Sistemas de Aprendizagem

Catarina Cruz, Mariana Marques, Miguel Solino, and Paulo Lima

University of Minho, Department of Informatics, 4710-057 Braga, Portugal
[a84011,a85171,a86435,a89983]@alunos.uminho.pt
Grupo 4
3 de Janeiro de 2021

1 Introdução

Neste relatório será apresentada a segunda fase do trabalho prático da unidade curricular de Agentes Inteligentes, integrada no perfil de especialização em Sistemas Inteligentes. Este projeto tem como principal foco a modelação e desenvolvimento de um sistema multi-agente (SMA) para a monitorização e disponibilização de métodos de locomoção para alugar, através da aplicação de sensores virtuais de captura de localização GPS. Neste caso, os métodos de locomoção serão bicicletas alugáveis e o sistema multi-agente será composto por diversos tipos de agentes, tais como: os que representam as estações, os agentes utilizadores que serão os responsáveis por simular indivíduos que alugam no sistema e os agentes interface que irão comunicar com os agentes estação e utilizadores para representar num mapa a posição de cada um.

Na realização desta fase, foi realizado um sistema multi-agentes tendo por base a conceção e arquitetura realizadas anteriormente.

1.1 Estrutura do documento

O presente documento inicializa-se com uma breve introdução. De seguida, será feita uma contextualização sobre o tema em questão bem como agentes e multi-agentes. Após a contextualização, serão detalhados os diversos agentes que integrarão o SMA a ser concebido. Na secção seguinte, é apresentada uma proposta de resolução inicial do problema e os diversos protocolos delineados. Por último, conclui-se apresentando a perspectiva de trabalho futuro

2 Contextualização

A procura por uma mobilidade mais sustentável é cada vez mais frequente, principalmente pelos grandes centros urbanos. São várias as cidades que cada vez mais adotam medidas para incentivar a utilização de modos suaves nas pequenas deslocações diárias, tentando mostrar que são cidades interessantes e atrativas. Uma das medidas é implementar sistemas de bicicletas partilhadas(SBP) na cidade, onde na última década este tipo de sistemas tem adquirido uma maior visibilidade devido ao avanço tecnológico que permitiu melhorar estes sistemas e torná-los mais atrativos.

Portugal é um país que apresenta uma percentagem baixa (aproximadamente 1%) no uso da bicicleta como meio de deslocação. Os municípios querem contrariar esta realidade e já existem alguns sistemas de bicicletas partilhadas implementados no território nacional. Um dos objetivos da estratégia Portugal 2020 é mais sistemas deste tipo em várias cidades e executar medidas complementares para incentivar o seu uso.

Os SBP são, na maioria das vezes, promovidos através dos seus benefícios, é a forma mais comum e segura de o fazer. É difícil questionar estes sistemas quando não existe um verdadeiro propósito ou uma meta que assegure o verdadeiro sucesso destes sistemas.

Um dos tipos de SBP, e aquele que é atualmente um dos mais adotados devido à facilidade de utilização, é o programa que possui bicicletas que podem ser levantadas e deixadas

em sistemas de docas (Fishman 2016)[1], abertas ao público e que podem servir como meio de transporte. Normalmente, as estações estão localizadas em vários pontos estratégicos da cidade e os utilizadores podem levantar a bicicleta numa estação e devolvê-la noutra totalmente diferente (Oliveira et al. 2016)[2]. Para encorajar a população a utilizar este tipo de programas, normalmente os primeiros 30 minutos são grátis, com um aumento ligeiro no preço à medida que a utilização excede o período de tempo gratuito (Fishman, Washington, and Haworth 2013) [1,3].

Os SBP existem há mais de 50 anos, mas a implementação deste tipo de sistemas tem sido afetada por uma grande adesão a nível mundial na última década. Isto deve-se ao facto de atualmente existirem estratégias melhores, tornando a sua implementação mais atrativa (Fishman et al. 2013; Wang and Zhou 2017) [3,4].

2.1 Agentes e Sistemas multi-agentes

O presente projeto consistem em desenvolver um sistema SBP utilizando para tal um DSS baseado em agentes.

Um agente pode ser visto como algo que corporiza um sistema computacional capaz de uma ação flexível e autónoma, desenvolvido num determinado meio ou sobre um dado universo de discurso. Entre as suas características encontram-se aspetos como a proatividade, dinamismo, autonomia, o facto de serem orientados a objetivos e terem perceção sobre o ambiente em que inserem. Um sistema multi-agente pode ser definido como uma coleção de agentes autónomos que comunicam entre si para coordenarem as suas atividades, de forma a que, coletivamente, consigam resolver um problema que sozinhos não conseguiriam. As soluções baseadas em agentes vêm, então, substituir a visão de controlo centralizado, rígido e monolítico por um paradigma onde das interações entre indivíduos surge a caraterística inteligente do sistema.

Por exemplo, os SMA têm vindo a afirmar-se como uma área do conhecimento em que se realizada investigação de qualidade, surgindo a todo o momento novos produtos, em diversos setores de atividade económica. Além disso, os SMA encontram-se alinhados com a tendência crescente em desenvolver ferramentas de forma modular, com controlo descentralizado, capazes de responder a distúrbios, reconfigurando-se sem necessidade de parar todo o sistema.

3 Agentes a desenvolver

Após análise e contextualização sobre o problema proposto, procedemos à apresentação dos agentes que integrarão o SMA que se pretende conceber. Os agentes que constituirão o sistema são os seguintes:

- Agente Estação (AE)
- Agente Utilizador (AU)
- Agente Interface (AI)

De seguida iremos descrever os agentes aqui enumerados, detalhando, para cada um, as características identificadas tendo em vista a criação de um sistema o mais completo possível.

