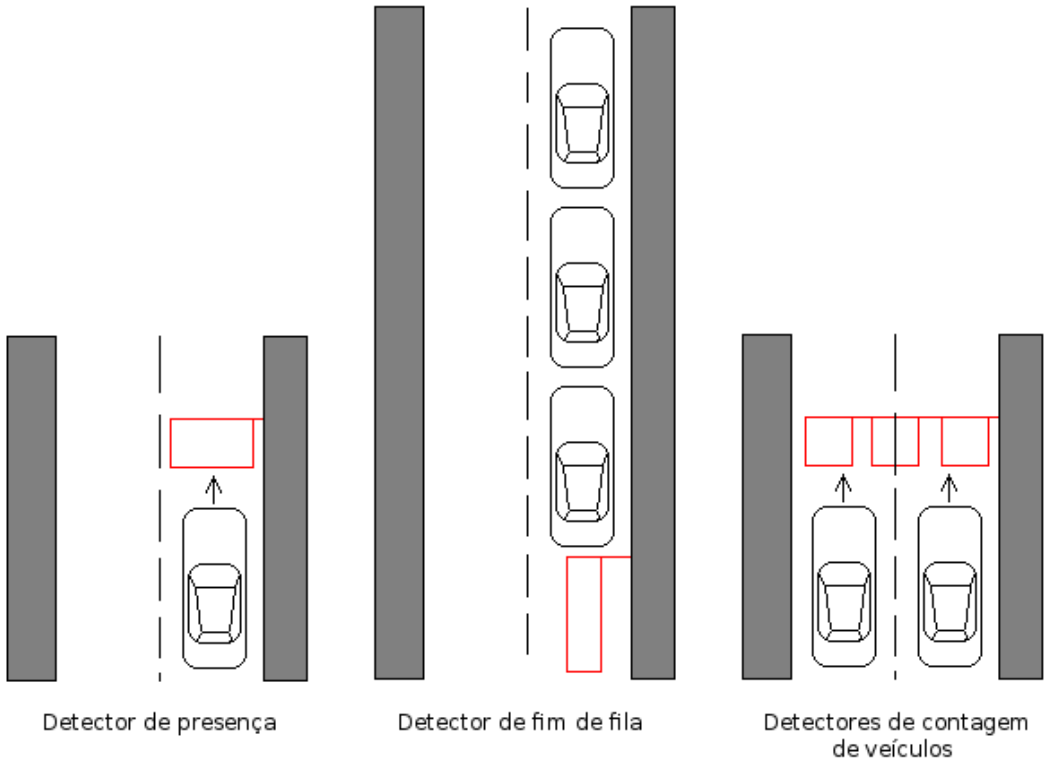

Figura 3.8: As três configurações mais utilizadas nos casos nacionais de Lisboa e Porto

- Contagem de veículos

Os detectores electromagnéticos são compostos por um aparelho electrónico e um cabo que forma um círculo com uma ou mais voltas normalmente denominado por espira ou "loop". Nesta espira é criado um campo magnético que, ao ser distorcido ou interrompido pela passagem de um veículo, despoleta uma acção na componente electrónica enviando um sinal de que foi detectada a presença de um veículo. Depois, e dependendo da colocação da espira, são recolhidos dados relativos ao que se pretende.

As três configurações mais utilizadas nos casos nacionais de Lisboa e Porto (fig.3.8), são descritas da seguinte forma:

- **Detector de presença**

Uma espira ligada a um detector electromagnético é colocada estrategicamente (antes do semáforo a que diz respeito com uma distância que varia normalmente entre os 10 e os 30 metros) e sempre de forma a que consiga abranger toda a faixa de rodagem. Quando um veículo passa pela zona de detecção, é enviado um sinal para o controlador. No caso de existirem várias faixas de rodagem para o mesmo sentido de tráfego, pode ser construída uma espira maior (no máximo, uma espira poderá ir até duas faixas) ou então são construídas várias espiras ligadas em paralelo para simular uma única espira de detecção de presença.

Este tipo de detector representa cerca de 90% de todos os detectores existentes num sistema centralizado e são considerados as peças fundamentais de um sistema deste tipo.

- **Detector de fim de fila**

  Esta configuração é, em termos técnicos, em tudo semelhante às espiras de detecção de presença com duas particularidades relativamente a estas: são construídas longitudinalmente, de forma a ocupar apenas um corredor estreito ao centro da faixa de rodagem e por fim, são colocadas a uma distância relativamente ao semáforo a que correspondem que varia entre os 100 e os 300 metros.

  O princípio é simples, quando um sinal de detecção é contínuo (indicação de que um veículo está parado sobre a espira), o controlador pode ser programado para, ao fim de um determinado tempo sem interrupção dessa continuidade, efectuar uma acção correspondente a esse acontecimento. Isto permite concluir que uma fila de tráfego atingiu uma determinada distância considerada exagerada. Daqui se pode compreender os valores de distância indicados.

Um veículo que passe sem parar ou, estando parado, não ultrapasse o tempo programado, não é considerado pelo controlador apesar de os impulsos de passagem chegarem ao mesmo. Poderia, no máximo, considerar-se uma dupla função para os detectores de fim de fila, poderiam servir como contadores e veículos e detectores de fim de fila ao mesmo tempo mas isto desde que se tratasse de apenas uma via de trânsito, como veremos de seguida, nos detectores de contagem de veículos.

A questão da construção longitudinal à faixa de rodagem permite que, estando uma fila formada sobre a espira, a mesma não fique no espaço vazio entre dois veículos impedindo a detecção da fila. Por fim, a espira é estreita e central à faixa de rodagem para impedir que um veículo estacionado ou parado na berma provoque uma leitura errada.

- **Detectores de contagem de veículos**

  Mais uma vez, este tipo de detecção é muito semelhante às anteriores no que diz respeito à construção das espiras no entanto, as semelhanças terminam aí e isto porque o equipamento electrónico utilizado é diferente dos restantes.

  Continua a ser um detector electromagnético mas com um sistema suplementar denominado MLL[2] montado lado a lado com os detectores e que permite uma leitura lógica das detecções realizadas. Observando a fig.3.8, existem 3 espiras colocadas em duas faixas de rodagem. Uma em cada faixa e a terceira a meio destas. A saída dos detectores é ligada ao MLL (cada MLL pode ler dados provenientes de 6 detectores no máximo) sendo identificados, da esquerda para a direita na figura, como espiras A, B e C.

[2]MLL - Módulo de leitura lógica

O MLL produz impulsos de saída (cada impulso corresponde a um veículo) de acordo a seguinte tabela:

A	B	C	número de impulsos
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	2
1	1	0	1
1	1	1	2

Desta forma é possível determinar, de acordo com o número de impulsos recebidos, qual o número de veículos que circula numa via. Esta tabela de verdade, corresponde ao exemplo apresentado na fig.3.8 mas caso se tratem de mais vias de circulação, a tabela será, obviamente, adaptada.

Capítulo 4

Sistema centralizado

Antes de mais, um sistema centralizado consiste na utilização de vários controladores locais aos quais é adicionada a capacidade de comunicar com um denominado controlador central. Uma das questões que desde logo é conveniente esclarecer é: a comunicação é feita estritamente entre os controladores locais e o controlador central. Não existe qualquer comunicação entre controladores locais. Para além disto, as comunicações são realizadas através de "ramos" que partem do controlador central e a iniciativa de comunicação parte maioritariamente do controlador central.

No caso de um controlador local que é inserido num sistema de gestão centralizado, a filosofia de funcionamento mantém-se apenas com uma alteração. Os parâmetros que despoletam a utilização de uma determinada programação dependem não do que o utilizador introduziu mas do que o sistema central determina.

Por sua vez, o sistema central envia as ordens de mudança de programação consoante as informações recolhidas pelos sensores e respectivo processamento. Por exemplo, se num controlador local está a ser utilizada a programação em que uma via tem um determinado tempo e existe outra linha de programação

no mesmo controlador local que, para a mesma via providencia mais tempo, o sistema central pode emitir uma ordem ao controlador para mudar da primeira programação para a segunda se a via em questão tiver um fila demasiado comprida ou se o número de veículos por hora é superior às restantes vias que intersectam no mesmo local.

Não existindo comunicação entre os controladores locais e sendo necessário, por exemplo, proceder a um sincronismo entre intersecções, podemos referir que, quando se pretende limitar a velocidade dos veículos numa via procedendo à abertura de verde pelas várias intersecções que a via atravessa de modo sequencial e com um tempo tal que, se os veículos não ultrapassarem uma velocidade pré estabelecida, poderão realizar todo o percurso numa "onda verde". Esta tarefa é delegada ao controlador central que emitirá as respectivas ordens de utilização das linhas de programação e ainda as ordens para sincronizar as aberturas iniciais de verde entre os vários controladores locais.

Uma outra forma de utilizar o sincronismo passa por distribuir o congestionamento de tráfego existente numa intersecção pelas intersecções circundantes diminuindo os tempos de verde para as vias que conduzem veículos para o ponto de conflito.

A composição de um sistema de gestão de tráfego centralizado pode ser resumida pela fig.4.1.

Este tipo de ligação para estabelecimento das comunicações, está associado desde logo a um problema que poderá ser de alguma gravidade: caso seja interrompida a comunicação num dos ramos (por exemplo, um corte do cabo devido a obras), todos os controladores a jusante da interrupção ficarão desligados do controlador central, sem qualquer hipótese de receber instruções ou de enviar dados. Em casos mais extremos, em que a interrupção da ligação provoca um curto-circuito poderá perder-se a comunicação na totalidade do

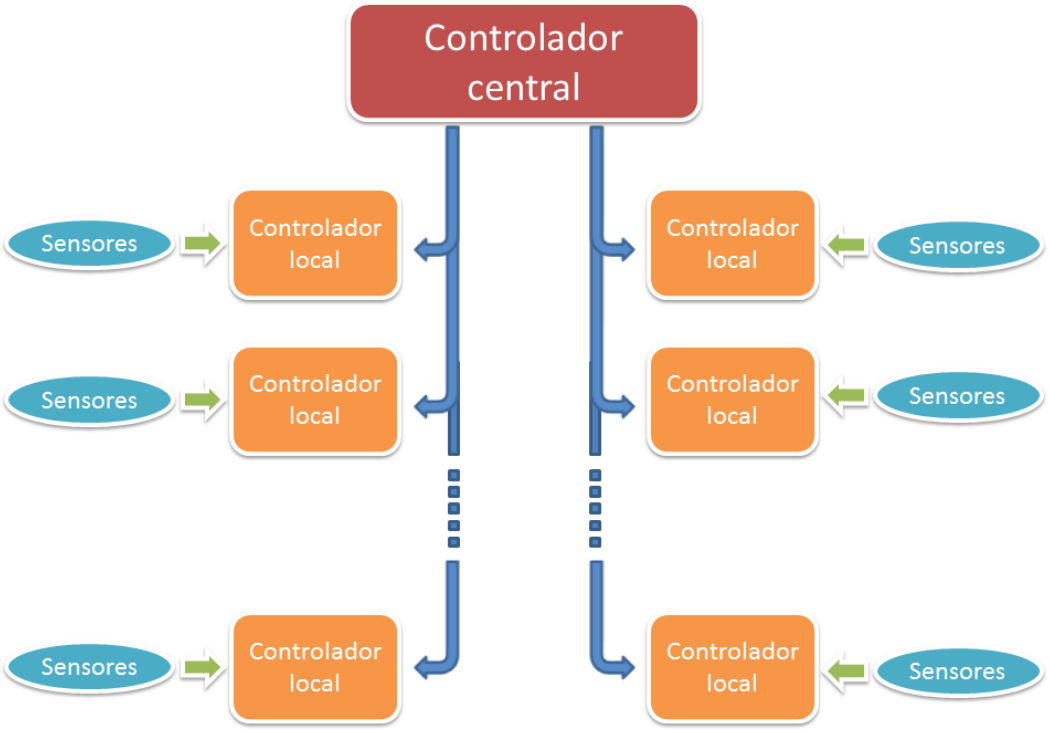

Figura 4.1: Representação de um Sistema de Gestão de Tráfego Centralizado - O controlador central possui "ramos" de comunicação através dos quais comunica com os controladores locais. Cada "ramo" liga a um máximo de 7 controladores.

ramo afectado.

De referir ainda que, para estes casos de falhas nas comunicações, o controlador local tem um programação "base" que entrará em funcionamento mas que de modo algum poderá substituir eficientemente a intervenção do controlador central no funcionamento da intersecção.

Um caso especial num sistema centralizado tem a ver com o acompanhamento dos veículos de emergência que pode ser feita de duas formas: por GPS ou através de emissores nos veículos em questão. O sistema de gestão utiliza estes meios para saber em cada instante ou ponto de verificação (caso se trate respectivamente da primeira ou segunda forma de acompanhamento) qual a direcção que o veículo especial leva para que os controladores que se encontram no caminho recebam ordens de funcionamento adequadas à livre passagem do veículo pelas intersecções.

Devido à rapidez necessária na tomada de decisões num sistema deste tipo, seria impensável recolher os dados fornecidos pelos sensores, processar esses mesmos dados e informar o utilizador das conclusões para que este tomasse uma decisão informada. é bastante mais prático dotar o sistema de meios para dar seguimento ao processamento dos dados, sendo este a tomar as decisões dentro de parâmetros pré estabelecidos. Caberá ao utilizador analisar a forma de actuação do sistema para proceder a ajustes nos parâmetros de decisão que este utiliza.

Uma das tarefas mais habituais para os utilizadores destes sistemas de gestão, é a análise de dados provenientes relativos a intersecções semaforizadas localizadas na proximidade de uma intersecção não semaforizada. Após essa análise, na qual são incluídos dados como o número de veículos nas diferentes vias ou ainda o número e tipo de acidentes, o utilizador poderá chegar à conclusão que há necessidade de colocar semáforos na intersecção em causa e ligar

o controlador respectivo ao sistema central.

De igual forma poderá o utilizador concluir que determinada intersecção semaforizada já não necessitará de o continuar a ser embora este tipo de situação seja menos frequente.

Em suma, um sistema de gestão de tráfego opera no sentido de aumentar a fluidez de circulação, diminuir as filas de tráfego, diminuir a poluição atmosférica, diminuir os tempos de espera para os tansportes públicos tendo tudo isto como última consequência um aumento da qualidade de vida na zona de intervenção do sistema.

Uma das grandes referências de sistemas centralizados para controlo de tráfego, é o sistema GERTRUDE, existindo a nível nacional exemplos práticos da sua utilização na cidade do Porto e Lisboa.

O sistema GERTRUDE (ou SIGA) tem por objectivo ajudar a melhorar a circulação numa determinada zona abrangida por várias intersecções semaforizadas. Para tal, terá que existir interacção entre o sistema e os semáforos e para este caso particular a interacção é feita através de sensores de presença que fornecem dados para processamento e posterior variação no funcionamento dos semáforos.

Note-se ainda que nos casos em que o referido sistema existe, nem todas as intersecções existentes dentro da área de intervenção estão semaforizadas. Mais ainda, a área de intervenção sofre expansões regulares de acordo com as necessidades de propagar o sistema para novas zonas onde o fluxo de tráfego aumenta. Aqui observam-se duas lacunas do sistema actual, sendo que este sistema não fornece conclusões ou indicações quanto à necessidade de semaforizar as intersecções que, estando dentro da área, não o são e ainda quanto ao tipo de programação a adoptar nos semáforos.

4.1 Descrição do Sistema

Com o aumento do tráfego nas cidades e o consequente aumento o número de intersecções semaforizadas, tornou-se necessário implementar um sistema que estabelecesse uma série de parâmetros no funcionamento dos semáforos devido, em grande parte, à proximidade entre estes e ao fluxo variável do número de veículos em circulação ao longo do dia.

Os parâmetros a implementar foram os seguintes:

- Sincronismo entre a abertura de verde numa intersecção para uma qualquer direcção e o verde da intersecção seguinte para a mesma direcção.
- Adaptação do sincronismo para as necessidades de veículos de emergência facilitando a circulação dos mesmos através da criação de canais de verde contínuo.
- Aumento ou diminuição do tempo de verde para uma dada direcção como consequência da varição do fluxo de tráfego.
- Aumento ou diminuição do tempo de vermelho como forma de nivelar o fluxo de tráfego distribuindo-o pelas várias intersecções.
- Detecção de bloqueamentos nas vias como forma de impedir a alimentação contínua de veículos nessas zonas desviando-os para zonas menos congestionadas.

4.2 Cronologia

- Década de 70: A gestão dos semáforos era feita localmente através de controladores electromecânicos.

- 1974: É criado o conceito de sistema de controlo de tráfego.
- Abril de 1976: É criado o sistema GERTRUDE.
- 1985: O Sistema GERTRUDE é implementado na cidade de Lisboa.
- 1993: A cidade do Porto centraliza parte dos semáforos do centro histórico com a criação do sistema SIGA.

4.3 Como funciona

Este tipo de sistema foi concebido para responder a três necessidades principais:

- Assegurar fluidez no tráfego.
- Melhorar o desempenho dos transportes públicos.
- Assegurar prioridade a veículos de emergência.