3.1 Agente Estação

O agente estação tem como função escutar as mensagens dos Agentes Utilizador, controlar as entradas e saídas dos AUs dentro da sua área de proximidade(APE), monitorizar os alugueres e devoluções das bicicletas da sua estação e com esses dados determinar o seu estado e caraterizar o nível de preenchimento da estação, sendo os níveis: baixo, médio ou

alto. Com base nessa informação o agente terá ainda de decidir se pretende atrair bicicletas de utilizadores ou incentivar os mesmos a alugar, com base nas posições de todos os agentes utilizadores no mapa.

Posto isto, os dados necessários para representar este agente serão a sua posição, um raio que determina a área de proximidade, bem como o número máximo de bicicletas que pode armazenar e o número atual que permite definir o nível de aluguer no momento.

A posição poderá ser caraterizada por dois valores que representarão as suas coordenadas num mapa 2D.

3.2 Agentes Utilizador

Agente que, com base nos dados fornecidos pelo sensor GPS do seu smartphone, toma a decisão de qual estação escolher para devolução da bicicleta do utilizador. Além desses dados fornecidos também será influenciado na escolha dependendo das solicitações recebidas para entregar a bicicleta, sendo apresentado ao mesmo todas as estações com capacidade disponíveis para poder realizar a devolução nessa estação.

Com base no que foi dito, um agente utilizador será caraterizado pelas suas coordenadas no mapa, pela duração do seu trajeto, pela sua velocidade média e pela posição das estações onde aluga e devolve a bicicleta.

3.3 Agente Interface

O agente interface é o agente responsável por comunicar com os agentes estação e clientes para obter o estado atual do sistema (capacidade das estações e posição dos utilizadores), e consequentemente apresentar o resultado da simulação. Além disso, também terá como função apresentar um conjunto de estatísticas ao utilizador durante o recorrer da simulação. Estas estatísticas incluem o tempo mínimo, médio e máximo de todos os trajetos realizados pelos clientes e apresentam também o número de alugueres ativos por estação bem como os alugueres ativos ao longo do tempo.

O seu objetivo passa por captar métricas que nos permite avaliar o desempenho do sistema e a sua capacidade na gestão das estações.

4 Proposta de Resolução Inicial

Primeiramente é importante referir algumas considerações que foram tomadas para o desenvolvimento do projeto. Estas considerações procuram extrapolar o enunciado, e delinear certas características que gostaríamos de ver implementadas no projeto final.

4.1 Aspetos a considerar

De forma a tornar o SMA a desenvolver mais realista, os agentes utilizador terão autonomia própria, permitindo a estes escolherem a estação e o percurso que pretendem realizar. Ou seja, os pontos iniciais e finais de cada agente terão de ser aleatórios para garantir algum realismo e dinamismo no sistema. Bem como a velocidade e duração do percurso serão aleatórios dentro de um limite possível.

Para além disso, todas as decisões que os agentes tiverem de tomar, como por exemplo, se o utilizador pretende ou não aceitar o incentivo de uma estação terão em consideração o ambiente à sua volta para além de um fator aleatório. Por exemplo, caso um agente tente alugar uma bicicleta numa estação que não tenha nenhuma disponível, e caso haja outra estação num raio de 2 kilómetros, então este agente pode optar por alugar uma bicicleta nessa estação em vez de desistir automaticamente.

4.2 Arquitetura do Sistema

Seguidamente é apresentada a arquitetura do Sistema através de um diagrama de classes. No diagrama é possível identificar:

- Agentes intervenientes no sistema, como Agente Utilizador, Agente Estação, Agente Interface
- Estado dos agentes do sistema, que inclui por exemplo a sua posição
- Especificação do Aluguer realizado pelo Agente Utilizador
- Nível de ocupação da Estação, podendo esta ser baixo, média ou alta, consoante o número de bicicletas atuais disponíveis em proporção à capacidade que suporta
- A classe Mapa, representativa do ambiente no qual os agentes se encontram inseridos e atuam, que recebe contribuições dos vários agentes.