Existe um quarto objectivo que não sendo menos importante, está relacionado com os restantes dependendo directamente do sucesso destes e que é o controlo da poluição atmosférica dentro da zona de actuação.

Tudo isto é realizado através da leitura em tempo real de sensores de presença espalhados estrategicamente pela várias zonas. Estes sensores permitem saber onde existe maior fluxo de tráfego, quais as vias bloqueadas e ainda qual o percurso dos veículos de emergência.

Munido destas informações, o sistema age directamente sobre a actuação dos semáforos, modificando o seu modo de funcionamento atavés da variação dos tempos de verde e vermelho em cada intersecção.

A adopção deste sistema não eliminou no entanto a existência de controladores locais às intersecções. Bem pelo contrário porque em última instância

e em caso de falha nas comunicações com a central do sistema, são estes que tomam sobre si a responsabilidade de continuar a gerir os semáforos localmente.

A gestão quase partilhada entre a central e os controladores resume-se a dizer que, para cada intersecção, são criadas várias programações (dependendo do local podem ser dezenas) armazenadas redundantemente na central e no controlador local. Ao analisar o estado dos sensores, o computador central dá indicações ao controlador acerca da programação que deve utilizar naquele instante. Em caso de falha nas comunicações, como já foi referido, existe sempre uma programação por defeito que o controlador utiliza até que seja reposta a normalidade.

Os veículos de emergência surgem como uma grande excepção no sistema pois este está preparado para criar verdadeiros canais de verde de forma a facilitar e acompanhar o percurso e evolução destes veículos através das várias zonas controladas. Normalmente este acompanhamento é feito por GPS e no caso particular do território nacional, foram instalados emissores especiais nos veículos de emergência que emitem sinais únicos que são detectados pelos sensores espalhados pelas zonas.

4.4 Outros exemplos

Similarmente aos sistemas de gestão de tráfego já referidos, existem outros sistemas que perseguem os mesmos objectivos embora com abordagens diferentes no seu funcionamento.

Exemplo disso mesmo é o sistema colocado em Leeds, no Reino Unido, que faz uso de controladores de semáforos em conjunto com sensores electrónicos para aumentar a fluidez de tráfego em intersecções de conflito recorrente. O sistema foi denominado MOVA(19).

Outro exemplo que está até mais aproximado aos casos nacionais referidos está presente nos Estados Unidos, mais propriamente no estado do Utah com a denominação ITS(20). A maior diferença entre este exemplo e os casos já expostos é que neste, foi incorporado um sistema de apoio à decisão que auxilia os operadores e assume algumas das conclusões como exequíveis partindo de instruções actualizadas de forma diária. Neste sistema são propostos objectivos a cumprir e o sistema tenta concluir no sentido de satisfazer esses mesmos objectivos.

Capítulo 5

Sistemas baseados em agentes

É cada vez mais comum a existência de interoperabilidade entre sistemas que executam tarefas diversas ou com objectivos de funcionamento diferentes. O mesmo será dizer que é cada vez mais necessário existir algum tipo de troca de informação entre sistemas que à partida poderiam ser totalmente incompatíveis. Algumas das razões para estas incompatibilidades têm a ver com a falta de homogeneidade dos vários sistemas existentes. Programas diferentes são escritos em linguagens diferentes, por pessoas ou equipas de pessoas diferentes e em diferentes períodos de tempo. Os próprios *interfaces* acabam por ser diferentes mesmo dentro de sistemas com a mesma função.

Os sistemas baseados em agentes são uma resposta a estas dificuldades, permitindo que componentes de software diferentes sejam encapsulados em agentes de software, possuindo a capacidade de comunicação com outros agentes, através da utilização de uma linguagem comum. Isto poderá, à partida, assemelhar-se bastante ao comportamento humano quando estamos perante situações em que duas ou mais pessoas sentem a necessidade de interagir, frequentemente em busca de objectivos que sirvam as necessidades de cada indivíduo mas não raras vezes em prossecução de objectivos comuns.

5.1 Agentes

O que é um agente? Existem várias definições possíveis para um agente de software, sendo algumas delas referenciadas por Franklin e Graesser(2) e das quais podemos destacar as seguintes:

- Um agente é capaz de percepcionar o ambiente que o rodeia através de sensores e de actuar nesse mesmo ambiente através de actuadores.
- Agentes autónomos são sistemas computacionais que habitam um ambiente complexo, sentem e actuam sobre esse ambiente, e ao fazê-lo determinam uma série de objectivos ou metas para o qual foram implementados.
- Um agente pode ser definido como uma entidade de software persistente dedicada a um objectivo específico. "Persistência" distingue agentes de subrotinas; agentes têm ideias próprias quanto à forma como desempenham as suas funções. "Objectivo específico" distingue-os de grandes aplicações multitarefas; agentes são tipicamente muito mais pequenos.

Os agentes de software fazem hoje parte de uma enorme variedade de sistemas espalhados pelo mundo. Estes podem ser tão simples como um pequeno *script* mas são, na sua maioria, entidades de software relativamente complexas e dotadas de grandes capacidades. De uma forma geral, um agente pode ser comparado a um objecto em software cujo funcionamento interno está normalmente encapsulado através de métodos que permitem a manipulação do mesmo de forma controlada. Da mesma forma o agente de software possibilita a comunicação com outros agentes, encobrindo ao mesmo tempo o seu funcionamento interno. A grande diferença está na actuação do agente de software que é mais autónoma.

Segundo Wooldridge e Jennings(4), as propriedades básicas inerentes a um agente de software são:

- Os agentes conseguem funcionar sem controlo directo de outros agentes ou pessoas. São autónomos.
- Os agentes podem interagir com outros agentes ou com pessoas. Possuem capacidades de comunicação.
- Os agentes reagem a estímulos provenientes do ambiente em que se encontram. São reactivos.
- Os agentes podem tomar decisões com vista a atingir objectivos e metas definidos. São pró-activos.

Russel e Norvig(3) têm uma noção de agente onde, segundo eles, "*An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors*". Esta noção está plenamente de acordo com o sentido de utilização de um sistema de controlo de semáforos baseado em agentes onde se pretende dotar o sistema de capacidades de percepção do ambiente (sensores de veículos) para que possa actuar sobre esse ambiente através de actuadores (os sinais luminosos).

5.1.1 Linguagem de comunicação de agentes

A criação de *standards* para as linguagens utilizadas na comunicação entre agentes facilita a criação de programas que conseguem efectivamente comunicar entre si. Desde que os programas a desenvolver cumpram os requisitos estabelecidos, não terá qualquer importância a estrutura ou a plataforma que os suporta.

Actualmente, é comum o estabelecimento de *standards* numa grande diversidade de domínios. Por exemplo, diferentes fornecedores de serviços de mail conseguem trocar mensagens entre si devido à utilização de *standards* como o SMTP. Os vários navegadores de internet existentes no mercado não dependem da tecnologia utilizada no servidor web para poderem abrir uma página porque todos os navegadores são desenvolvidos de acordo com *standards* como o HTML para o código das páginas e o HTTP para o protocolo de comunicação.

O grande problema surge efectivamente quando é necessário que dois programas, construídos em plataformas diferentes ou que não comunicam na mesma linguagem, interajam. Antes de mais, é necessário lidar com as eventuais inconsistências na sintaxe utilizada. A mesma expressão poderá ter interpretações diferentes consoante se trate de um programa ou outro. Também poderão surgir incompatibilidades quando dois programas utilizam expressões diferentes para caracterizar a mesma coisa.

Para resolver estas questões, foram criados *standards* que definem linguagens de comunicação de agentes. Um exemplo é a *ACL*[1], proposta pela *FIPA*[2]. Nesta proposta de linguagem, são definidas uma série de primitivas e o respectivo significado. Para além da utilização desta linguagem de comunicação, é necessário utilizar também uma ontologia comum aos vários agentes.

Outro exemplo de linguagem de comunicação de agentes é o *KQML*[3] que resulta de um estudo da *DARPA KSE*[4], tendo igualmente produzido o *KIF*[5] como uma linguagem de definição de conteúdo (lógica de *first-order* + teoria de *sets*) e a *OntoLingua* como uma especificação de ontologia. O *KQML* é

[1] Agent Communications Language
[2] Foundation for Intelligent Physical Agents
[3] Knowledge Query and Manipulation Language
[4] Knowledge Sharing Effort
[5] Knowledge Interchange Format

insensível acerca da utilização da linguagem de conteúdo, qualquer que esta seja.

A especificação do *KQML* define a sintaxe de uma colecção de mensagens, denominadas "*performatives*", que declaram a linguagem de comunicação dos agentes. Existe um núcleo de mensagens reservadas mas o conjunto é extensível. A tabela 5.1 classifica esse núcleo de mensagens e indica se as mesmas são utilizadas na comunicação entre pares de agentes ou em contextos mais alargados (comunidades de agentes).

5.2 Multi-Agentes

Sendo possível criar agentes de software, poderá existir a necessidade de aumentar o seu número dentro do mesmo ambiente de acção, o que nos leva aos sistemas multi-agente. Nestes cenários, é necessário que exista colaboração entre os vários agentes, pelo que foram criadas duas abordagens possíveis: a colaboração directa e a colaboração assistida (1). Com a colaboração directa, os agentes não dependem de mais ninguém que não sejam eles próprios para garantir a realização de pedidos e com a colaboração assistida, introduz-se a definição de um mediador nas comunicações entre agentes.

Algumas das desvantagens da colaboração directa são:

Custo Quando o número de agentes é reduzido, o custo poderá não ser um factor muito desvantajoso mas em embientes onde o número de agentes é efectivamente elevado, o custo de envio de pedidos e o consequente processamento desses mesmos pedidos poderá conduzir a situações de funcionamento caóticas. Essencialmente, será necessário introduzir num ambiente deste tipo as capacidades necessárias para lidar com este tipo de volume de informação, o que nem sempre será possível quer ao nível dos

custos, quer ao nível da própria funcionalidade, resultando num sistema pobre e pouco eficaz.

Complexidade Uma outra desvantagem passa pela complexidade de implementação deste tipo de sistemas. Nos esquemas de colaboração directa, cada agente é responsável por negociar directamente com os outros agentes e devem suportar toda a programação necessária a estas negociações, traduzindo-se em agentes demasiado elaborados para que seja possível, de uma forma simples e rápida, proceder tanto à sua manutenção como à sua alteração e evolução.

Com o objectivo de eliminar este tipo de desvantagens e conseguir criar agentes menos complexos mas bastante mais funcionais, uma alternativa passa pela colaboração assistida, mediante a utilização de uma terceira entidade, os facilitadores. Desta forma, os agentes não comunicam directamente uns com os outros mas com os facilitadores que, por sua vez, comunicam quer com os agentes, quer entre si. Reduzem-se desta forma os custos associados aos volumes de informação que circulam no sistema e também a complexidade dos agentes que passam a ser mais simples.

Num sistema multi-agente, os agentes podem ter um determinado conhecimento que corresponde normalmente ao conhecimento local que cada um tem relativamente ao ambiente que o rodeia. Cada agente, embora no mesmo sistema, poderá fazer face a acontecimentos e eventos totalmente diferentes de todos os outros. Nestes casos, poderá ser útil ter um conhecimento global de todos os ambientes e por conseguinte de todo o sistema. Aqui entra uma outra definição dos sistemas multi-agentes: os agentes coordenadores.

A função deste tipo de agentes passa não só por interligar todos os agentes do sistema (os agentes continuam a poder comunicar directamente entre si)

mas também, e talvez seja o mais importante, por manter o conhecimento global. Isto permite, em qualquer altura da execução do sistema, ter uma perspectiva que não seria possível através de apenas alguns dos agentes.

Em suma, os sistemas baseados em multi-agentes representam um conjunto mais ou menos complexo de entidades de software que, à semelhança dos sistemas mais simples, conseguem comunicar e cooperar entre si mas de forma a conseguir tratar um maior volume de informação ou reter um conhecimento mais alargado sobre os meios ambientes e sistemas onde estão inseridos.

Class	*Performatives*	Utilização
Query and response	ask-if,ask-all, ask -one, tell, untell, deny, sorry	par de agentes
Cursor Manipulation and Result Formatting	ready, next, discard, rest stream-all, eos	par de agentes
advertise or commit to a capability	advertise, unadvertise	comunidade
kb editing	insert,uninsert, delete-one, delete-all, undelete	par de agentes
enactment	achieve, unachieve	par de agentes
error handling	error	par de agentes
communication primitives other than pure asynchronous messages	broadcast, forward, standby, subscribe	ambos
trading	broker-one, broker-all, recommend-one, recommend-all, recruitone, recruit-all	comunidade
name service	register, unregister, transport-address	comunidade

Tabela 5.1: Tabela classificação de *KQML performatives*

Capítulo 6

Solução proposta

Vimos até este momento qual a composição actual de um sistema de controlo de tráfego quer na implementação isolada, quer na vertente centralizada onde a complexidade é consideravelmente superior. Pretende-se, com este trabalho, propôr a alteração de um sistema centralizado actual para um sistema de gestão de tráfego baseado em agentes. As alterações propostas são as seguintes:

Substituição do controlador central O controlador central actual consiste num ou mais computadores com a capacidade de comunicar com os controladores locais através de um protocolo criado especificamente para o efeito e utilizando o sistema de comunicações descrito na figura 4.1. Este deverá ser substituído por equipamentos com capacidade de processamento maior onde será instalado o agente de software que desempenhará o papel de agente coordenador no novo sistema de gestão de semáforos. A ligação deste agente ao novo sistema de comunicações será feita de acordo com a representação da figura 6.3.

Substituição dos controladores locais Os controladores locais que actualmente são compostos por componentes electrónicos construídos especi-

ficamente para o controlo de semáforos (figuras 3.4 e 3.5) deverão ser totalmente sustituídos por equipamentos informáticos (computadores) que permitam a instalação de agentes de software que desempenharão as funções de agentes controladores.

Substituição do sistema de comunicações Actualmente, as comunicações são feitas através de ligações dedicadas que partem do controlador central em direcção aos controladores locais (figura 4.1). Pretende-se substituir este sistema na totalidade por uma rede que utilize o protocolo TCP/IP onde serão ligados todos os agentes, quer controladores, quer o coordenador. As ligações poderão ser do tipo *ethernet* ou *wireless*, dependendo do custo associado a cada solução.

Esta proposta de substituição surge porque o sistema actual está associado a alguns problemas como o possível corte nas comunicações que, a acontecer em qualquer um dos "ramos" que partem do controlador central, inviabilizam as comunicações pelo menos do local de corte em diante (cada "ramo" liga a um máximo de 7 controladores).

Manutenção dos equipamentos de superfície Designam-se por equipamentos de superfície todos os componentes utilizados para interacção com os intervenientes no tráfego (veículos, peões) e propõe-se a manutenção destes equipamentos no novo sistema a implementar já que o controlo dos mesmos não implica interfaces ou ligações dedicadas.

6.1 Controlador local - Agente

Com este trabalho, propõe-se que os controladores locais (doravante designados apenas por controladores) sejam substituídos por componentes de hard-

ware com maior capacidade de processamento e armazenamento, possibilitando igualmente a instalação de um componente de software (agente). O agente a considerar deverá ter características idênticas aos controladores actuais no que respeita ao funcionamento de controlo de semáforos, adicionando a estas as características básicas de um agente de software.

Figura 6.1: Representação do agente controlador de semáforos

A estrutura do agente controlador de semáforos será a representada na fig.6.1 e que se descreve de seguida:

Base de conhecimento O agente deverá possuir e manter uma base de conhecimento que consegue aumentar e enriquecer ao longo do tempo. O conhecimento de um agente encerra parâmetros tais como: quantas vias controla de momento, qual o número médio diário de veículos em cada uma, qual o número máximo de veículos e as datas e horas em que

tal ocorre, quais os agentes que lhe estão próximos (com os respectivos endereços para possibilitar a comunicação) ou ainda quais as vias com prioridade (vias com transportes públicos ou rotas habituais de veículos de emergência).

Módulo de comunicação Comunicar é talvez a característica mais importante de um agente. Através da comunicação (com outros agentes), consegue partilhar e trocar informação útil que lhe permita atingir os objectivos a que se propõe. É igualmente através desta comunicação que os agentes informam o coordenador central da sua situação actual, permitindo uma visão global sobre todo o sistema para que se possa proceder a ajustes que, de outra forma, poderiam ser imperceptíveis quando observados apenas numa perspectiva local ou limítrofe a algumas intersecções.

Processamento de sensores A capacidade de processar os dados recolhidos por aparelhos construídos para o efeito é outra das características que o agente controlador de semáforos deverá possuir. Estes dados resumem-se a impulsos (como o abrir e fechar de interruptores) e coincidem com a passagem dos veículos nas diferentes vias. Na realidade dos centros urbanos, podemos estar a falar de uma grande quantidade de impulsos (chegam a ser milhares por hora) que, após recepção, carecem de tratamento próprio e eficiente. Daí a necessidade de um módulo especialmente construído para o tratamento e processamento dos dados recolhidos.

Processamento de rotinas Um agente controlador de semáforos deverá, antes de tudo, fazer aquilo que o próprio nome refere: controlar semáforos e através deste controlo permitir ou bloquear a passagem dos veículos nas vias afectadas. Este controlo é realizado mediante a utilização de rotinas

que não são mais do que todas as sequências possíveis para os diferentes estados dos semáforos que estão ligados ao agente. Isto sugere de imediato que estas rotinas possam ser programáveis (cada intersecção tem um número pré-determinado de semáforos) e adaptáveis a cada situação.

É igualmente habitual, nos sistemas actuais, fazerem-se sub-rotinas que indicam, para além das sequências possíveis, qual a ordem pela qual essas sequências são apresentadas. No entanto, tratando-se de um agente com capacidades de autonomia e decisão, estas sub-rotinas deverão ser suprimidas, deixando para o próprio agente a decisão sobre qual a sequência a adoptar nas transições dos vários estados possíveis. Deste modo, o agente pode decidir qual a sequência seguinte (qual a melhor) para a realidade momentânea do ambiente em que este se insere.

Sensores Representam o meio através do qual o agente "sente" o que se passa à sua volta. Através destes é possível saber em qualquer momento quais as vias com maior e menor tráfego, qual o número de veículos em circulação (média, total), qual o comprimento de uma fila de trânsito. Todas estas informações são passadas para o processamento de sensores e aí tratadas convenientemente para utilização do agente e eventual enriquecimento da base de conhecimento.

Rotinas de funcionamento Um controlador actual segue uma programação linear ou seja, depois de programadas as sequências de estados pretendidas e a ordem pela qual estas são executadas, estas rotinas são seguidas de forma sistemática ao longo do tempo. Poderíamos eventualmente equacionar algumas alterações da ordem de execução resultante, por exemplo, da detecção de impulsos especiais por parte dos sensores (ex: detecção de um veículo de emergência). No entanto, estas situações não

deixariam de ter que ser previstas na programação das rotinas.

Tal não é verdade quando nos referimos a agentes. Neste caso, apenas são colocadas nas rotinas de funcionamento todos os estados possíveis para os semáforos ligados ao agente para que este depois decida autonomamente sobre qual a sequência a utilizar na transição entre os estados. Esta decisão é realizada na parte de processamento de rotinas cujo principal objectivo é exactamente esse: decidir o que fazer a seguir.

6.2 Controlador central - Agente coordenador

Figura 6.2: Representação do agente coordenador

Quanto à implementação do controlador central (doravante desginado por coordenador), é proposto com este trabalho a adaptação do sistema actual para um sistema baseado em agentes. A criação de um agente coordenador num sistema de controlo de semáforos permite a existência de algumas características que contribuem para a melhoria de funcionamento do referido sistema.

Se os agentes controladores detêm o conhecimento de tudo o que se passa e existe à sua volta, isso poderá ser uma vantagem quando analisamos o funcionamento dos semáforos individualmente em cada intersecção ou, no máximo, num pequeno conjunto de intersecções vizinhas. Se, no entanto, existir a necessidade de proceder a uma análise mais global, ao nível do sistema como um todo, o conhecimento existente nos controladores não é suficiente.

Exemplos desta necessidade poderão ser as horas de ponta, onde poderá ser preciso condicionar ou bloquear a entrada de mais veículos dentro da zona controlada pelo sistema porque foi atingido um nível de saturação. Neste caso seria manifestamente difícil para um controlador conseguir tomar conhecimento desta situação e decidir pelo condicionamento ou bloqueio mas para o agente coordenador, que detém o conhecimento global e comunica com qualquer controlador, será mais fácil de concluir e decidir neste sentido.

A estrutura e características inerentes ao agente coordenador estão representadas na fig.6.2 e serão as seguintes:

Base de conhecimento O agente coordenador deverá possuir e manter uma base de conhecimento actualizada onde constarão factos como os endereços de cada controlador, as relações de vizinhança entre controladores, as vias existentes na área de controlo do sistema, os sensores existentes em cada via e as respectivas funções (já vimos anteriormente que os sensores podem ter várias configurações e funções), as vias prioritárias e a localização das possíveis origens de emergências (localizações de hospitais, bombeiros, polícia).

Para além disto, é igualmente necessário guardar na base de conhecimento os dados provenientes dos controladores e recolhidos nos sensores de veículos. Estes dados servirão para uma análise mais detalhada da

realidade da zona abrangida pelo sistema, permitindo ao agente coordenador concluir sobre uma série de acontecimentos que, de outro modo, seriam imperceptíveis ao nível dos controladores. Pormenores como a necessidade de condicionar o fluxo de veículos que se dirigem para o interior da zona controlada e aumentar o fluxo de veículos que se dirigem para o exterior com o objectivo de aliviar uma situação de saturação das vias afectadas, serão apenas alguns dos vários exemplos para os quais este tipo de análise contribuirá.

Ao permitir a recepção e armazenamento dos dados provenientes dos controladores, o coordenador está também a retirar alguma da já grande carga colocada em cima destes. Dadas as características do coordenador (maiores capacidades de processamento e armazenamento), esta tarefa fica facilitada quando comparada com a alternativa de guardar e analisar os dados nos controladores. Para além de tudo isto, o coordenador também pode, como tarefa adicional exercida durante a análise dos dados referidos, concluir sobre aspectos tais como a necessidade de semaforizar intersecções que até ali não o eram ou, pelo contrário, concluir que uma intersecção semaforizada simplesmente já não precisa de o ser.

Módulo de comunicação Com uma capacidade idêntica aos agentes controladores, o agente coordenador deverá conseguir comunicar com cada um deles sempre que tal seja necessário. Nos sistemas convencionais, a comunicação entre controlador central e controladores locais é, como foi já referido, bastante limitada já que pode facilmente ser interrompida, originando os problemas óbvios e que são já conhecidos.

Já neste caso em particular, se o coordenador perder o contacto com um controlador específico, poderá, por exemplo, tentar influenciar o compor-

tamento desse controlador emitindo pedidos aos controladores que lhe são vizinhos e isto porque o novo esquema de comunicações pretendido por este trabalho, garante que as comunicações existirão sempre nos casos de cortes parciais e focalizados em apenas alguns controladores. O novo esquema de comunicações está representado na fig.6.3.

Processamento de dados Este módulo representa o núcleo do agente coordenador. Aqui serão processados e analisados os dados provenientes dos controladores, permitindo ao coordenador concluir sobre eventuais necessidades do sistema, após o que emitirá as respectivas indicações. Devemos referir que todos os dados provenientes dos controladores (abertura de vias, bloqueio de vias, número de veículos, fins de fila actuados) são enviados para o coordenador como uma espécie de sinais ou disparos que ocorrem e que servirão para alimentar a base de conhecimento global.

Todos os dados guardados são então processados de forma esporádica ou periódica, consoante seja necessário. O processamento periódico deverá originar uma série de conclusões baseadas em factos como as vias com maior volume de tráfego, ou ainda as vias que persistentemente bloqueiam devido a saturação excessiva. Essas conclusões poderão ser tão simples como mudar o sentido de circulação de uma via ou tão complexas como criar ondas de verde em determinados trajectos que envolvam várias intersecções e por conseguinte vários agentes controladores.

Já o processamento esporádico deverá ser realizado no sentido de encontrar eventuais conclusões sobre o alargamento do sistema para zonas não controladas e que o sendo, poderiam afectar positivamente o funcionamento do sistema.

6.3 Percepção do ambiente - sensores

Um agente de software deve poder receber informação relativa ao ambiente que o rodeia. Para isso, o agente utiliza os chamados sensores que fornecem informação sobre o estado desse mesmo ambiente. Este tipo de dispositivos são imprescindíveis para o bom funcionamento de qualquer agente de software e poderá assumir várias formas e implementações.

No caso de um sistema de controlo de semáforos, os sensores utilizados actualmente e em maior número são os detectores electromagnéticos. Para esta solução, propõe-se a utilização do mesmo tipo de detectores com as mesmas configurações existentes nos casos já referidos de Lisboa e Porto para que os agentes controladores tenham a informação necessária sobre a intersecção que lhes está afecta:

Sensores de presença Estes tipo de sensor compõe a maioria do conjunto total de sensores do sistema de controlo dos semáforos proposto. A sua função principal será a de fornecer informação ao agente sempre que um veículo chega à via onde o sensor está colocado. Poderá eventualmente ser utilizado como contador de veículos embora não seja o equipamento ideal para essa tarefa.

Sensores de fim de fila Os sensores de fim de fila irão desempenhar uma função importante na actividade sensorial de um agente já que permite que este saiba quando uma via está saturada de veículos. Isto poderá interferir positivamente nas decisões do agente, levando-o a optar por um funcionamento em que é dada prioridade aos sentidos de circulação que não vão saturar ainda mais a via, podendo mesmo chegar ao ponto de bloquear por completo a abertura da passagem de veículos na direcção dessa via até que esta retome o seu estado normal.

Sensores de contagem Estes sensores são ligados a dispositivos construídos especificamente para o efeito e são compostos por um equipamento electrónico de detecção e um módulo MLL. Com a combinação dos dois, é possível obter informação detalhada sobre o número de veículos que circulam numa ou mais vias com o mesmo sentido de circulação. Estes valores de contagem servirão, na sua maioria, para as decisões a tomar futuramente pelo agente controlador num âmbito de comportamento próactivo.

Através da utilização dos vários tipos de sensores disponíveis, todos os agentes controladores conseguirão ter uma percepção sobre o estado da intersecção onde estão colocados. Este conhecimento deverá ser guardado na base de conhecimento de cada agente, enviado para o controlador central e ainda partilhado entre os agentes que o solicitem numa perspectiva de prossecução colectiva de objectivos.

6.4 Sistema de comunicações

Um dos maiores problemas do sistema centralizado actual é o sistema de comunicações que este utiliza. As ligações são estabelecidas através de cablagem de cobre, partindo do controlador central para os controladores locais mas com uma série de limitações:

"Ramos" de ligações Existem várias ligações a partir do controlador central, consideradas como "ramos" que se estendem através de uma série de controladores locais. O máximo que estes "ramos" permitem ligar são 7 desses mesmos controladores. Por exemplo, para uma série de 70 controladores espalhados num sistema centralizado actual, seriam necessários 10 "ramos" de ligações.

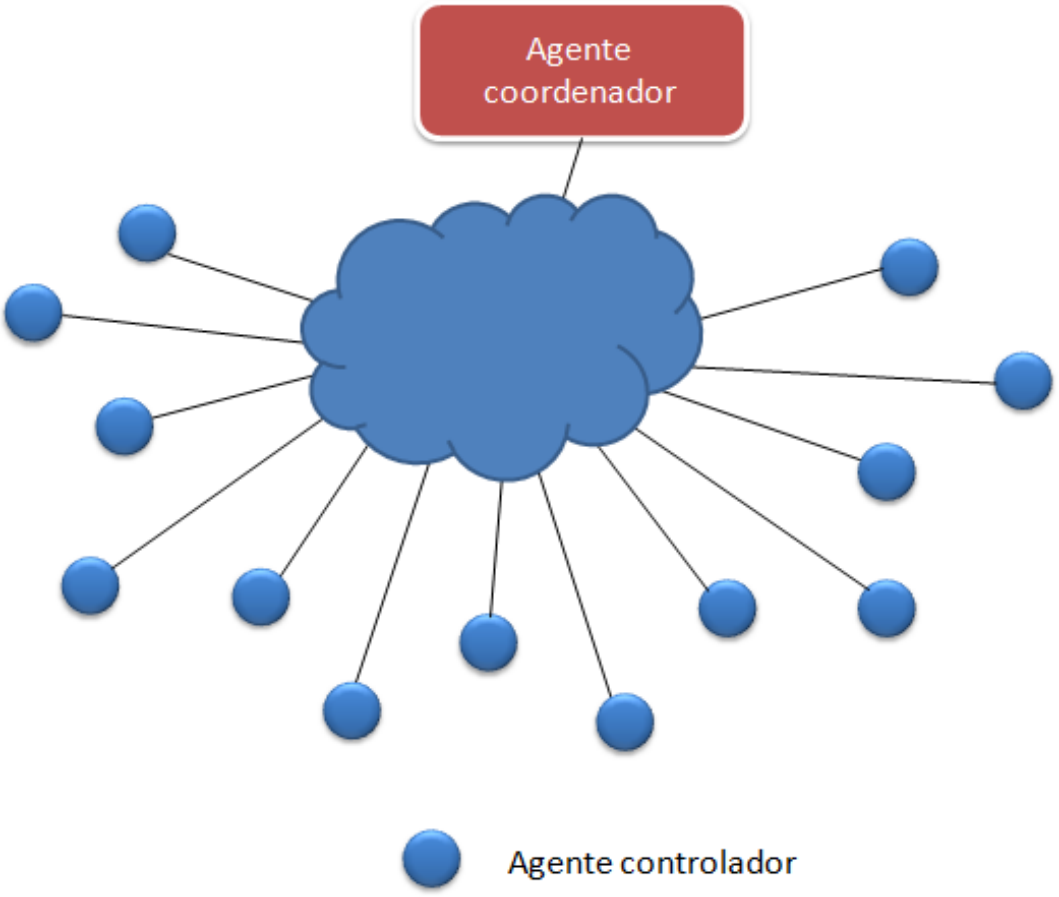

Figura 6.3: Novo sistema de comunicações

Considera-se uma limitação porque, no caso de uma quebra de comunicações num dos "ramos" (devido a obras, por exemplo) e dependendo do tipo de corte, a comunicação não será possível a partir do ponto de corte em diante e em casos extremos até poderá não ser possível em todo o "ramo". De referir ainda que, na eventualidade de cortes nas comunicações, os controladores locais entram no chamado modo de funcionamento local que está previamente configurado em cada um deles.

Este funcionamento local será o equivalente a ter uma intersecção isolada e independente, sem qualquer tipo de ligação a outros sistemas. O controlador assume uma postura de funcionamento em que apenas reage aos

pedidos de passagem que lhe vão sendo dirigidos, ignorando por completo eventuais indicações de fim de fila, passagens prioritárias ou contagens de veículos. A existência de um ou mais controladores nesta situação dentro da zona controlada poderá originar situações de congestionamento que não existiriam em funcionamento normal.

Comunicação ponto a ponto Outra das limitações deste tipo de comunicação reside no facto de esta ser realizada ponto a ponto entre o controlador central e cada um dos controladores locais. O mesmo será dizer que apenas o controlador central pode comunicar com cada um dos controladores, estando impedida a comunicação entre controladores locais mesmo que pertençam ao mesmo "ramo" de ligação.

Sendo uma característica inerente à sua própria definição, os agentes de sofware deverão conseguir comunicar entre si, desde que estejam reunidas as condições para isso. Neste trabalho propõe-se o novo sistema de comunicações como se mostra na fig6.3 que servirá para substituir as actuais ligações ponto a ponto e através de ligações do tipo TCP/IP (*ethernet* e *wireless*). Ao transformar os controladores locais em componentes de hardware com outro tipo de capacidades, entre elas a existência de interfaces de rede, passa a ser possível substituir as ligações actuais por outras mais eficazes e que permitem as seguintes vantagens:

Ligações TCP/IP Ao permitir a existência deste tipo de ligações, a comunicação deixa de estar limitada a grupos de controladores e passa a estar globalizada ao longo do sistema. Fica desta forma estabelecida uma rede onde cada agente possui um endereço que lhe permite não apenas aceder à rede mas também ser localizado e contactado na mesma. Actualmente, na eventualidade de cortes nas ligações e dependendo da localização do

corte (o pior caso seria logo no início do "ramo"), todos os controladores que estão daí em diante perdem a comunicação com o controlador central. Com estas novas ligações, cada corte que eventualmente ocorra, apenas afectará um controlador. Para além disto, a possibilidade de utilizar ligações *wireless* minimizaria ainda mais a possibilidade de qualquer ocorrência de cortes (por exemplo, devido a obras).

Comunicação global Com a possibilidade de ter um endereço e poder ser contactado por qualquer entidade ligada à mesma rede, passa a ser possível a comunicação não apenas entre o coordenador e os controladores mas também (e talvez seja esta a característica mais importante do novo esquema de comunicações) entre os controladores. Desta forma, os agentes controladores poderão partilhar conhecimento, realizar pedidos entre si e negociar no sentido de conseguir realizar os objectivos que pretendem.

6.5 Pró-actividade - Análise e acção

A actividade de um agente não se limita a sentir o que se passa à sua volta, a tomar decisões e a comunicar com outros agentes. Um agente também está orientado para a realização de tarefas pró-activas na tentativa de assegurar um melhor funcionamento em situações futuras.

Num sistema de controlo de semáforos actual, a pró-actividade é uma tarefa que se baseia na leitura manual e pontual de dados recolhidos ao longo da execução do sistema, bem como a partir de observações feitas por técnicos designados para o efeito. Este tipo de trabalho, para além de exaustivo, é um trabalho essencialmente de análise e pesquisa e pode ser executado por sistemas computacionais que demonstrarão com certeza um desempenho superior.