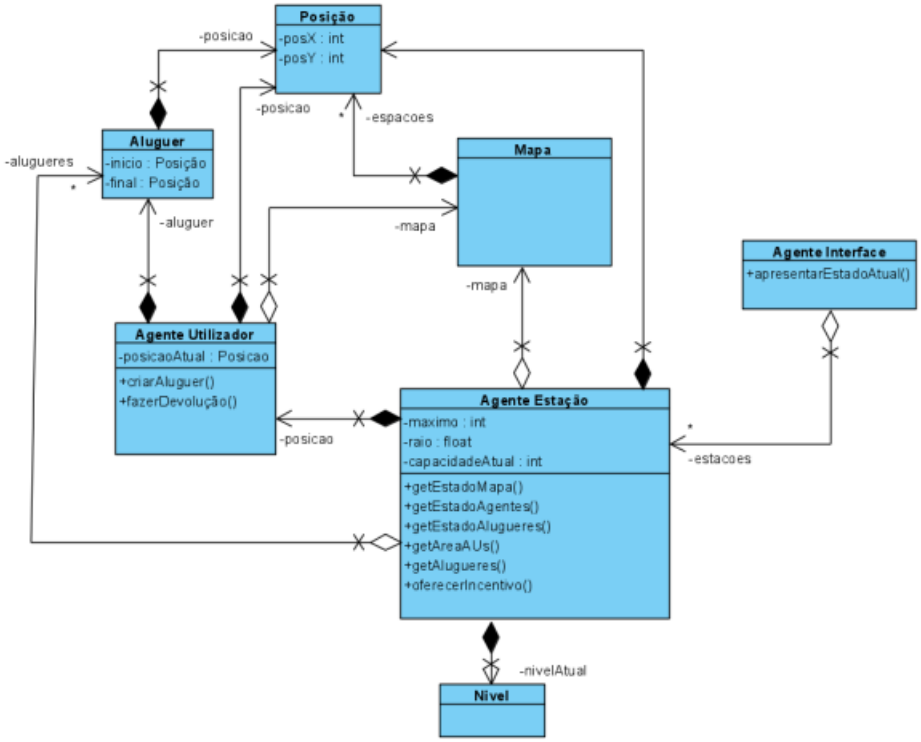

Figura 1. Arquitetura do Sistema

É provável que novas relações possam surgir, por exemplo, o Agente Interface poderá utilizar a classe Mapa, caso seja pertinente para facilitar a amostra dos dados, sendo um objeto dessa classe enviado pelo Agente Estação ao Agente Interface aquado da requisição da informação sobre a simulação. Para além disso, algumas das relações identificadas poderão, na prática, ser abstraídas pelos mecanismos de comunicação disponibilizados pelo JADE.

É de realçar que a informação sobre os agentes utilizador será feita de *forma cíclica* pelo Agente Estação, para esta informação estar sempre atualizada, e ele saber exatamente onde é que os utilizadores estão.

De forma análoga, o agente Utilizador apenas realizará as ações de aluguer e devolução de bicicletas uma vez, como tal apresentará um *OneShotBehaviour*.

4.3 Fluxo de Aluguer de Bicicletas

De forma a ser facilmente compreendido o modo como se processa o aluguer e devolução de bicicletas é apresentado o diagrama de atividades. Neste diagrama é possível compreender a influência dos agentes no sistema bem como a interação entre os vários agentes, tendo em vista todo o processo envolvente do aluguer de bicicletas.

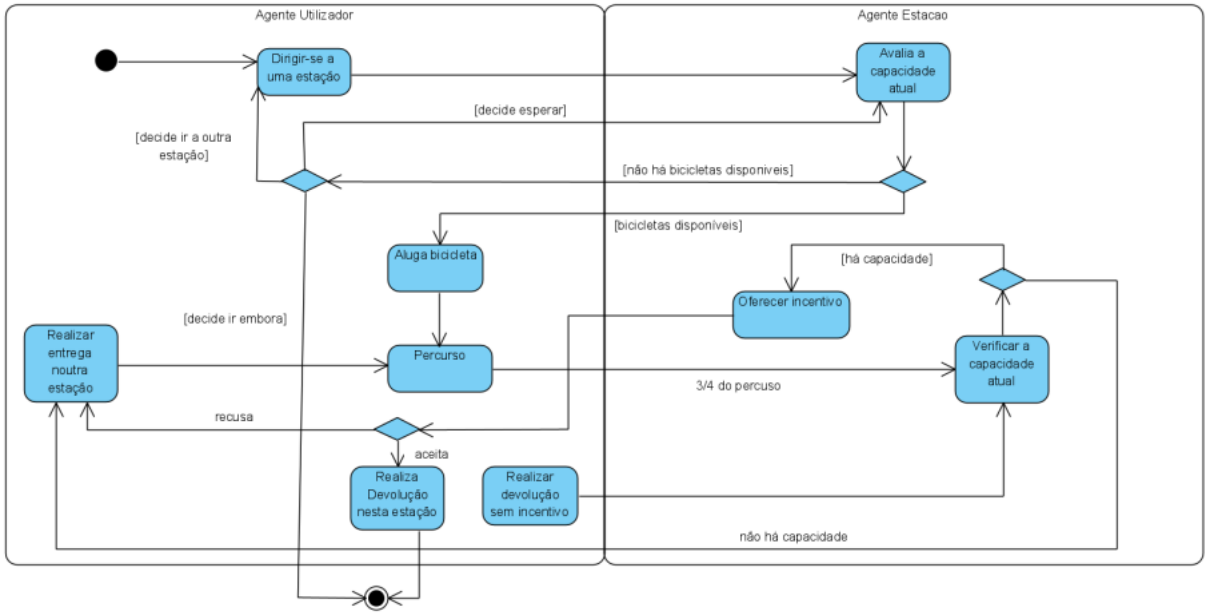

Figura 2. Fluxo de alugueres/devoluções

Através da observação do diagrama podemos verificar que o processo tem início quando um agente utilizador se desloca para uma estação com o intuito de alugar uma bicicleta. Nesse momento, a estação avalia se tem bicicletas disponíveis, caso possua bicicletas o aluguer fica feito. Se não existirem bicicletas disponíveis para alugar, o agente utilizador pode optar por se deslocar a outra estação para efetuar lá o aluguer, pode esperar naquela estação que alguém devolva uma bicicleta ou pode desistir e não efetuar aluguer, terminando aí o fluxo.

Após alugada uma bicicleta, o utilizador realiza o seu percurso, e quando estiver a 3/4 deste, as estações cuja área de proximidade incluam a posição do utilizador e cujo nível de ocupação seja baixo ou médio, enviam um incentivo ao utilizador. Caso o utilizador aceite o incentivo, a bicicleta fica entregue a estação, e o aluguer fica completo. Caso o utilizador pretenda continuar o percurso e entregar a bicicleta numa outra estação, ainda é necessário confirmar se a estação possui capacidade para aceitar a bicicleta e caso não possua o utilizador ainda se terá de deslocar para outra estação para realização a devolução da bicicleta.