Para esta solução propõe-se a utilização de uma das grandes características dos agentes de software, a pró-actividade. Cada agente controlador mantém uma base de conhecimento que poderá analisar. Através desta análise poderão ser encontradas soluções que permitam melhorar o funcionamento da intersecção. Para além disto, a análise dos dados pode resultar em items de negociação a utilizar em eventuais conversações com outros agentes quando tal se verifique necessário.

Foi já descrito o comportamento do coordenador central quanto ao processamento dos dados recebidos e eventuais conclusões tiradas sobre os mesmos. Este tipo de funcionamento é efectivamente pró-activo e faz parte das características que se pretendem vincar nos agentes aqui utilizados. Já quanto aos controladores, a pró-actividade não poderá ser de modo algum global (não detêm o conhecimento necessário) mas antes ao nível local ou, no máximo, restringida à área de intervenção dos vizinhos de cada controlador.

Esta pró-actividade deverá concluir sobre questões pertinentes e que se apresentem como entraves ao bom funcionamento da intersecção. A título de exemplo, se existir uma via que tenha muitos pedidos de passagem (quando confrontado o número de pedidos com as restantes vias), o agente poderá classificá-la como uma via prioritária ou, no caso de já existir uma via prioritária diferente, esta poderá deixar de o ser em detrimento daquela que agora tem mais pedidos. Esta situação poderá até chegar ao ponto em que o agente modifica a via prioritária várias vezes ao dia, consoante as horas a que observa o número de pedidos a aumentar em cada via.

6.6 Funcionamento - tabelas e algoritmos

Os agentes controladores terão que armazenar uma série de dados que poderão eventualmente ser alterados durante a execução e que se traduzem nas características únicas de cada intersecção. Factos como o número de vias, os semáforos existentes, o tipo de semáforo utilizado, os sensores instalados, as vias prioritárias, as vias bloqueadas e as contagens realizadas são alguns dos exemplos a considerar.

Utilizando uma representação com recurso a tabelas, podemos observar a descrição dos dados referidos, que estarão armazenados nos controladores nas respectivas bases de conhecimento:

Via	Tempo mín	Tempo máx	Incremento
A	10	40	4
B	6	30	3

Tabela 6.1: Tabela de tempos (unidade de tempo: segundos)

Via	Semáforo	Tipo
A	1	Veículo
A	2	Veículo
A	4	Peão
A	5	Peão
A	6	Peão
B	3	Veículo
B	7	Peão

Tabela 6.2: Tabela de semáforos

Tabela de tempos (tabela: 6.1) Cada via existente numa intersecção terá dois tempos considerados como mínimo e máximo e ainda um tempo denominado por incremento. Ao iniciar a contagem do tempo de cada via, o agente terá que assegurar que esse tempo atinge o valor de tempo

Sensor	Tipo	Via
1	Presença	A
2	Fim de fila	A
3	Contador	A
4	Presença	B
5	Contador	B

Tabela 6.3: Tabela de sensores

Via	Máx	mín	Média	Máx actual	mín actual	Média actual
A	1000	100	550	400	10	205
B	300	8	154	245	5	125

Tabela 6.4: Tabela de contagens (unidade: veículos por hora)

mínimo, após o que dará sucessivos incrementos de tempo à via (cada incremento tem o valor definido na tabela) até que aconteça uma de duas situações possíveis: não existem mais pedidos de veículos na via ou foi atingido o tempo máximo.

Tabela de semáforos (tabela: 6.2) É característica de qualquer intersecção existirem um ou mais semáforos a controlar cada uma das vias. Esses semáforos estão associados a diversos tipos, sendo os mais vulgares os de veículos e os de peões. Nesta tabela são armazenados os dados relativos aos semáforos de cada via através de um identificador local à intersecção e ainda através do tipo de semáforo que este representa.

Tabela de sensores (tabela: 6.3) Como já foi referido, os sensores são a forma que um agente terá para perceber o que se passa à sua volta. Como tal, terá que saber quais os sensores que lhe estão atribuídos e quais as vias a que dizem respeito. Nesta tabela é possível guardar a identificação de cada sensor juntamente com o seu tipo e a via que está a ser controlada.

Tabela de contagens (tabela: 6.4) Uma forma de proceder a análises de funcionamento passa pela observação de uma série de valores de contagens que são actualizados frequentemente através dos impulsos provenientes dos sensores. Ao conseguir ter este tipo de informação, o agente poderá decidir acrescentar uma via às vias prioritárias ou retirar uma via que já seja considerada prioritária para a substituir por outra. Isto poderá ocorrer várias vezes durante o dia e poderá depender de factores como as horas de ponta, os dias da semana ou até mesmo determinados meses do ano.

Ainda durante o funcionamento dos agentes existentes no sistema, existem vários procedimentos que estes deverão executar e que irão ser aqui traduzidos sob a forma de pequenos algoritmos. Questões de como se deverá comportar perante algumas das situações que lhe serão apresentadas são aqui respondidas através da descrição de passos a percorrer e soluções a considerar.

Algoritmo 1: Funcionamento periódico do agente controlador

```
Fim do tempoVia
Seleccionar novaVia
while continuar is true do
    if novaVia is bloqueada then
        Seleccionar novaVia
    else
        continuar ← false
Iniciar novaVia
Enviar mensagem ao coordenador
```

Um agente controlador terá que ter normalmente uma via seleccionada para fazer circular o trânsito. Essa via permanece aberta até que se esgote o tempo que lhe está atribuído após o que o controlador tentará abrir outra via. Essa selecção depende de vários factores como saber se a nova via que se pretende abrir está declarada como bloqueada ou não, ou saber se a nova via

tem pedidos nos sensores ou não (algoritmos: 1 e 2).

Se não existir nenhuma via que reúna as condições necessárias para ser aberta a sua circulação, permanecerá a via actual em funcionamento para além do tempo que lhe foi estipulado (se numa intersecção não existirem pedidos para passar em nenhuma via ou apenas existirem pedidos numa única via, não existe a necessidade de alternar a abertura das mesmas).

Algoritmo 2: Selecção de via

```
input  : viasPrioritarias, viaActual
output: novaVia
begin Selecção de via
    if viasPrioritarias not null then
        novaVia ← nextVia
        while novaVia not viaActual do
            if novaVia tem pedidos then
                return novaVia
            else
                novaVia ← nextVia
        novaVia ← viaPrioritaria
        return novaVia
    else
        novaVia ← viaPrioritaria
        return novaVia
end
```

Quando estamos perante situações de pedidos de bloqueio, o número de vias a bloquear deverá ser limitado para impedir o excesso de bloqueios. Considere-se que, em qualquer intersecção, apenas poderão estar no máximo 2 vias bloqueadas, uma pelo próprio controlador e outra devido a um pedido realizado por um vizinho ou pelo coordenador. Para minimizar esta situação, o pedido de bloqueio por parte de outro agente apenas será atendido se ainda não estiver nenhuma via bloqueada (mesmo que pelo próprio controlador) ou seja, o número máximo de 2 apenas será atingido se o controlador bloquear uma das suas vias depois de ter atendido um pedido de outro agente (algoritmos: 3 e

4).

Algoritmo 3: Tratamento de pedido de bloqueio

```
input: via, viasBloqueadas, emissor
begin Trata pedido de bloqueio
    if viasBloqueadas > 0 then
        mensagem ao emissor "não bloqueia"
    else
        viasBloqueadas ← via
        mensagem ao emissor "via bloqueada"
        mensagem ao coordenador "via bloqueada"
end
```

Algoritmo 4: Tratamento de fim de fila

```
input: via, viasBloqueadas, listaVizinhos
begin Trata fim de fila
    viasBloqueadas ← via
    mensagem ao coordenador "via bloqueada"
    vizinho is nextVizinho from listaVizinhos
    while via ∉ vizinho do
        vizinho is nextVizinho from listaVizinhos
    mensagem ao vizinho "prioridade para via"
end
```

Quanto ao tratamento de pedidos de declaração de vias prioritárias (um agente detectou um fim de fila e pede ao vizinho que dê prioridade à via para que o fim de fila seja resolvido), o agente deverá tratar esse pedido concedendo-o apenas se a via pretendida não estiver declarada como bloqueada ou se não tiver actualmente nenhuma outra via declarada como prioritária (algoritmos: 4 e 5). Aqui poderá eventualmente ser considerada a possibilidade de *challenge* já que, a existir um pedido de prioridade que interfira com outra via que já é considerada prioritária, o emissor do pedido poderá evocar razões suficientes para que o receptor atenda ao seu pedido mesmo que não o pretenda fazer de início. Por exemplo, o emissor apresenta como argumento um número de veículos superior ao da via que está declarada como prioritária (algoritmos: 6

e 7).

Algoritmo 5: Tratamento de pedido de prioridade

input: *via*, *viasBloqueadas*, *viasPrioritarias*, *emissor*
begin Trata pedido de prioridade
  if *viasPrioritarias* > *0 OR via* ∈ *viasBloqueadas* **then**
    mensagem ao *emissor* "prioridade não atribuída"
  else
    viasPrioritarias ← *via*
    mensagem ao *emissor* "prioridade da *via* atribuída"
    mensagem ao *coordenador* "prioridade da *via* atribuída"
end

Algoritmo 6: Tratamento de resposta de prioridade não atribuída (*challenge*)

input: *via*, *mensagem*, *emissor*
begin Trata resposta negativa de prioridade
  maxActual ← (valor Máx Actual de *via*)
  mensagem ao *emissor* "pedido prioridade através de *challenge* (*via*, *maxActual*)"
end

Algoritmo 7: Tratamento de pedido de prioridade através de *challenge*

```
input: via, max, viasBloqueadas, viasPrioritarias, emissor
begin Trata pedido de prioridade através de challenge
    if viasPrioritarias > 0 then
        maxActual ← (valor Máx Actual de viaPrioritaria)
        if mxActual > max OR via ∈ viasBloqueadas then
            mensagem ao emissor "challenge negado"
        else
            mensagem ao emissor "challenge da via permitido"
            mensagem ao coordenador "challenge da via permitido"
    else
        mensagem ao emissor "challenge da via permitido"
        mensagem ao coordenador "challenge da via permitido"
end
```

Capítulo 7

Implementação de demonstração

Foi objectivo deste trabalho descrever e caracterizar uma solução que, a ser implementada, pretende melhorar um sistema que apresenta actualmente algumas limitações. Para demonstrar, de alguma forma, a implementação e funcionamento de um conjunto de agentes cuja função passa por realizar o controlo de sistemas de semáforos, foi desenvolvida uma pequena aplicação para proceder à referida demonstração.

A aplicação foi totalmente desenvolvida em *LPA-Prolog* (*Winprolog*) e a arquitectura da mesma está representada na figura 7.1. Todos os agentes existentes no sistema têm a possibilidade de comunicar entre si, partilhando informação e enriquecendo as respectivas bases de conhecimento. O papel do agente coordenador passa pela criação de novos agentes controladores e pela manutenção do conhecimento global relativamente a todo o sistema.

7.1 Interfaces gráficos

Para esta demonstração foram criados vários tipos de janelas gráficas, todas recorrendo aos predicados disponibilizados pelo *Winprolog*(25) para esse efeito.

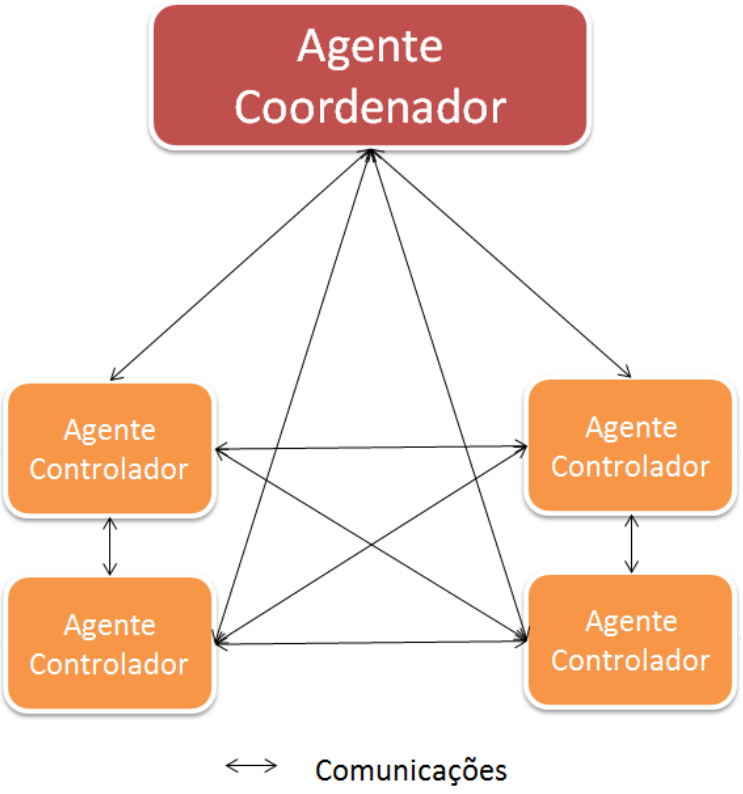

Figura 7.1: Estrutura do sistema de demonstração baseado em agentes

Ao iniciar a aplicação, a primeira janela que é mostrada e que será, ao longo do funcionamento da aplicação, a janela principal é a da figura 7.2 e é criada através do código descrito em A.1. Esta janela está dividida em 3 partes:

Gestão de controladores Os dois botões colocados na parte superior da janela servem para proceder à gestão dos agentes controladores de forma a que seja possível criar novos controladores ou carregar controladores já existentes. Sempre que é criado um novo controlador, este é guardado num ficheiro com a extensão "*.ctrl*" (exemplo de um ficheiro deste tipo na lista A.11), podendo depois ser reaberto para voltar ao funcionamento normal.

Ao seleccionar "Criar controlador", é mostrada a janela da figura 7.3 (lista A.3) onde é possível caracterizar a nova entidade a criar. Ao proceder à

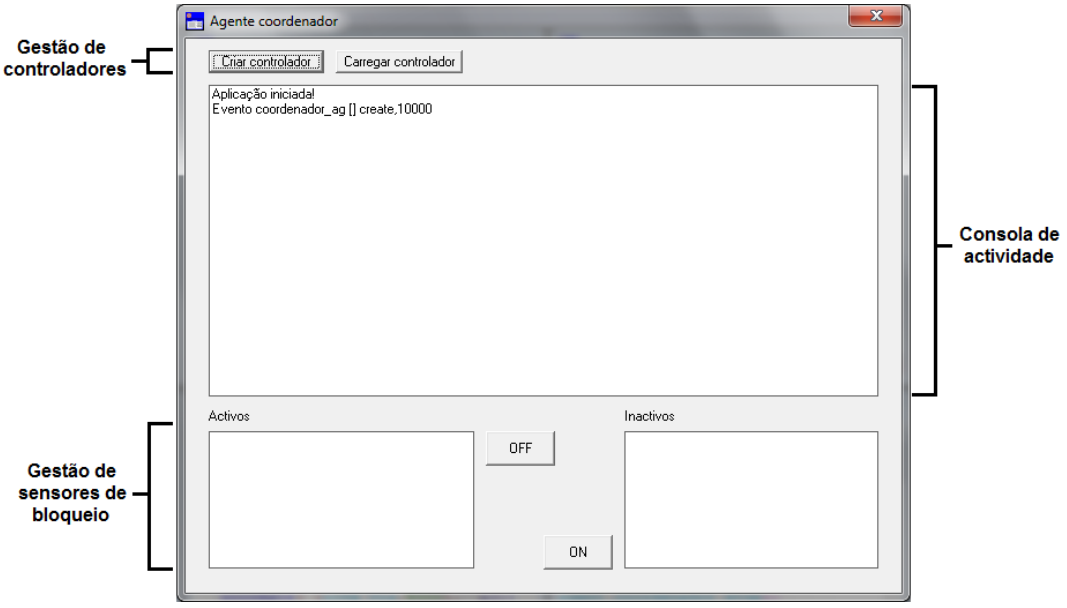

Figura 7.2: Janela principal da aplicação de demonstração

adição de novas vias, bloqueios ou vizinhos, são mostradas as respectivas janelas gráficas que se podem observar na figura 7.4.

Por outro lado, ao seleccionar "Carregar controlador" é possível indicar a localização do ficheiro correspondente ao controlador que se pretende carregar. Após realizar este procedimento, é lançada a respectiva janela de controlador para acompanhar o estado do mesmo.

Consola de actividade A consola de actividade é representada por uma caixa de texto no centro da janela e possibilita a visualização de todas as ocorrências durante a execução do sistema. As ocorrências são maioritariamente registos de comunicações dentro do sistema. Serve para simular a recepção e envio de mensagens entre os vários agentes que estão em funcionamento e poderá ser encarada como a escrita na base de conhecimento do coordenador (a base de conhecimento global do sistema).

Gestão de sensores de bloqueio Esta característica foi criada apenas para

esta demonstração e não deverá estar presente em eventuais desenvolvimentos da solução proposta neste trabalho. Num funcionamento real de um sistema deste tipo, os sensores serão activados pelos veículos que circulam dentro da zona abrangida mas para esta demonstração, determinou-se que era mais simples dar a possibilidade de "activar" os únicos sensores presentes, os sensores de bloqueio. Estes sensores servem para indicar o bloqueio de uma via de trânsito e são equivalentes aos denominados sensores de fim de fila descritos no trabalho.

Ao criar ou carregar controladores, são adicionados na caixa da direita os respecivos sensores de bloqueio que estão por defeito desligados. Os sensores podem então ser seleccionados e ligados, passando para a caixa da esquerda. Esta ocorrência é transmitida ao agente respectivo para que este possa bloquear a via pretendida.

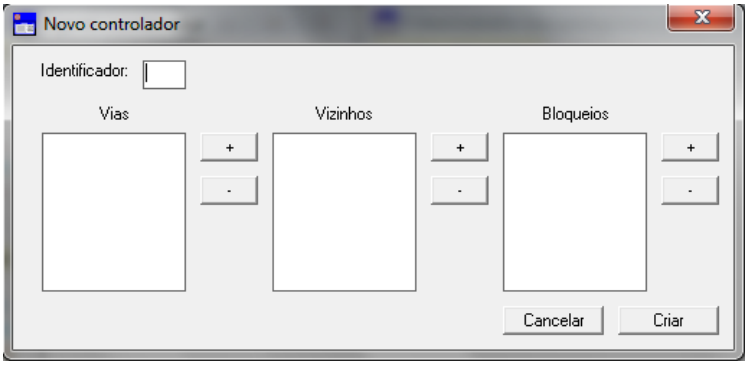

Figura 7.3: Janela de criação de um novo controlador

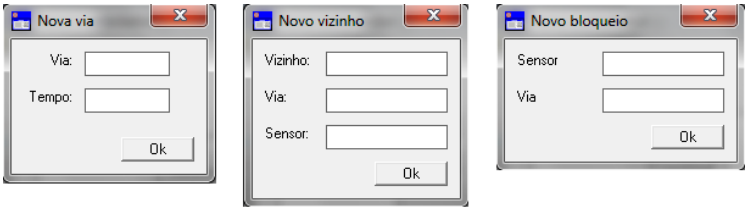

Figura 7.4: Janelas suplementares para criação de elementos de um novo controlador

Sempre que é criado ou carregado um controlador, é mostrada uma nova janela (uma por cada controlador) com o aspecto da figura 7.5 e cujo código fonte está na lista A.2. Aqui é possível visualizar e acompanhar o estado de funcionamento de um controlador, sabendo quais as características do mesmo, qual a via que está aberta em cada momento e qual o tempo da mesma.

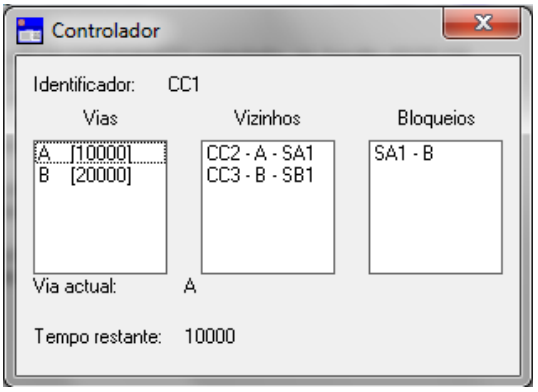

Figura 7.5: Janela de estado de funcionamento de um controlador

Em funcionamento normal, a aplicação terá o aspecto descrito na figura 7.6 (código fonte nas listas A.4, A.5, A.6) onde é possível observar um controlador com a respectiva janela de estado e onde é possível ver igualmente a janela principal com os sensores de bloqueio disponíveis e com a consola de actividade onde estão a ser mostrados as mensagens que são enviadas pelo controlador.

De referir que cada controlador tem um identificador que deverá ser único pois desta designação depende o próprio endereço do controlador, permitindo que este comunique com as restantes entidades.

7.2 Agentes

Nesta aplicação de demonstração, para além dos interfaces gráficos disponíveis, foram implementados agentes de software através da utilização dos predicados disponibilizados com o módulo *chimera*(26), incluído na instalação do *winpro-*

log. Este módulo permite a criação e utilização de agentes através de predicados específicos e podemos observar a sua utilização na criação do agente coordenador, na lista A.7.

O agente coordenador, está encarregue da criação e manutenção de todos os outros agentes do sistema. É a partir deste que são criados os agentes controladores e é igualmente através desta entidade que os agentes controladores estabelecem a comunicação que permite a existência de um conhecimento global de tudo o que se passa no sistema.

O módulo *chimera* permite uma série de particularidades como por exemplo, a comunicação (através de *sockets TCP/IP*) e o tratamento de eventos. Os eventos acabam por ser o motor de funcionamento dos agentes já que possibilita o agendamento de tarefas a executar em cada agente. Estes eventos são tratados através de *handlers* na programação de cada agente.

No agente coordenador, existem tratamentos para vários tipos de eventos, provenientes dos agentes controladores e existem também predicados de envio de eventos para os mesmos controladores. É sempre possível, na lógica de funcionamento dos agentes com a utilização do módulo *chimera*, agendar eventos no próprio agente que os despoleta.

A lista A.8 mostra o código de desenvolvimento dos agentes controladores, que procedem ao tratamento das comunicações e eventos respectivos.

7.3 Comunicação

Como já foi referido, a comunicação entre os diversos agentes é realizada com recurso às potencialidades do módulo *chimera* e através de *sockets TCP/IP*. Cada agente existente no sistema está pronto a comunicar através de uma determinada porta de comunicação, aberta na altura da sua criação. Sempre

que um agente pretende comunicar com outro, fá-lo recorrendo a essa porta e criando os denominados *links*, caracterizados por valores inteiros que são guardados na base de conhecimento dos agentes.

Ao pretender agendar um evento, o agente emissor envia um predicado através da comunicação disponível, identificando-se a si próprio assim como a porta e o *link* para o agente receptor. O funcionamento será semelhante ao de um túnel de comunicação por onde são passados os dados pretendidos.

A título de exemplo, quando um agente controlador abre uma via, comunica esse facto ao agente controlador, utilizando a ligação entre os dois e agendando um evento do tipo "protocol(nova_via, Client, Via)". Este evento é tratado pelo coordenador com o predicado "coordenador_ag_handler(Name, Link, protocol(nova_via, Client, Via))".

Nesta demonstração de utilização, é possível perceber que as comunicações entre agentes são tratadas da forma mais simples possível, através da utilização de *sockets* que tornam esta parte da aplicação totalmente abstracta relativamente ao utilizador. Não existe a necessidade de preocupação quanto à forma como a comunicação é estabelecida, mas apenas ao que é transmitido e recebido.

Como tivemos oportunidade de verificar pelo exemplo acima descrito, apesar de um agente poder enviar tudo aquilo que pretender através da comunicação que estabelece com outro agente, é de todo conveniente que o outro agente entenda o que foi transmitido e o que fazer com essa informação. Para isso, foram criados os denominados *handlers* que existem em cada um dos agentes que estão prontos a receber dados.

No coordenador, os *handlers* são predicados com a estrutura - *coordenador_ag_handler(Name, Link, Argumento)* - e de acordo com a mesma, cada um destes predicados recebe como parâmetro o nome do agente, o *link*

utilizado para a comunicação e o argumento que vai ser tratado. Os argumentos possíveis são:

protocol(login, Client, P, Port, List) Ao receber este argumento, o coordenador fica com a indicação de que um controlador ligou-se ao sistema e está a dar esse facto a conhecer ao coordenador, identificando-se. O coordenador insere o controlador na lista de controladores activos.

protocol(nova_via, Client, Via) Sempre que um controlador muda a via que está activa (ou reinicia a via actual), envia este tipo de argumento para o coordenador que regista o facto na base de conhecimento.

protocol(log, Msg) Este argumento serve genericamente para os controladores enviarem uma mensagem (genérica ou de informação) para o coordenador. A mensagem é então registada no *log*.

(close, Host, Port) Quando um controlador se desliga do sistema, envia este argumento para o coordenador que assim fica com a indicação que isso aconteceu, retirando o controlador da lista de controladores activos.

(error,Code,X) Quando ocorrem erros de comunicação (erros de *socket*), o próprio sistema envia argumentos previamente programados e que são tratados pelo coordenador. Valores possíveis para X: sck_close, sck_read, sck_write, sck_accept, sck_connect.

Quanto aos *handlers* utilizados nos controladores, têm a estrutura - *controlador_ag_handler(Name,Link, Argumento)* - e à semelhança do coordenador existem 3 parâmetros, o nome do agente, o *link* utilizado na comunicação e um argumento para proceder ao tratamento do mesmo. Os argumentos possíveis no controlador são:

protocol(close) Quando um controlador recebe este argumento, recebe a indicação de que deverá desligar-se do sistema e encerrar o seu funcionamento.

protocol(sensor_activate, Sens) Quando um sensor é activado, o controlador recebe este argumento e procede à adição desse facto na sua base de conhecimento.

protocol(sensor_deactivate, Sens) Por outro lado, quando um sensor é desactivado, o controlador recebe este argumento e procede às actualizações necessárias.

Dado tratar-se de um exemplo de demonstração, os agentes emissores sabem antecipadamente o formato de mensagens que os receptores estão prontos a receber. Numa implementação real, isto não acontecerá já que, ao invés de utilizar predicados para envio e recepção de mensagens, será utilizada uma linguagem conhecida por ambas as partes (por exemplo, KQML).

Só não foi utilizada uma linguagem do género neste exemplo porque o módulo *chimera* permite uma exemplificação de capacidades e utilização de agentes muito mais simples, possibilitando a criação dos mesmos juntamente com todo o tratamento da parte de comunicações e permitindo também o envio de argumentos de um agente para outro como uma simples troca de mensagens entre duas partes. Apenas restou proceder ao tratamento dessas mensagens.

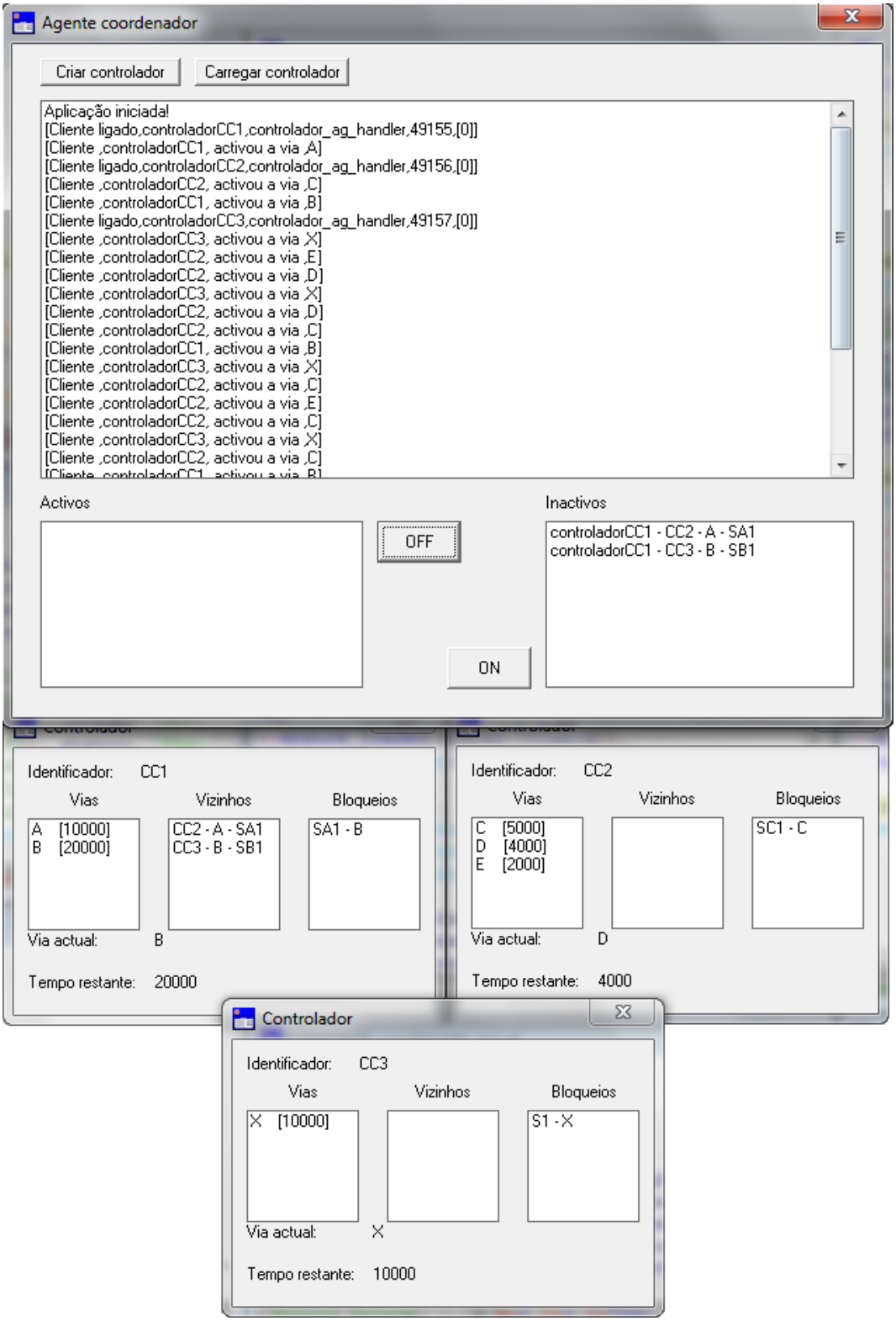

Figura 7.6: Funcionamento normal da aplicação com a janela principal, eventos e controlador

Capítulo 8

Conclusões

Os vários sistemas de gestão de tráfego demonstram características inerentes a um sistema de apoio à decisão, sendo que na sua maioria possuem uma componente fortemente reactiva já que se atribui ao sistema a capacidade não só de concluir sobre os dados processados como também de formar e executar decisões com base nessas mesmas conclusões. Somos no entanto de opinião que o trabalho posterior, normalmente realizado por analistas, tem uma importância relevante e pretende-se assim complementar os sistemas existentes com a capacidade de agir sobre eventos futuros e que poderão ser previstos com base na referida análise.

Existem vários métodos para concluir sobre a necessidade de semaforizar uma intersecção(22) e sobre quais as alterações a realizar na mesma depois desta estar em funcionamento. Podemos afirmar que, na grande maioria dos casos, a determinação da necessidade de semaforização e o tipo de programação a aplicar depende da experiência dos técnicos e da análise que cada um faz dos casos. O mesmo será dizer que, se um técnico conclui algo para determinada intersecção, outro técnico poderá chegar a uma conclusão totalmente diferente. Este é mais um motivo para parametrizar a análise a realizar para poder decidir

de forma mais apoiada e com bases de facto para o fazer.

De igual modo, as alterações que, não sendo de longo curso, são necessárias ao funcionamento diário de um sistema de controlo de semáforos, actualmente são realizadas com frequências demasiado elevadas e inaceitáveis para estabelecer um funcionamento considerado eficaz. Normalmente, um ou mais técnicos procedem a observações com determinada periodicidade, concluindo depois se é necessário realizar alterações. Mais uma vez, estas alterações poderão depender do ponto do vista da pessoa que as concluiu. O ideal seria deixar as conclusões para o próprio sistema para que este evolua de forma crescente no tempo.

Ao propor um sistema multi-agente para substituir o sistema actual de controlo centralizado de semáforos, estamos a dar a possibilidade de corrigir tudo aquilo que é referido acima como problemas ou limitações actuais. O sistema passa a ser composto por uma série de entidades (agentes) que conseguem realizar algum tipo de raciocínio, analisando e concluindo sempre no sentido de melhorar a fluidez do trânsito. As alterações necessárias passam a ser realizadas em tempo real, muito antes de qualquer técnico sequer se aperceber que existia a necessidade de alterar qualquer coisa.

Um dos maiores problemas que ficariam efectivamente resolvidos é o das comunicações já que, na falta destas, cada controlador afectado torna-se numa entidade em situação de "abandono" e passa a constituir um obstáculo ao funcionamento normal do sistema. Com a solução proposta, os agentes continuam a reagir às occorrências indicadas pelos sensores, ficando apenas limitados na partilha de informação com os restantes agentes. Para além disto, a perda de comunicações apenas afectará os agentes directamente atingidos e não, como acontece actualmente, todos os que estão no enfiamento da ligação.

Em suma, tendo iniciado como sistemas bastante simples, ao nível do fun-

cionamento de sistemas electro-mecânicos e que depois evoluiram para sistemas electrónicos, o próximo passo lógico na evolução de sistemas de controlo de semáforos (e principalmente em sistemas centralizados onde a cooperação é fulcral), considerando as tecnologias disponíveis actualmente, será o de sistemas baseados em agentes ou, mais especificamente, multi-agentes que disponibilizam maior capacidade de análise e processamento ou que exibem características consideradas tão essenciais como a capacidade de comunicação com os seus pares ou a capacidade de negociar a resolução de um objectivo.