4.4 Protocolos de Comunicação

De forma a tornar o sistema dinâmico, os agentes apresentados anteriormente têm necessariamente de comunicar entre si. Para que todas estas interações ocorram de forma correta, os agentes cumprem protocolos de comunicação estabelecidos na criação do sistema.

De seguida, apresenta-se os diversos protocolos de comunicação entre os agentes, introduzindo-se o tema com a definição de um diagrama de colaboração simplificado que serve para identificar as *perfomatives* utilizadas entre cada par de agentes e em que sentido estas são usadas.

O panorama geral de comunicação entre os agentes encontra-se representado na seguinte figura:

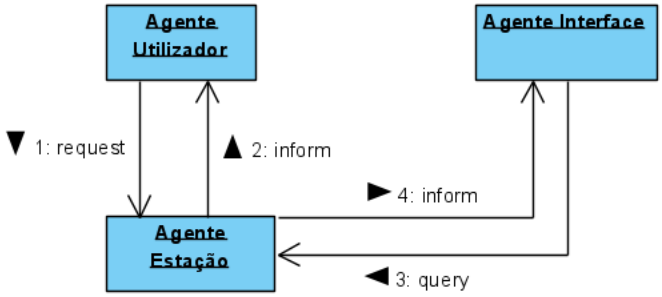

Figura 3. Panorama geral de comunicação entre agentes

Ao observar a imagem conseguimos concluir que entre o Agente Utilizador e o Agente Estação existem mensagens do tipo `request`, que correspondem à delegação de tarefas como alugueres ou devoluções. Da mesma forma, as mensagens do tipo `inform` entre o Agente Estação e o Agente Utilizador servem para informar sobre a conclusão destas mesmas tarefas. Por último, consegue-se inferir o pedido de dados por parte do Agente Interface ao Agente Estação via mensagem do tipo `query`, sendo a resposta dada com uma mensagem do tipo `inform`.

Protocolo entre Agente Utilizador e o Agente Estação

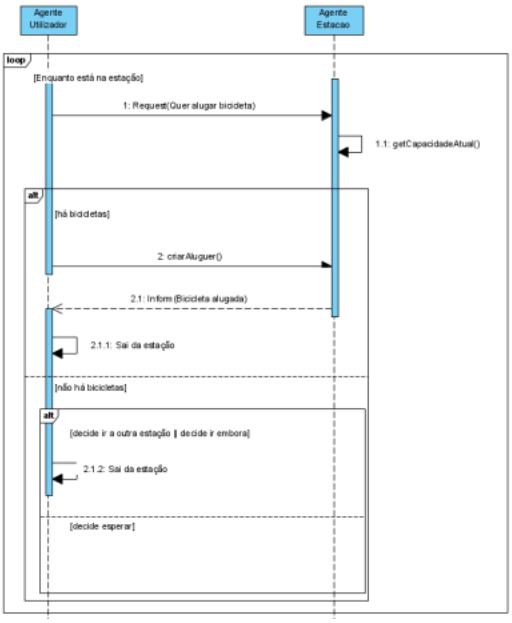

Figura 4. Diagrama de Sequência que descreve a comunicação entre o agente utilizador e o agente estação

Na estação o utilizador tenta alugar uma bicicleta, mas para isso é necessário avaliar a capacidade das bicicletas nessa mesma estação e assim concluir se há ou não bicicletas disponíveis a alugamento. Caso a resposta seja positiva a bicicleta é alugada pelo cliente (e consequentemente o cliente é avisado que o aluguer foi bem sucedido) e de seguida este abandona a estação. No entanto, se a reposta for negativa e portanto não houverem bicicletas que possam ser alugadas, o utilizador pode escolher ficar na mesma estação à espera que bicicletas fiquem disponíveis, ou pode sair da estação e deslocar-se a outra para tentar proceder ao mesmo serviço.

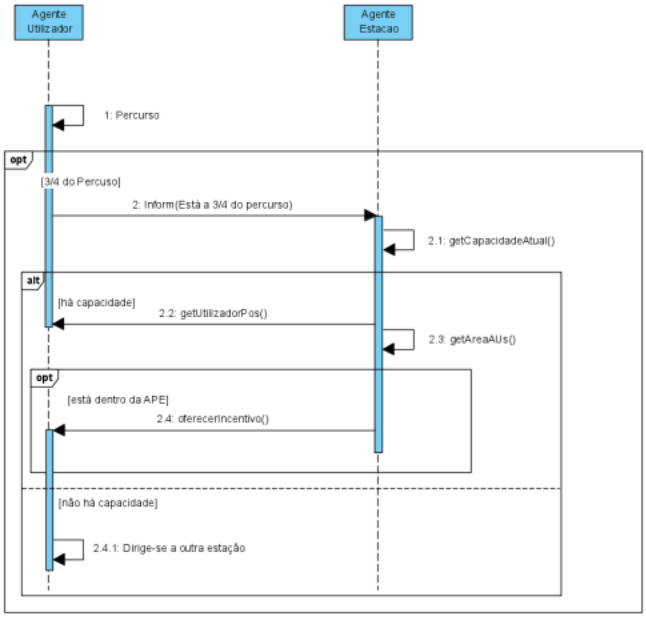

Figura 5. Diagrama de Sequência que descreve a comunicação entre o agente utilizador e o agente estação

Já na posse da bicicleta, quando o utilizador estiver a 3/4 do seu percurso a estação é notificada e avalia a sua capacidade para averiguar se a bicicleta pode ser ou não recebida nessa estação. Se a capacidade for favorável, a estação determina a posição do utilizador e avalia se este se encontra dentro da área de proximidade dessa estação e se estiver este procede à oferta de incentivo . Por outro lado, se a capacidade não for favorável o utilizador terá de abandonar a estação e deslocar-se a outra.