Capítulo 9

Limitações e trabalho futuro

A solução aqui apresentada ainda apresenta algumas limitações que poderão eventualmente ser consideradas como alvo de um trabalho a realizar futuramente:

Funcionamento do coordenador O coordenador é uma entidade central e cujo papel é fundamental para manter e assegurar uma ligação entre todos os agentes. Para além disto, a grande contribuição do coordenador passa pela análise mais exaustiva dos dados provenientes de todo o sistema pelo que devem ser elaborados os respectivos procedimentos, parametrizando-os de forma a tornar o estudo dos dados mais eficaz.

Com um estudo que obedece a determinados parâmetros e condições, é mais fácil proceder à aceitação do mesmo por parte de técnicos que eventualmente o visualizem e possam ter opiniões diferentes. Assegurando uma base de factos sólida e que consiga de alguma forma demonstrar o porquê de determinadas soluções propostas, fica assegurada a aceitação dessas mesmas conclusões.

Fica desta forma declarado que uma das limitações do trabalho passa

pela falta de parametrização dos estudos a realizar pelo coordenador.

Algoritmos de negociação Apesar de existirem exemplos de negociação entre agentes controladores, faltam determinar algoritmos sobre as eventuais negociações entre o agente coordenador e os agentes controladores. As negociações entre agentes, sendo uma questão pertinente e importante na utilização de agentes, deverá ser mais explorada e os respectivos algoritmos elaborados para melhorar substancialmente o funcionamento do sistema.

Utilização de linguagem própria No exemplo de aplicação descrito neste trabalho, é mostrado como é possível comunicar entre agentes mas numa implementação real deste tipo de sistema, teria que ser assegurada a utilização de uma linguagem própria para a comunicação entre agentes. Existem alguns tipos de linguagem, como o KIF e o KQML que poderão ser utilizados mas fica aqui a referência para a falta dessas descrições de utilização e respectivas implementações.

No exemplo dado, cada agente comunicava com os restantes através da utilização de predicados préviamente conhecidos o que, apesar de prático e simples para uma demonstração, é uma limitação para uma eventual implementação e deve ser evitado.

Bibliografia

[1] , M R Genesereth and S P Ketchpel, *Software Agents*, Communications of the ACM, 1994

[2] , S Franklin and A Graesser, *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents*, Proc. of the Third International Workshop on Agent Theories, Architectures and Languages, 1996

[3] , Stuart J. Russel and Peter Norvig, *Artificial Intelligence: A Modern Approach*, 1995

[4] , Michael Wooldridge and Nicholas R. Jennings, *Intelligent agents: Theory and practice*, The Knowledge Engineering Review, 1995

[5] , SFIM, Adquirida pela SAGEM Avionics, Inc, `http://www.sagemavionics.com`

[6] , GERTRUDE, SAEM, *Gestion Electronique de Régulation en Temps Réel pour l'Urbanisme, les Déplacements et l'Environnement*, Bordeaux, France, `http://www.gertrude.fr`

[7] , GERTRUDE, SAEM, *Exploitation et Régulation*, Bordeaux, France, `http://www.gertrude.fr/index.php?option=com_content&task=view&id=11&Itemid=11&lang=fr`

[8] , SAGEM Avionics, Inc, Safran group, Texas, U.S.A., http://www.sagemavionics.com

[9] , Michael Salgueiro and Nuno Gonçalo Ferreira and Pedro Godinho, *Controlo de tráfego automóvel: Sistema Gertrude*, Licenciatura em Engenharia Electrotécnica e Computadores, 2004

[10] , Município de Lisboa, Lisboa, Portugal, http://www.cm-lisboa.pt

[11] , Município de Lisboa, Ampliação do sistema gertrude, Lisboa, Portugal, http://www.cm-lisboa.pt/?idc=42&idi=33480

[12] , Município de Lisboa, *Segurança rodoviária em Lisboa e referências ao sistema centralizado*, Lisboa, Portugal, http://www.cm-lisboa.pt/?idc=42&idi=33989

[13] , Município de Lisboa, *Temporização dos semáforos e direitos dos peões*, Lisboa, Portugal, http://www.cm-lisboa.pt/archive/doc/Carta_Municipal_dos_Direitos_dos_Peoes_Temporizacao_dos_Semaforos.ppt

[14] , Município do Porto, Porto, Portugal, http://www.cm-porto.pt

[15] , Federal HighWay Administration, U.S. Department of Transportation, http://www.fhwa.dot.gov

[16] , Federal HighWay Administration, *Traffic detector handbook*, U.S. Department of Transportation, http://www.fhwa.dot.gov/tfhrc/safety/pubs/Ip90002/intro.htm

[17] , Federal HighWay Administration, *Benefícios de detectores de fim de fila*, U.S. Department of Transportation, http://www.tfhrc.gov/focus/archives/Fcs498/048chips.htm

[18] , Federal HighWay Administration, *Changes in Traffic Control Devices to Help Older Drivers, Pedestrians, Bicyclists, Workers*, U.S. Department of Transportation, http://www.fhwa.dot.gov/pressroom/fhwa0334.htm

[19] , Highway Agency, MOVA System, Leeds, U.K., http://www.highways.gov.uk/news/pressrelease.aspx?pressreleaseid=138486

[20] , Utah Department of Public Safety, *ITS - Intelligent Transportation Systems*, Utah, U.S.A, http://www.cisusa.org/case_studies/utah.php

[21] , ISM-Project, Frankfurt Rhein-Main, Deutchland, http://www.momatec.de/fileadmin/documents/publications/Pub_Kirschfink-Riegelhuth-Barcelo_ISM_ITS-Madrid_2003.pdf

[22] , Maria Elisabeth Pinheiro Moreira and Waldemiro de Aquino Pereira Neto, *Modelo multicritério para a priorização de interseções candidatas a implantação de controle semafórico*, Departamento de Engenharia de Transportes, Universidade Federal do Ceará, Brasil, http://redpgv.coppe.ufrj.br/arquivos/Controle_Semaforico_Hierarquizacao_Impant.pdf

[23] , Vânia Barcellos Gouvêa Campos and Altair dos Santos Ferreira Filho, *Um procedimento de apoio à decisão para escolha de sistemas de controle visando o planejamento do tráfego*, Mestrado em Engenharia de Tansportes, Instituto Militar de Engenharia, http://aquarius.ime.eb.br/~webde2/prof/vania/pubs/(10)Dispositivosdecontrole.pdf

[24] , Federal HighWay Administration, *Manual on Uniform Traffic Control Devices*, U.S. Department of Transportation, http://mutcd.fhwa.dot.gov/pdfs/2003/pdf-index.htm

[25] , Logic Programming Associates Ltd, *LPA WIN-PROLOG*, `http://www.lpa.co.uk/win.htm`

[26] , Logic Programming Associates Ltd, *Chimera Agents for WIN-PROLOG*, `http://www.lpa.co.uk/chi.htm`

Anexos

Apêndice A

Código fonte

A.1 Interfaces gráficos

Listing A.1: Janela principal

```
:-consult(coordenador_ag).
:-dynamic coordenador_port/1.
:-dynamic vizinho/3.

coordenador :-
    _S1 = [ws_sysmenu,ws_popup,ws_caption,
        dlg_ownedbyprolog],
    _S2 = [ws_child,ws_tabstop,ws_visible,bs_pushbutton,
        bs_text,bs_center,bs_vcenter],
    _S3 = [ws_child,ws_border,ws_tabstop,ws_visible,
        ws_vscroll,lbs_nointegralheight,lbs_nosel],
    _S4 = [ws_child,ws_visible,ws_border,ws_tabstop,
        ws_vscroll,lbs_sort],
    _S5 = [ws_child,ws_visible,ss_left],
```

```
wdcreate( coordenador, 'Agente coordenador',
          187, 60, 626, 508, _S1 ),
wccreate( (coordenador,1000), button, 'Criar
  controlador', 20, 10, 100, 20, _S2 ),
wccreate( (coordenador,4000), listbox, '',
          20, 40, 580, 270, _S3 ),
wccreate( (coordenador,1001), button, 'Carregar
  controlador', 130, 10, 110, 20, _S2 ),
wccreate( (coordenador,4001), listbox, 'List2',
          20, 340, 230, 130, _S4 ),
wccreate( (coordenador,4002), listbox, 'List3',
          380, 340, 220, 130, _S4 ),
wccreate( (coordenador,1002), button, 'OFF',
          260, 340, 60, 30, _S2 ),
wccreate( (coordenador,1003), button, 'ON',
          310, 430, 60, 30, _S2 ),
wccreate( (coordenador,11000), static, 'Activos',
          20, 320, 230, 20, _S5 ),
wccreate( (coordenador,11001), static, 'Inactivos',
          380, 320, 220, 20, _S5 ),
window_handler(coordenador, coordenador_handler),
show_dialog(coordenador),
coordenador_cleanup,
loggerCoordenador('Aplicação iniciada!'),
agent_coordenador_start,
%carrega a lista de sensores existentes
```

```
    init_sensores.

% cleanup dos predicados temporários
coordenador_cleanup:-
        retractall(f_controlador(Q1, Q2, Q3, Q4)),
        clean_retract.

% Confirmação do fecho da janela
coordenador_handler(coordenador, msg_close, _, _):-
        confirmarSaida('Tem a certeza que deseja fechar
            a aplicação?', 'Coordenador', R),
        R == 1
        ->
                (
                        % envia comunicação de fecho do
                            servidor
                        broadcast_close_message,
                        % agenda fecho da janela
                            principal para dar tempo aos
                            eventos de dispararem
                        timer_create(timer_stop,
                            timer_handler),
                        timer_set(timer_stop, 1000)
                );
                % colocar o focus na janela actual
                wfocus(coordenador).
```

```
% evento do temporizador de fecho da janela do servidor
timer_handler(Dummy, Status):-
        agent_coordenador_stop,
        wclose(coordenador).

% WM_COMMAND
% (coordenador,1000) Criar controlador
coordenador_handler((coordenador, 1000), msg_button, _,
   _):-
        novo_controlador,
        % obter os dados do novo controlador
        fact_alocate_cleanup(ID),
        % obter a porta do servidor
        coordenador_port(Port),
        % iniciar a janela do controlador
        controlador(ID, '127.0.0.1', Port),
        % actualizar os sensores
        init_sensores.

% (coordenador,1001) Carregar controlador
coordenador_handler((coordenador, 1001), msg_button, _,
   _):-
        clean_retract,
        opnbox('Controlador a carregar', [('Ficheiros do
            tipo controlador', '*.ctrl')], '', 'ctrl', [
```

```
            File|_]),
        loadOpenFile(File, Res),
        Res == 1 ->
        (
                % obter os dados do novo controlador
                fact_alocate_cleanup(ID),
                % obter a porta do servidor
                coordenador_port(Port),
                % iniciar a janela do controlador
                controlador(ID, '127.0.0.1', Port),
                % actualizar os sensores
                init_sensores
        ).

% (coordenador,1002) OFF
coordenador_handler((coordenador, 1002), msg_button, _,
    _):-
        wlstsel((coordenador,4001), SelectedPos),
        SelectedPos > -1 ->
        (
                wlstget( (coordenador,4001), SelectedPos
                    , String, Item ),
                wlstadd((coordenador,4002), -1, String,
                    Item),
                wlstdel((coordenador,4001),SelectedPos),
```

```
                desactivar_sensor(String, Controlador,
                    Vizinho, Via, Sensor),
                send_to_controlador(off, Controlador,
                    Vizinho, Via, Sensor)

        ); true.

% (coordenador,1003) ON
coordenador_handler((coordenador, 1003), msg_button, _,
    _):-
wlstsel((coordenador,4002), SelectedPos),
        SelectedPos > -1 ->
        (
                wlstget( (coordenador,4002), SelectedPos
                    , String, Item ),
                wlstadd((coordenador,4001), -1, String,
                    Item),
                wlstdel((coordenador,4002), SelectedPos)
                    ,
                activar_sensor(String, Controlador,
                    Vizinho, Via, Sensor),
                send_to_controlador(on, Controlador,
                    Vizinho, Via, Sensor)
        ); true.
```

```
% permite colocar os predicados temporarios num unico
    predicado
fact_alocate_cleanup(ID):-
        info(ID),
        lwrite_vias(Vias),
        lwrite_vizinhos(Vizinhos),
        lwrite_bloqueios(Bloqueios),
        (write('controlador'), write(ID))~>
            ControladorStr,
      atom_string(Controlador, ControladorStr),
        asserta(f_controlador(Controlador, Vias,
            Vizinhos, Bloqueios)),
        retractall(via(C, W, U)),
        retractall(controlo(via, size, _)),
        retractall(vizinho(Q2, Q7, Q8)),
        retractall(bloqueio(Q4, Q5)),
        retractall(info(Q3)).

% permite limpar uma listbox eliminando todas as linhas
lstClear(List):-
        wlstcount(List, Count),
        wpredicatefor(List, 0, Count).

% implementação de um predicado for = for(Var, Valor,
    Maximo)
wpredicatefor(_, Max, Max).
```

```
wpredicatefor(ListBoxID, Actual, Max):-
        wlstdel(ListBoxID, 0),
        Current is Actual + 1,
        wpredicatefor(ListBoxID, Current, Max).

% permite actualizar a lista de sensores
init_sensores:-
      lstClear((coordenador,4002)),
      lstClear((coordenador,4001)),
        findall( v(Controlador, Vizinhos) ,
           f_controlador(Controlador, Vias, Vizinhos,
           Bloqueios), L),
        write(L),
        init_list(L).
init_sensores.

% itera os vizinhos
init_list([]).
init_list([v(Controlador, Vizinhos)|R]):-
        init_list_vizinho(Controlador, Vizinhos),
        init_list(R).

init_list_vizinho(C, []).
init_list_vizinho(C, [v(A, B, D)|R]):-
        (write(C), write('-'), write(A), write('-'),
            write(B), write('-'), write(D))~>Msg,
```

```
        wlstadd((coordenador,4002), -1,Msg, 0),
        init_list_vizinho(C, R).



% itera os predicados temporarios para se posicionar no
   Controlador actual, depois invoca o predicado dos
   vizinhos
init_list_find([], _, _, _, _, _).
init_list_find([v(Controlador, Vizinhos)|R], StringFind,
    C, A, B, D):-
        str_contains(Controlador, StringFind, Result),!,
        Result == 0 ->
        (
                init_list_find(R, StringFind, C, A, B, D
                    )
        );
        (
                init_list_find_vizinho(Controlador,
                    Vizinhos, StringFind), !,
                vizinho_found(C, A, B, D),
                retract(vizinho_found(C, A, B, D))
        ).

% itera os vizinhos de forma a verificar se a string de
   pesquisa equivale ao vizinho actual, preenche um
   predicado temporario
```

```
init_list_find_vizinho(C, [], _):-write('sem vizinhos
    para testar'),nl.
init_list_find_vizinho(C, [v(A, B, D)|R], StringFind):-
        (write(C), write(' - '), write(A), write(' - '),
            write(B), write(' - '), write(D))~>Msg,
        str_contains(Msg, StringFind, Result),!,
        Result == 0 -> (
                init_list_find_vizinho(C, R, StringFind)
        );
        asserta(vizinho_found(C, A, B, D)),
        true.

% desactivar um sensor
desactivar_sensor(String, Controlador2, Vizinho2, Via2,
    Sensor2):-
        findall( v(Controlador, Vizinhos) ,
        f_controlador(Controlador, _, Vizinhos, _), L),
        init_list_find(L, String, Controlador2, Vizinho2
            , Via2, Sensor2).

% activar um evento
activar_sensor(String, Controlador2, Vizinho2, Via2,
    Sensor2):-
        write(String),nl,
        findall( v(Controlador, Vizinhos) ,
        f_controlador(Controlador, _, Vizinhos, _), L),
```

```
        init_list_find(L, String, Controlador2, Vizinho2
            , Via2, Sensor2).
```

Listing A.2: Janela de estado do agente controlador

```
:-dynamic serverLink/2.
:-dynamic controladorPort/2.

% criação de um novo controladro, passando o ID, o
    endereço do servidor e a porta do mesmo
controlador(ID, ServerAddress, ServerPort) :-
    _S1 = [ws_caption,ws_sysmenu,dlg_ownedbyprolog],
    _S2 = [ws_child,ws_visible,ss_left],
    _S3 = [ws_child,ws_border,ws_tabstop,ws_visible,
        ws_vscroll,lbs_nointegralheight,lbs_nosel],
    _S4 = [ws_child,ws_visible,ss_center],
    (write('controlador'), write(ID))~> ControladorStr,
    atom_string(Controlador, ControladorStr),
    wdcreate( Controlador, 'Controlador',
                    203, 69, 306, 218, _S1 ),
    wccreate( (Controlador,11000), static, '
        Identificador:', 10, 10, 70, 20, _S2 ),
    wccreate( (Controlador,11001), static, '',
                        90, 10, 80, 20, _S2 ),
    wccreate( (Controlador,4000), listbox, '',
                        10, 50, 80, 80, _S3 ),
    wccreate( (Controlador,4001), listbox, '',
                       110, 50, 80, 80, _S3 ),
```

```
    wccreate( (Controlador,11002), static, 'Vias',
                    10, 30, 80, 20, _S4 ),
    wccreate( (Controlador,11003), static, 'Vizinhos',
              110, 30, 80, 20, _S4 ),
    wccreate( (Controlador,11004), static, 'Via actual
       :',       10, 130, 80, 20, _S2 ),
    wccreate( (Controlador,11005), static, 'Tempo
       restante:', 10, 160, 80, 20, _S2 ),
    wccreate( (Controlador,11006), static, '',
                      100, 130, 80, 20, _S2 ),
    wccreate( (Controlador,11007), static, '',
                      100, 160, 80, 20, _S2 ),
    wccreate( (Controlador,4002), listbox, '',
                      210, 50, 80, 80, _S3 ),
    wccreate( (Controlador,11008), static, 'Bloqueios',
              210, 30, 80, 20, _S4 ),
    window_handler(Controlador, controlador_handler),
    show_dialog(Controlador),
    init_window(ID, Controlador),
    agent_controlador_start(Controlador, ServerAddress,
       ServerPort),
    start_generator_algorithm(Controlador).

% carrega os dados e coloca-os visiveis nos respectivos
    controlos
init_window(ID, Controlador):-
```

```
        wtext( (Controlador,11001), ID),
        f_controlador(Controlador, Vias, Vizinhos,
            Bloqueios),
        assign_vias(Vias, Controlador),write('vias'),
        assign_vizinhos(Vizinhos, Controlador),
        assign_bloqueios(Bloqueios, Controlador).
% failover, caso ocorra algum erro durante o load
init_window(_).

% itera as vias, adicionando à respectiva lista
assign_vias([], _).
assign_vias([v(Via, Tempo, _)|R], Controlador):-
        (write(Via), write('    ['), write(Tempo), write
            (']'))~>Msg,
        wlstadd((Controlador,4000), -1,Msg , 0),
        assign_vias(R, Controlador).

% itera os vizinhos, adicionando à respectiva lista
assign_vizinhos([], _).
assign_vizinhos([v(Vizinho, Via, Sens)|R], Controlador)
    :-
        (write(Vizinho), write(' - '), write(Via), write
            (' - '), write(Sens))~>Msg,
        wlstadd((Controlador,4001), -1,Msg , 0),
        assign_vizinhos(R, Controlador).
```

```
% itera os bloqueios, adicionando à respectiva lista
assign_bloqueios([], _).
assign_bloqueios([v(Sens, Via)|R], Controlador):-
        (write(Sens), write('-'), write(Via))~>Msg,
        wlstadd((Controlador,4002), -1,Msg , 0),
        assign_bloqueios(R, Controlador).

% Handler do fecho da janela
controlador_handler(Controlador, msg_close, _, _):-
        agent_controlador_stop(Controlador),
        wclose(Controlador).
```

Listing A.3: Janela de criação de novo agente controlador

```
:- consult(controlador_ag).

% Window novo agente
novo_controlador :-
    _S1 = [ws_sysmenu,ws_popup,ws_caption,
       dlg_ownedbyprolog],
    _S2 = [ws_child,ws_visible,ss_right],
    _S3 = [ws_child,ws_visible,ws_tabstop,ws_border,
       es_left,es_multiline,es_autovscroll,es_autohscroll
       ],
    _S4 = [ws_child,ws_border,ws_tabstop,ws_visible,
       ws_vscroll,lbs_nointegralheight,lbs_nosel],
    _S5 = [ws_child,ws_tabstop,ws_visible,bs_pushbutton,
       bs_text,bs_center,bs_vcenter],
```

```
_S6 = [ws_child,ws_visible,ws_tabstop,bs_pushbutton],
_S7 = [ws_child,ws_visible,ss_center],
wdcreate( novo_controlador, 'Novo controlador
   ', 187, 60, 506, 238, _S1 ),
wccreate( (novo_controlador,11000), static, '
   Identificador:', 20, 10, 60, 20, _S2 ),
wccreate( (novo_controlador,8000), edit, '',
                90, 10, 30, 20, _S3 ),
wccreate( (novo_controlador,4000), listbox, '',
                20, 60, 100, 110, _S4 ),
wccreate( (novo_controlador,4001), listbox, '',
               180, 60, 100, 110, _S4 ),
wccreate( (novo_controlador,1000), button, '+',
              130, 60, 40, 20, _S5 ),
wccreate( (novo_controlador,1001), button, '-',
              130, 90, 40, 20, _S5 ),
wccreate( (novo_controlador,1002), button, '+',
              290, 60, 40, 20, _S5 ),
wccreate( (novo_controlador,1003), button, '-',
              290, 90, 40, 20, _S5 ),
wccreate( (novo_controlador,1004), button, 'Criar',
            420, 180, 70, 20, _S6 ),
wccreate( (novo_controlador,1005), button, '
   Cancelar', 340, 180, 70, 20, _S6 ),
wccreate( (novo_controlador,11001), static, 'Vias',
             20, 40, 100, 20, _S7 ),
```

```
    wccreate( (novo_controlador,11002), static, '
       Vizinhos', 180, 40, 100, 20, _S7 ),
    wccreate( (novo_controlador,4002), listbox, '',
                    340, 60, 100, 110, _S4 ),
    wccreate( (novo_controlador,11003), static, '
       Bloqueios', 340, 40, 100, 20, _S7 ),
    wccreate( (novo_controlador,1006), button, '+',
                    450, 60, 40, 20, _S5 ),
    wccreate( (novo_controlador,1007), button, '-',
                    450, 90, 40, 20, _S5 ),
    window_handler(novo_controlador,
       novo_controlador_handler),
    call_dialog(novo_controlador, R).

% handlers
% Close
novo_controlador_handler(novo_controlador, msg_close, _,
    R):- wdlgcancel(novo_controlador , R).

% WM_COMMAND
% (novo_controlador,1004) Criar
novo_controlador_handler((novo_controlador,1004),
   msg_button, _, R):-
         wdlgok(novo_controlador , R),
         wtext((novo_controlador,8000), IDT),
         (write(IDT), write('.ctrl'))~>ID,
```

```
        assertz(info(IDT)),
        atom_string(ID2, ID),
        saveOpenFile(ID2),
        saveCloseFile(ID2).

% (novo_controlador,1005) Cancelar
novo_controlador_handler((novo_controlador,1005),
   msg_button, _, R):-
        wdlgcancel(novo_controlador , R).

% (novo_controlador,1000) Adicionar via
novo_controlador_handler((novo_controlador,1000),
   msg_button, _, _):-
        nova_via.

% (novo_controlador,1001) Remover via
novo_controlador_handler((novo_controlador,1001),
   msg_button, _, _):-
        wlstsel((novo_controlador,4000), SelectedPos),
        SelectedPos > -1 ->
        (
                wlstget((novo_controlador,4000),
                   SelectedPos, String, Item),
                wlstdel((novo_controlador,4000),
                   SelectedPos),
                remove_via(String)
```

```
        ); true.

% (novo_controlador,1002) Adicionar vizinho
novo_controlador_handler((novo_controlador,1002),
    msg_button, _, _):-
        novo_vizinho.

% (novo_controlador,1003) Remover vizinho
novo_controlador_handler((novo_controlador,1003),
    msg_button, _, _):-
        wlstsel((novo_controlador,4001), SelectedPos),
        SelectedPos > -1 ->
        (
                wlstget((novo_controlador,4001),
                    SelectedPos, String, Item),
                wlstdel((novo_controlador,4001),
                    SelectedPos),
                remove_vizinho(String)
        ); true.

% (novo_controlador,1004) Adicionar bloqueio
novo_controlador_handler((novo_controlador,1006),
    msg_button, _, _):-
        novo_bloqueio.

% (novo_controlador,1005) Remover bloqueio
```

```
novo_controlador_handler((novo_controlador,1007),
    msg_button, _, _):-
        wlstsel((novo_controlador,4002), SelectedPos),
        SelectedPos > -1 ->
        (
                wlstget((novo_controlador,4002),
                    SelectedPos, String, Item),
                wlstdel((novo_controlador,4002),
                    SelectedPos),
                remove_bloqueio(String)
        ); true.
```

Listing A.4: Janela de inserção de nova via

```
% window nova via
nova_via :-
    _S1 = [ws_caption,ws_sysmenu,dlg_ownedbyprolog],
    _S2 = [ws_child,ws_visible,ws_tabstop,ws_border,
        es_left,es_multiline,es_autovscroll,es_autohscroll
        ],
    _S3 = [ws_child,ws_tabstop,ws_visible,bs_pushbutton,
        bs_text,bs_center,bs_vcenter],
    _S4 = [ws_child,ws_visible,ss_right],
    wdcreate( nova_via, 'Nova via', 187,
        60, 166, 138, _S1 ),
    wccreate( (nova_via,8000), edit, '', 60, 10,
        70, 20, _S2 ),
```

```
    wccreate( (nova_via,8001), edit, '', 60, 40,
        70, 20, _S2 ),
    wccreate( (nova_via,1000), button, 'Ok', 90,
        80, 60, 20, _S3 ),
    wccreate( (nova_via,11000), static, 'Via:', 10,
        10, 40, 20, _S4 ),
    wccreate( (nova_via,11001), static, 'Tempo:', 10,
        40, 40, 20, _S4 ),
window_handler(nova_via, nova_via_handler),
call_dialog(nova_via, _).

% Close
nova_via_handler(nova_via, msg_close, _, R):- wdlgcancel
    (nova_via , R).

% (nova_via,1000) Ok
nova_via_handler((nova_via,1000), msg_button, _, R):-
        adiciona_via,
        wdlgok(nova_via, R).

% (novo_controlador,4000) Lista das vias
% (nova_via,8000) - Via
% (nova_via,8001) - Tempo
adiciona_via:-
        wtext((nova_via,8000), Via),
        wtext((nova_via,8001), Tempo),
```

```
        (write(Via), write('____['), write(Tempo), write
            (']'))~>Msg,
        wlstadd((novo_controlador,4000), -1,Msg , 0),
        get_next_via_size(Size),
        assertz(via(Via, Tempo, Size)).


remove_via(Msg_compare):-
        via(Via, Tempo, Q),
        (write(Via), write('____['), write(Tempo), write
            (']'))~>Msg,
        Msg = Msg_compare,
        retract(via(Via, Tempo, Q)),
        resync(Q).


resync(Q):-
        findall( v(Via, Tempo, N), (via(Via, Tempo, N),
            N > Q), L),
        resynclist(L).


resynclist([]).
resynclist( [v(Via, Tempo, N)|R] ):-
        retract(via(Via, Tempo, N)),
        N1 is N - 1,
        assert(via(Via, Tempo, N1)),
        resynclist(R).
```

Listing A.5: Janela de inserção de novo bloqueio

```
% novo bloqueio
novo_bloqueio :-
    _S1 = [ws_caption,ws_sysmenu,dlg_ownedbyprolog],
    _S2 = [ws_child,ws_visible,ss_left],
    _S3 = [ws_child,ws_border,ws_tabstop,ws_visible,
        es_left,es_multiline,es_autohscroll,es_autovscroll
        ],
    _S4 = [ws_child,ws_tabstop,ws_visible,bs_pushbutton,
        bs_text,bs_center,bs_vcenter],
    wdcreate( novo_bloqueio, 'Novo bloqueio',
        187, 60, 196, 128, _S1 ),
    wccreate( (novo_bloqueio,11000), static, 'Sensor',
        10, 10, 60, 20, _S2 ),
    wccreate( (novo_bloqueio,11001), static, 'Via',
        10, 40, 60, 20, _S2 ),
    wccreate( (novo_bloqueio,8000), edit, '',
        80, 10, 100, 20, _S3 ),
    wccreate( (novo_bloqueio,8001), edit, '',
        80, 40, 100, 20, _S3 ),
    wccreate( (novo_bloqueio,1000), button, 'Ok',
        120, 70, 60, 20, _S4 ),
    window_handler(novo_bloqueio, novo_bloqueio_handler),
    call_dialog(novo_bloqueio, _).

% Close
```

```
novo_bloqueio_handler(novo_bloqueio, msg_close, _, R):-
    wdlgcancel(novo_bloqueio, R).

% (novo_bloqueio,1000) Ok
novo_bloqueio_handler((novo_bloqueio,1000), msg_button,
    _, R):-
        adiciona_bloqueio,
        wdlgok(novo_bloqueio, R).

% (novo_controlador,4000) Lista dos bloqueios
% (novo_bloqueio,8000) - Sensor
% (novo_bloqueio,8001) - Via
adiciona_bloqueio:-
        wtext((novo_bloqueio,8000), Sensor),
        wtext((novo_bloqueio,8001), Via),
        (write(Sensor), write('␣-␣'), write(Via))~>Msg,
        wlstadd((novo_controlador,4002), -1,Msg , 0),
        assertz(bloqueio(Sensor, Via)).

remove_bloqueio(Msg):-
        bloqueio(Sens, Via),
        (write(Sensor), write('␣-␣'), write(Via))~>Msg,
        retract(bloqueio(Sens, Via)).
```

Listing A.6: Janela de inserção de novo vizinho

```
% window novo vizinho
novo_vizinho :-
```

```
    _S1 = [ws_caption,ws_sysmenu,dlg_ownedbyprolog],
    _S2 = [ws_child,ws_visible,ss_left],
    _S3 = [ws_child,ws_border,ws_tabstop,ws_visible,
        es_left,es_multiline,es_autohscroll,es_autovscroll
        ],
    _S4 = [ws_child,ws_visible,ws_tabstop,bs_pushbutton],
    wdcreate( novo_vizinho, 'Novo vizinho',
        187, 60, 176, 158, _S1 ),
    wccreate( (novo_vizinho,11000), static, 'Vizinho:',
        10, 10, 50, 20, _S2 ),
    wccreate( (novo_vizinho,8000), edit, '',
        60, 10, 100, 20, _S3 ),
    wccreate( (novo_vizinho,1000), button, 'Ok',
        100, 100, 60, 20, _S4 ),
    wccreate( (novo_vizinho,11001), static, 'Via:',
        10, 40, 50, 20, _S2 ),
    wccreate( (novo_vizinho,11002), static, 'Sensor:',
        10, 70, 50, 20, _S2 ),
    wccreate( (novo_vizinho,8001), edit, '',
        60, 40, 100, 20, _S3 ),
    wccreate( (novo_vizinho,8002), edit, '',
        60, 70, 100, 20, _S3 ),
window_handler(novo_vizinho, novo_vizinho_handler),
call_dialog(novo_vizinho, _).

% Close
```

```
novo_vizinho_handler(novo_vizinho, msg_close, _, R):-
    wdlgcancel(novo_vizinho, R).

% (nova_via,1000) Ok
novo_vizinho_handler((novo_vizinho,1000), msg_button, _,
    R):-
        adiciona_vizinho,
        wdlgok(novo_vizinho, R).

% (novo_controlador,4001) Lista dos vizinhos
% (novo_vizinho,8000) - Vizinho
adiciona_vizinho:-
        wtext((novo_vizinho,8000), Vizinho),
        wtext((novo_vizinho,8001), Via),
        wtext((novo_vizinho,8002), Sensor),
        (write(Vizinho), write('␣-␣'), write(Via), write
            ('␣-␣'), write(Sensor))~>Msg,
        wlstadd((novo_controlador,4001), -1,Msg , 0),
        assertz(vizinho(Vizinho, Via, Sensor)).

remove_vizinho(Msg_compare):-
        vizinho(Vizinho, Via, Sensor),
        (write(Vizinho), write('␣-␣'), write(Via), write
            ('␣-␣'), write(Sensor))~>Msg,
        Msg = Msg_compare,
        retract(vizinho(Vizinho, Via, Sensor)).
```