Protocolo entre Agente Interface e o Agente Estação

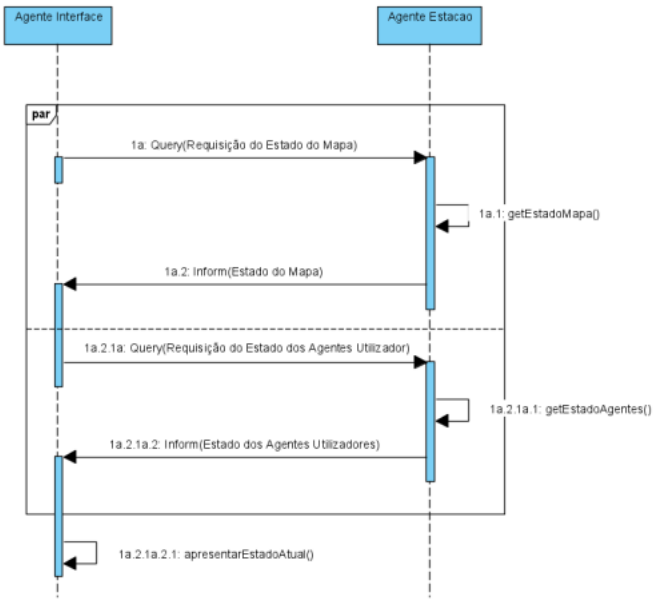

Figura 6. Diagrama de Sequência que descreve a comunicação entre o agente interface e o agente estação

Este protocolo de comunicação realiza a requisição de informação por parte do agente interface em relação ao agente estação. A informação transferida, em paralelo, é relativa ao estado do mapa e aos agentes utilizadores.

5 Alterações aos Diagramas UML

Durante a segunda fase do trabalho prático verificamos que alguns dos diagramas UML elaborados anteriormente tinha alguns pormenores que não se adequavam à solução implementada. Assim sendo, nesta secção iremos apresentar os que tiveram de sofrer pequenas alterações, bem como o motivo para tal, assim como se pode observar de seguida.

5.1 Diagrama de Classes

Neste diagrama foram adicionados, maioritariamente, os behaviours e os seus tipos, visto que antes não eram referenciados. Desta forma, cada agente possui um *CyclicBehaviour* para estar constantemente a receber pedidos e poder processá-los, sendo estes behavious *Reply*, *Reciever Request* e *Reciever* para os Agentes Client, Station e Interface, respetivamente.

Tanto o Agente Client como o Station, possui um *OneShotBehaviour* chamado EnviaParaInterface que envia uma mensagem para a Interface com os dados de cada Agente, no caso do Cliente a sua posição e o estado atual, isto é, se está à espera de alugar uma bicicleta, se já se encontra no percurso, se já está a 3/4 do mesmo, se terminou o percurso ou se já entregou a bicicleta.

Para além disso, o Cliente possui outro *OneShotBehaviour*, o *RequestBike* para alugar bicicletas, e possui ainda dois *WakerBehaviour Avisa_meio_percurso* e *Final_percurso*,

para avisarem, ao fim de um determinado tempo, a estação do seu estado atual. Por último, como seria de esperar, para poder atualizar continuamente a sua posição,*Atualiza_possui* é um *TickerBehaviour* que é ativado a cada segundo.

O Agente Interface, para poder exibir graficamente o estado do sistema e estatísticas do seu funcionamento possui dois *TickerBehaviours*, *PrintEstados* e *TimeCount*.

Por último, foram adicionadas duas Classes, a *PositionStatus* possui uma posição e uma String para indicar, principalmente, o status atual bem como a posição atual do Cliente. A classe *InformStationStatus* possui todas as váriaveis que definem uma estação, isto é, a sua posição, a sua capacidade (bikes), o número de bicicletas disponíveis no momento (available) e a sua área de proximidade (raio). A classe Mapa que inicialmente inicialmente se mencionou, não foi utilizada e como tal, foi retirada do diagrama.

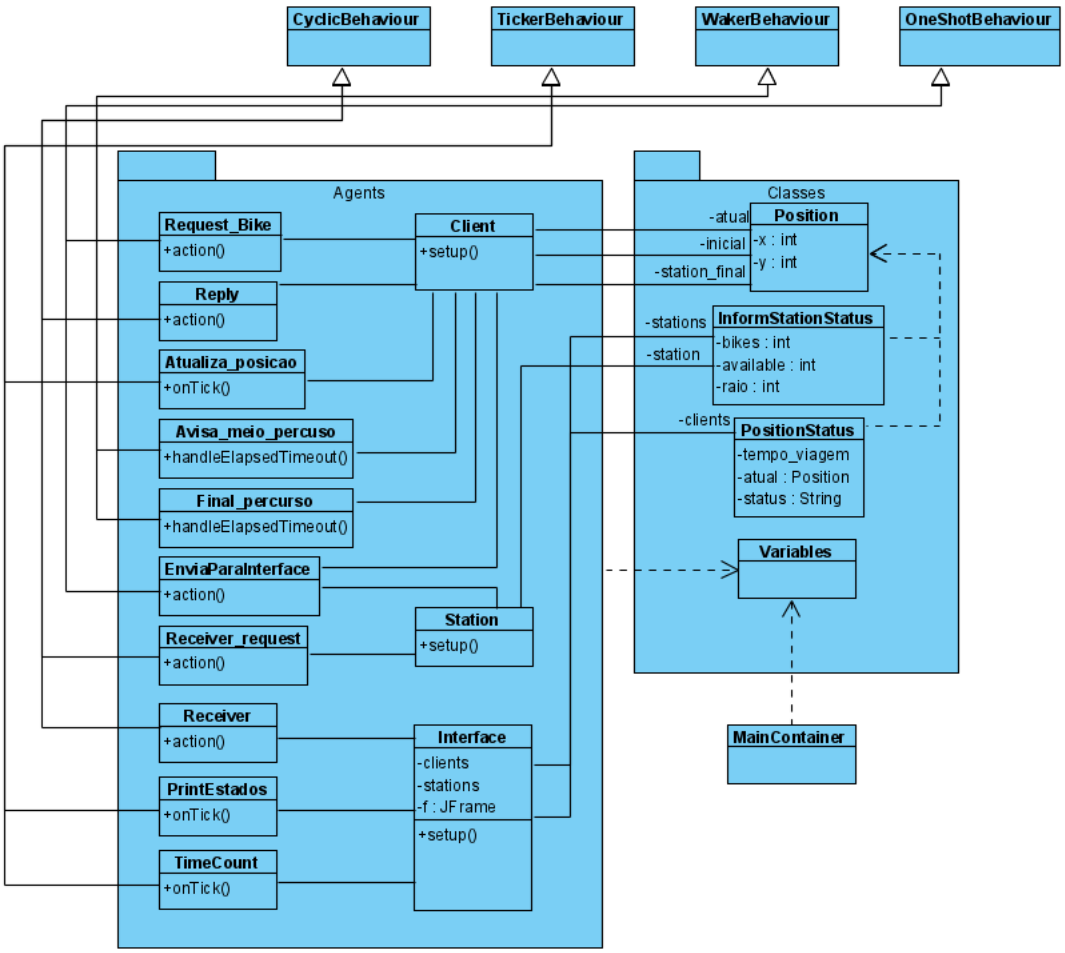

Figura 7. Diagrama de Classes

5.2 Protocolos de Comunicação

Numa primeira abordagem do trabalho, ponderamos o Agente Interface estar constantemente a questionar o Agente Estação sobre as alterações ocorridas no sistema e como resposta, este enviar as mudanças ocorridas. No entanto, apercebemo-nos que, para além

do Agente Estação não possuir a posição dos utilizadores, era inconveniente estar constantemente a trocar mensagens entre a Interface e a Estação. Como tal, sempre que ocorre alguma alteração na estação, esta envia uma mensagem com a informação atualizada à Interface e para além disso, o Agente Utilizador sempre que atualiza a sua posição envia também uma nova mensagem ao Agente Interface. Em suma, sempre que ocorre alguma alteração nalgum dos Agentes, estes comunicam ao Agente Interface e este não precisa de pedir informação a nenhum deles.

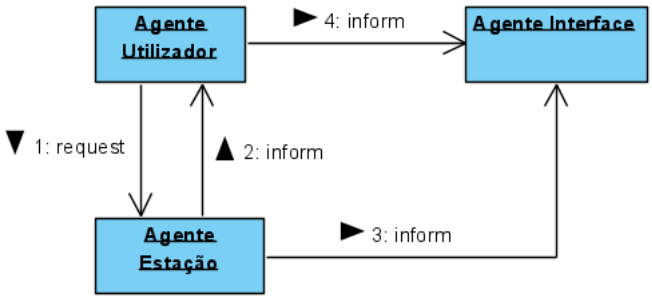

Figura 8. Protocolos de Comunicação

5.3 Diagramas de Sequência

Foi necessário alterar o seguinte diagrama, uma vez que na primeira fase tínhamos pensado em colocar o agente interface a pedir constantemente a informação do mapa ao agente estação. No entanto, verificamos que apenas fazia sentido sempre que houvesse alguma alteração por parte da estação, esta enviar a informação nova ao agente interface.

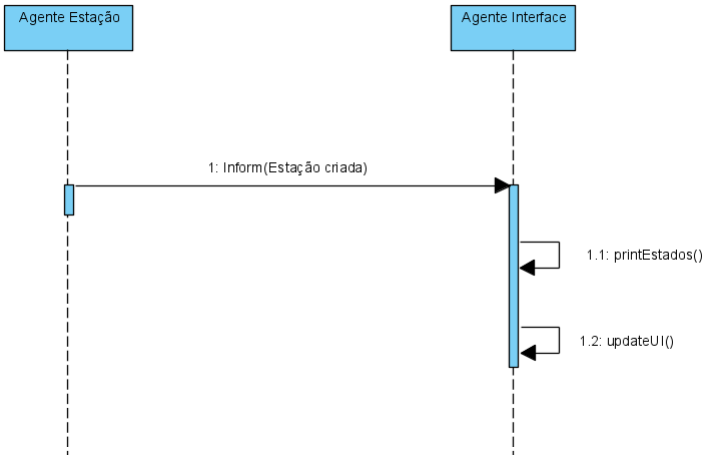

Figura 9. Diagrama de Sequência de Estação - Interface

Tal como já referido, inicialmente consideramos que apenas o Agente Estação envia a informação para o Agente Interface, contudo, este não contém a informação dos clientes.

Como tal, estes, sempre que atualizam a sua posição, bem como o seu status, enviam a nova informação ao Agente Interface para este poder exibir graficamente as alterações ocorridas no sistema.