A.2 Agentes

Listing A.7: Código do agente coordenador

```
:-dynamic client/3.

% iniciar o agente coordenador na porta 10000 por
   defeito e guardar predicado temporario com a porta
agent_coordenador_start:-
        Port = 10000,
        agent_create(coordenador_ag,
           coordenador_ag_handler, Port),
        asserta(coordenador_port(Port)).

% terminar o agente coordenador e limpar os predicados
   temporários criados
agent_coordenador_stop:-
        retract(coordenador_port(P)),
        retractall(client(C)),
        agent_close(coordenador_ag).

% itera sobre os clientes ligados e comunica o
   encerramento do servidor
broadcast_close_message:-
        findall((Client, Link), client(Client, Link, _),
            List),
        forall( member((C, L), List),
         (
```

```
            agent_post(coordenador_ag, L, protocol(close)
                )
          )
        ).

% failover dos clientes
broadcast_close_message.


% predicado auxiliar para simplificar a invocação
loggerCoordenador(LogMsg):-
        logger((coordenador, 4000), LogMsg).

% predicado auxiliar para simplificar a invocação
loggerCoordenadorList(LogList):-
        loggerList((coordenador, 4000), LogList).

% envia comunicação para activar sensor
send_to_controlador(on, Controlador, Vizinho, Via,
   Sensor):-
        client(_, Link, Controlador),
      agent_post(coordenador_ag, Link, protocol(
         sensor_activate, Sensor)).

% envia comunicação para desactivar sensor
```

```
send_to_controlador(off, Controlador, Vizinho, Via,
    Sensor):-
        client(_, Link, Controlador),
       agent_post(coordenador_ag, Link, protocol(
           sensor_deactivate, Sensor)).

%Protocolo de comunicação
% protocolo de login
coordenador_ag_handler( Name, Link, protocol(login,
    Client, P, Port, List)):-
        loggerCoordenador(['Cliente ligado', Client, P,
            Port, List]),nl,
        % guarda os dados de ligação de um novo cliente
        asserta(client(Name, Link, Client)).

% protocolo de nova via
coordenador_ag_handler( Name, Link, protocol(nova_via,
    Client, Via)):-
        loggerCoordenador( ['Cliente ', Client, '
            activou a via ', Via]).

% protocolo de log
coordenador_ag_handler( Name, Link, protocol(log, Msg))
    :-
        loggerCoordenador(Msg).
```

```
% fecho de um controlador
coordenador_ag_handler( Name, Link, (close, Host, Port))
   :-
        loggerCoordenadorList(['Fechado o cliente', Name
           , 'com o link', Link, '[ Host=', Host,'Porta:
           ', Port, ']']), nl,
        retract(client(Name, Link, Client)).

% fecho de um controlador
coordenador_ag_handler( Name, Link, (close,Port)):-
        loggerCoordenadorList(['Fechado o cliente', Name
           , 'com o link', Link, '[Porta:', Port, ']']),
            nl,
        retract(client(Name, Link, Client)).

% tratamento de erros no socket
coordenador_ag_handler( Name, Link, (close,Host, Port))
   :-              loggerCoordenador('close HOST port
   no agente').
coordenador_ag_handler( Name, Link, (error,Code,
   sck_close)):-   loggerCoordenador('socket closed
   abruptely').
coordenador_ag_handler( Name, Link, (error,Code,sck_read
   )):-    loggerCoordenador('socket read error').
coordenador_ag_handler( Name, Link, (error,Code,
   sck_write)):-   loggerCoordenador('socket read error
```

```
    ').
coordenador_ag_handler( Name, Link, (error,Code,
    sck_accept)):- loggerCoordenador('socket accept
    error').
coordenador_ag_handler( Name, Link, (error,Code,
    sck_connect)):- loggerCoordenador('').
```

Listing A.8: Código do agente controlador

```
:-dynamic active_sensor/2.
:-dynamic via_no_locked/3.

% inicia o agente do controlador e estabelece a ligação
    ao servidor, enviando os respectivos dados de login
agent_controlador_start(Controlador,ServerAddress,
    ServerPort):-
        agent_create(Controlador,controlador_ag_handler,
            Port),
        asserta(controladorPort(Controlador, Port)),
        connect_to_server(Controlador, ServerAddress,
            ServerPort),
        send_login_data(Controlador).

% termina o agente do controlador, termina o
    temporizador de agendamento
agent_controlador_stop(Controlador):-
        retract(serverLink(Controlador, Dummy)),
        retract(controladorPort(Controlador, Port)),
```

```
        timer_close(Controlador),
        agent_close(Controlador).

connect_to_server(Controlador, ServerAddress, ServerPort
    ):-
        agent_create( Controlador, Link, ServerAddress,
            ServerPort ),
        asserta(serverLink(Controlador, Link)).

% inicia o algoritmo de iteração das vias
start_generator_algorithm(Controlador):-
        % obter uma via livre
        get_via_by_id(Controlador, 0, Via, Time, Done),
        % dispoletar o evento de mudança da via actual
        syncroniza_via_actual(Controlador, Via, Time),
        % agendar nova iteração do algoritmo
        rearm_timer(Controlador, Time).

% prossegue com o algoritmo de iteração
continue_generator_algorithm(Controlador, Via, Time):-
        % dispoletar o evento de mudança da via actual
        syncroniza_via_actual(Controlador, Via, Time),
        % agendar nova iteração do algoritmo
        rearm_timer(Controlador, Time).

% permite agendar um novo temporizador
```

```
rearm_timer(Controlador, CurrentTime):-
        % cria temporizador a disparar dentro de
            CurrentTime
        atom_string(CA, CurrentTime),
        number_atom(Time, CA),
        timer_create(Controlador,
            controlador_timer_handler),
        timer_set(Controlador, Time).

% vai disparar o evento do temporizador
controlador_timer_handler(Controlador, Status):-
        f_controlador(Controlador, Vias, _, _),
        length(Vias, L),
        % obter uma via que não esteja sinalizada como
            bloqueada
        get_via_not_blocked(Controlador, L),!,
        % ler a partir do predicado temporário
            via_no_locked a via encontrada
        via_no_locked(Controlador, Via, Time),
        % remover o predicado temporário
        retract(via_no_locked(Controlador, Via, Time)),
        % proseguir com algoritmo
        continue_generator_algorithm(Controlador, Via,
            Time).
```

```
% permite obter uma via não sinalizada para bloqueio, a
    via será devolvida recorrendo a um predicado
    temporário
get_via_not_blocked(Controlador, L):-
        time(1, T), seed(T), R is int(rand(L)), time(1,
            Y), seed(Y),
        get_via_by_id(Controlador, R, Via, Time, Done),
            !,
        asserta(via_no_locked(Controlador, Via, Time)),
        Done == 0 ->
        (
                retract(via_no_locked(Controlador, DD,
                    DDD)),
                get_via_not_blocked(Controlador, L), !
        );
        (true, !).

% predicado para testar se via está livre
get_via_by_id(Controlador, ID_Via, Via, Tempo, 1):-
        f_controlador(Controlador, Vias, _, Bloqueios),
        write(Vias), nl,
        member(v(Via, Tempo, ID_Via), Vias),write(Via),
            nl,
        not(via_bloqueada(Controlador, Via, Bloqueios)),
             !.
```

```
% predicado para testar se via está bloqueada
get_via_by_id(Controlador, ID_Via, Via, Tempo, 0):-
        f_controlador(Controlador, Vias, _, Bloqueios),
        member(v(Via, Tempo, ID_Via), Vias),
        via_bloqueada(Controlador, Via, Bloqueios), !.

% via sinalizada como bloqueada
via_bloqueada(Controlador, Via, Bloqueios):-
        active_sensor(Controlador, Sens),
        member(v(Sens, Via), Bloqueios).

% permite tratar a mudança da via actual, tanto
   visualmente como comunicando com o coordenador
syncroniza_via_actual(Controlador, Via, Time):-
        wtext((Controlador,11006), Via),
        wtext((Controlador,11007), Time),
        send_via_to_server(Controlador, Via).

% Comunica ao coordenador uma nova via
send_via_to_server(Controlador, NovaVia):-
        % obtem do predicado temporario os dados de
            ligação
        serverLink(Controlador, Link),
        agent_post(Controlador, Link, protocol(nova_via,
            Controlador, NovaVia)).
```

```
% Comunica ao coordenador os dados de login
send_login_data(Controlador):-
        % obtem do predicado temporario os dados de
            ligação
        agent_data(Controlador, P, Port, L),
        serverLink(Controlador, Link),
        agent_post(Controlador, Link, protocol(login,
            Controlador, P, Port, L)).

% protocolo de comunicação
% sinalização de fecho do coordenador
controlador_ag_handler(Name,Link, protocol(close)):-
        agent_controlador_stop(Name),
        wclose(Name).

% activar um sensor
controlador_ag_handler(Name,Link, protocol(
    sensor_activate, Sens)):-
        asserta(active_sensor(Name, Sens)).

% desactivar um sensor
controlador_ag_handler(Name,Link, protocol(
    sensor_deactivate, Sens)):-
        retract(active_sensor(Name, Sens)).
```

A.3 Utilitários

Listing A.9: Código de arranque da aplicação

```
:-ensure_loaded(system(chimera)).
:-dynamic vizinho/3.
:-dynamic via/3.
:-dynamic info/1.
:-dynamic bloqueio/2.
:-dynamic f_controlador/4.
:-dynamic controlo/3.

:-consult(controlador_gui).
:-consult(coordenador_gui).
:-consult(novo_controlador_gui).
:-consult(nova_via_gui).
:-consult(novo_bloqueio_gui).
:-consult(novo_vizinho_gui).
:-consult(utilitarios).
```

Listing A.10: Predicados auxiliares e utilitários

```
% declarações dinâmicas
:-dynamic vizinho/3.
:-dynamic via/3.
:-dynamic info/1.
:-dynamic bloqueio/2.
:-dynamic f_controlador/4.
:-dynamic controlo/3.
```

```
% Permite efectuar uma limpeza aos predicados dinâmicos
clean_retract:-
        retractall(info(Q1)),
        retractall(bloqueio(Q7, Q8)),
        retractall(via(Q2, Q3, U)),
        retractall(controlo(via, size, _)),
        retractall(vizinho(Q4, Q5, Q6)).

% failover, caso ocorra algum erro durante a limpeza
clean_retract.

% itera as vias conhecidas
lwrite_vias(L):-
        findall(v(V, T, U), via(V, T, U), L),
        lwrite_via(L).

% itera os vizinhos conhecidos
lwrite_vizinhos(L):-
        findall(v(V, Via, Sens), vizinho(V, Via, Sens),
           L),
        lwrite_vizinho(L).

% itera os bloqueios conhecidos
lwrite_bloqueios(L):-
        findall(v(S, V), bloqueio(S, V), L),
```

```
        lwrite_bloqueio(L).

% escreve as vias para o STDOUT (que pode ser um
   ficheiro)
lwrite_via([]).
lwrite_via([v(V, T, U)|R]):-
        write(V), write(T), write(U), nl,
        lwrite_via(R).

% escreve os vizinhos para o STDOUT (que pode ser um
   ficheiro)
lwrite_vizinho([]).
lwrite_vizinho([v(V, Via, Sens)|R]):-
        write(V), write(Via), write(Sens), nl,
        lwrite_vizinho(R).

% escreve os bloqueios para o STDOUT (que pode ser um
   ficheiro)
lwrite_bloqueio([]).
lwrite_bloqueio([v(S, V)|R]):-
        write(S), write(V), nl,
        lwrite_bloqueio(R).

% apresentação de uma mensagem de confirmação de saída
confirmarSaida( Message, Title, Result):-
        mymsgbox(Message, Title, yesno, Result).
```

```
% definição de uma msgbox de confirmação
mymsgbox(Message, Title, yesno, Result):-
        write(Message) ~> Message2,
        write(Title) ~> Title2,
        msgbox(Title2, Message2, 16'00000024, Res),
        Res == 6
        ->
            Result is 1;
            Result is 0.

% definição de uma msgbox de confirmação
mymsgbox(Message, Title, info, Result):-
        write(Message) ~> Message2,
        write(Title) ~> Title2,
        msgbox(Title2, Message2, 16'00000040, Result).

% Cancelar uma modal dialog
wdlgcancel(Window, Res):-
        Res is 0,
        hide_dialog(Window).

% OK de uma modal dialog
wdlgok(Window, Res):-
        Res is 1,
        hide_dialog(Window).
```

```
% guardar ficheiro com os dados do controlador
saveOpenFile(File):-
        open(File, write),
        tell(File),
        write(':-dynamic info/1.'),nl,
        write(':-dynamic via/3.'),nl,
        write(':-dynamic vizinho/3.'),nl,
        write(':-dynamic bloqueio/2.'),nl,
        nl,
        write_info,
        write_all_vias,
        write_all_vizinhos,
        write_all_bloqueios,
        nl.

% close do ficheiro
saveCloseFile(File):-
        told,
        close(File).

% escita para ficheiro dos predicados
write_info:-
        wtext((novo_controlador,8000), ID),
        write_canonical(info(ID)),write('.'), nl.
```

```
% escita para ficheiro da lista de predicados
write_all_vias:-
        findall(v(V, T, U), via(V, T, U), L),
        write_via(L).

% escita para ficheiro dos predicados
write_via([]).
write_via([v(V, T, U)|R]):-
        write_canonical(via(V, T, U)),write('.'),nl,
        write_via(R).

% escita para ficheiro da lista de predicados
write_all_vizinhos:-
        findall(v(V, Via, Sens), vizinho(V, Via, Sens),
           L),
        write_vizinho(L).

% escita para ficheiro dos predicados
write_vizinho([]).
write_vizinho([v(V, Via, Sens)|R]):-
        write_canonical(vizinho(V, Via, Sens)),write('.'
           ),nl,
        write_vizinho(R).

% escita para ficheiro da lista de predicados
write_all_bloqueios:-
```

```
        findall(v(Sens, Via), bloqueio(Sens, Via), L),
        write_bloqueio(L).

% escita para ficheiro dos predicados
write_bloqueio([]).
write_bloqueio([v(Sens, Via)|R]):-
        write_canonical(bloqueio(Sens, Via)),write('.'),
           nl,
        write_bloqueio(R).

% carregar um ficheiro de um controlador
loadOpenFile(File, Res):-
        file(File, -1, S),
        S == 1
        ->
                (reconsult(File), Res = 1);
                (fail, Res = 0).

% obter o próximo indíce da via
get_next_via_size(NewSize):-
        controlo(via, size, Size),
        nonvar(Size),
        NewSize is Size + 1,
        retractall(controlo(via, size, _)),
        assert(controlo( via, size, NewSize)).
```

```
% caso não exista nenhuma via ainda criada, considera 0
   como sendo o índice actual
get_next_via_size(0):-
        assert(controlo( via, size, 0)).

% Permitir fazer o log de uma lista
loggerList(Window, [Str|LStrs]):-
        loggerList2(Window, LStrs, Str).
% Failover do log para uma lista vazia
loggerList(Window, []):-
        logger(Window, ``).

% iterar os elementos da lista de log
loggerList2(Window, [Str|LStrs], Msg):-
        (write(Msg), write('␣'), write(Str))~>Msg2,
        loggerList2(Window, LStrs, Msg2).
% efectuar o log sobre um elemento da lista
loggerList2(Window, [], Msg):-
        logger(Window, Msg).

% invocar o log com controlo de numero de linhas
logger(Window, LogMsg):-logtoCtrl(Window, LogMsg, 200).

% Adicionar um item à lista de items da listbox
% respeitando o máximo especificado, dando o 'focus'
% ao novo item.
```

```
logtoCtrl(ListBoxID, LogMsg, MaxItems):-
        wiswindow(ListBoxID, Res),
        Res == 1
        ->
                (
                 wlstcount(ListBoxID, N),
                 ((MaxItems > 0,
                  N >= MaxItems)
                  -> (wlstdel(ListBoxID, 0),
                        logtoCtrl(ListBoxID, LogMsg)
                        );
                        logtoCtrl(ListBoxID, LogMsg)
                 )
                );
        write(LogMsg).

logtoCtrl(ListBoxID, LogMsg):-
        write(LogMsg) ~> LogMsg2,
        wlstadd(ListBoxID, -1, LogMsg2, 0),
        wlstcount(ListBoxID, N),
        N2 is N - 1,
        wlstsel(ListBoxID, N2, 1).

% Verificar se é uma janela, utilizando a API do windows
wiswindow(Window, Result):-
        wndhdl(Window, Handle),
```

```
        winapi((user32,'IsWindow'),[Handle], 0, Result).

% obter o numero de linhas de uma listbox utilizando a
   API do windows
wlstcount(Listbox, Count):-
        sndmsg(Listbox, lb_getcount, 0, 0, Count).

% implementação do método string.contains de forma a
   testar se uma string BB contem a string AA
str_contains(AA, BB, Result):-
        convertStr(AA, A),
        convertStr(BB, B),

        string_chars(A, LA),
        string_chars(B, LB),
        length(LA, LLA),
        compare_to(LA, LB, 0, LLA, Result), !.

% caso seja string
convertStr(A, A):-
        string(A).
% se nao for string converte para string
convertStr(A, SA):-
        not(string(A)),
        atom_string(A, SA).
```

```
% permite comparar duas listas até Max caracteres,
   Result = 1 => 1ª lista contida na 2ª lista
compare_to([], _,_,_, 1).
compare_to([LA|RA], [LB|RB], Current, Max, Result):-
        Current < Max,
        LA == LB,
        NextCurrent is Current + 1,
        compare_to(RA, RB, NextCurrent, Max, Result).

compare_to([LA|RA], [LB|RB], Current, Max, Result):-
        Current >= Max.

compare_to(_, _, _, _, 0).
```

Listing A.11: Conteúdo exemplo de um ficheiro *.ctrl*

```
:-dynamic info/1.
:-dynamic via/3.
:-dynamic vizinho/3.
:-dynamic bloqueio/2.

info('CC1').
via('A','10000',0).
via('B','20000',1).
vizinho('CC2','A','SA1').
vizinho('CC3','B','SB1').
bloqueio('SA1','B').
```