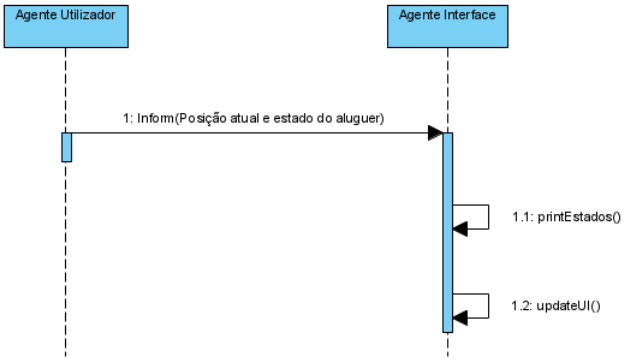

Figura 10. Diagrama de Sequência de Utilizador - Interface

6 Implementação

6.1 Geração do Mapa e Estações

Inicialmente é gerado um mapa que contém todos as estações existentes. Para isso, primeiro tivemos de inicializar a posição destas, de forma a que, no momento da geração do mapa estas posições já tivessem sido obtidas. Para tal, optámos por atribuir posições aleatórias às estações.

```
// Registo do agente
DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(getAID());
ServiceDescription sd = new ServiceDescription();
sd.setName(getLocalName());
sd.setType("station");
dfd.addServices(sd);

try {
  DFService.register(this, dfd);
} catch (FIPAException fe) {
  fe.printStackTrace();
}

// Geracao da posicao
Random rand = new Random();
Position position = new Position(rand.nextInt(100),
    rand.nextInt(100));
```

É importante ainda referir que, no instante em que a estação é criada, a interface é notificada (mais tarde será mencionado na parte da interface).

```
// Inicio dos behaviours
```

```
addBehaviour(new EnviaParaInterfaceStation(station));
addBehaviour(new Receiver_request());
```

6.2 Geração dos utilizadores

Os utilizadores são gerados no início do processo de aluguer de bicicleta, ou seja, apenas quando o utilizador tenta alugar uma bicicleta é que começa a fazer parte do sistema. É ainda de realçar que ao iniciar este processo é criado o canal de comunicação entre o agente utilizador e agente estação para o início do pedidos.

```
// Inicio dos behaviours para inicio de tentativa de
   aluguer de uma bicicleta
this.addBehaviour(new Request_bike());
this.addBehaviour(new Reply());

// Criacao do canal de comunicacao com as estacoes
DFAgentDescription template = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType("station");
template.addServices(sd);

// Escolha da estacao para aluguer
int inicial = rand.nextInt(result.length);
while (station_inicial == inicial)
    inicial = rand.nextInt(result.length);
station_inicial = inicial;
ACLMessage mensagem = new ACLMessage(ACLMessage.REQUEST);
mensagem.addReceiver(result[station_inicial].getName());
mensagem.setContent("Request_Bike");
myAgent.send(mensagem);
```

6.3 Alugar

Para proceder ao aluguer de uma bicicleta, é necessário inicialmente começar por enviar uma mensagem a uma estação aleatória, essa estação verifica se tem bicicletas disponíveis e fornece essa informação ao utilizador.

```
// Escolha da estacao para aluguer
int inicial = rand.nextInt(result.length);
while (station_inicial == inicial)
    inicial = rand.nextInt(result.length);
station_inicial = inicial;
ACLMessage mensagem = new ACLMessage(ACLMessage.REQUEST);
mensagem.addReceiver(result[station_inicial].getName());
mensagem.setContent("Request_Bike");
myAgent.send(mensagem);

// Estacao recebeu pedido de aluguer
if (!station.isEmpty()) {
    station.requestBike();
    addBehaviour(new EnviaParaInterfaceStation(station));
//    resp.setContent("Bicicleta alugada");
    try {
        resp.setContentObject(station.getPosition());
```

```
    } catch (IOException e) {
        e.printStackTrace();
    }
    resp.setPerformative(ACLMessage.CONFIRM);
    myAgent.send(resp);
} else {
    resp.setContent("Nao ha bicicletas disponiveis");
    resp.setPerformative(ACLMessage.REFUSE);
    myAgent.send(resp);
}
```

Posteriormente, tendo o utilizador recebido a confirmação que pode alugar uma bicicleta este inicia três novos comportamentos:

- **Atualiza_posicao :**
  É um *TickerBehaviour* que a cada tick, com duração fornecida na criação do mesmo, atualiza a sua posição atual.

- **Avisa_meio_percurso:**
  É um *WakerBehaviour* que é responsável por avisar o sistema que o utilizador se encontra a 3/4 no percurso.

- **EnviaParaInterface:**
  É um *OneShotBehaviour* e é responsável por, a cada nova mudança de estado, envia essa mesma mudança para a interface.

```
// Utilizador realiza o aluguer
if (msg.getPerformative() == ACLMessage.CONFIRM) {
                    // se existem bicicletas disponiveis
    try {
        atual = (Position) msg.getContentObject();
    } catch (UnreadableException e) {
        e.printStackTrace();
    }
    Random random = new Random();
    tempo_viagem = random.nextInt(90) + 1;
    speed = random.nextInt(5 - 3) + 3;
    percurso = true;
    k = -2;
    v = -1;
    int x = tempo_viagem * 3000 / 4;
    status = "alugada";
    System.out.println(tempo_viagem);
    addBehaviour(new Atualiza_posicao(this.myAgent, 1000));
    addBehaviour(new Avisa_meio_percurso(this.myAgent, x));
    addBehaviour(new EnviaParaInterface());
}
```

Caso a estação onde o cliente tenta realizar o aluguer não possuir bicicletas disponíveis, o utilizador será avisado pela estação que o aluguer não poderá ocorrer. Ao receber esta informação, o utilizador deverá tomar uma de entre três decisões possíveis :

- esperar algum tempo e voltar a perguntar à mesma estação se já pode realizar o aluguer;
- mudar para uma outra estação aleatória para efetuar o aluguer;
- sair do sistema, e não efetuar qualquer aluguer.

```
// Estacao avisa o utilizador
resp.setContent("Nao ha bicibletas disponiveis");
resp.setPerformative(ACLMessage.REFUSE);
myAgent.send(resp);

// Utilizador decide esperar, ir a outra estacao ou nao
   realizar aluguer
if (prob <= 40) {
//espera
try {
    Thread.sleep(5000);
} catch (InterruptedException e) {
    e.printStackTrace();
}
ACLMessage resp = new ACLMessage(ACLMessage.REQUEST);
resp.addReceiver(msg.getSender());
resp.setContent("Request_Bike");
myAgent.send(resp);
}
//vai a outra estacao
if (prob > 40 && prob <= 80) {
    addBehaviour(new Request_bike());
}
//desiste
if (prob > 80) {
    myAgent.doDelete();
}
```

6.4 Devolução

Quando os utilizadores se encontram a 3/4 do percurso, avisam todas as estações da sua posição atual. Caso estes se encontrem dentro da área de proximidade de uma estação e caso ainda não esteja completamente cheia, então esta envia para o utilizador uma proposta de devolução para aquela estação.

```
// Estacoes enviam para utilizadores que estejam a 3/4 do
   percurso dentro da sua area de proximidade
if (client_obj.getStatus().equals("34 percurso")) {
    if (station.area_proximidade(client_obj.getAtual()) &&
       !station.isFull()) {
    PositionStatus status_pos = new
       PositionStatus(station.getPosition(), "34
       percurso");
    resp.setContentObject(status_pos);
    resp.setPerformative(ACLMessage.PROPOSE);
    myAgent.send(resp);
}
```

Ao receber a proposta, o utilizador decide se quer ou não aceitar o pedido. Sendo que 75% das vezes o pedido é aceite.

```
// Utilizador decide ou nao aceitar a proposta
if (accept < 75 && percurso) {
    resp.setContent("ACEITO");
    resp.setPerformative(ACLMessage.ACCEPT_PROPOSAL);
```

```
    myAgent.send(resp);
    percurso = false;
    station_final = station_obj.getAtual();
    inicial = new Position(atual.getX(), atual.getY());
    v = atual.distancia(station_final);
    k = 0;
}
```

No caso do utilizador não aceitar a proposta de nenhuma das estações, então este continua o seu percurso normalmente. Assim que seja ao fim, notifica todas as estações da sua posição atual e todas aquelas que tenham espaço disponível avisam o utilizador que pode efetuar lá a devolução da bicicleta.

O utilizador, assim que receber uma oferta aceita de imediato a mesma e inicializa um percurso reto até à posição da estação que enviou a oferta.

É ainda de realçar que, no caso do utilizador receber várias propostas, a escolhida será a que enviou o primeiro pedido.

```
// Utilizador aceita a entrega de uma estacao aleatoria
if (station_obj.getStatus().equals("fim percurso")) {
    if (percurso) {
    resp.setContent("ACEITO");
    resp.setPerformative(ACLMessage.ACCEPT_PROPOSAL);
    myAgent.send(resp);
    percurso = false;
    station_final = station_obj.getAtual();
    inicial = new Position(atual.getX(), atual.getY());
    v = atual.distancia(station_final);
    k = 0;
    }
}
```

6.5 Interface

O agente interface recebe várias informações por parte de todos os outros agentes presentes no sistema, de forma a poder apresentar um mapa com todos os agentes utilizador e estação.

Como tal, apresenta dois HashMaps, uma para guardar a informação relativa às estações e outro para guardar as posições e status dos clientes ativos no sistema.

Por isso, através de um *CyclicBehaviour* verifica constantemente se recebe mensagens do tipo *Performative Inform*.

Para além disso, confirma ainda o id da conversa. Caso este seja "station_info", então o conteúdo da mensagem será relativo à estação. Com isto, o Agente Interface verifica se já possui informação sobre o agente em questão. Em caso afirmativo, substitui a informação pela recebida, em caso negativo, adiciona a nova informação ao HashMap relativo às estações.

De forma análoga, caso o id seja "client_info", a informação é relativa aos clientes, e é realizado o mesmo processo, armazenado e atualizando o informação no HashMap dos clientes. Caso o status do cliente seja "cheguei", então o utilizador completou o seu percurso e entregou a bicicleta numa estação, e por isso, é removido do HashMap.

Com toda a informação processada e guardada tivemos a possibilidade implementar um mapa 2D bem como as várias estatísticas que são apresentados na secção seguinte deste documento.

```
// Informacoes das estacoes
```

```
if (msg.getPerformative() == ACLMessage.INFORM &&
    msg.getConversationId().equals("station_info")){
   try {
      InformStationStatus station_info =
         (InformStationStatus) msg.getContentObject();
      if(stations.containsKey(sender)){
          stations.replace(sender, station_info);
      }
      else{
          stations.put(sender, station_info);
      }
      work.updateUI();
    } catch (UnreadableException e) {
      e.printStackTrace();
}

// Informacoes dos utilizadores
if (msg.getPerformative() == ACLMessage.INFORM &&
    msg.getConversationId().equals("client_info")) {
  try {
    PositionStatus client_info = (PositionStatus)
        msg.getContentObject();
  if(clients.containsKey(sender)){
    if (client_info.getStatus().equals("cheguei")) {
      clients.remove(sender);
    }else
      clients.replace(sender, client_info);
    } else {
      TimeUpdate(client_info.getTempoViagem());
      Update();
      clients.put(sender, client_info);
    }
      work.updateUI();
    } catch (UnreadableException e) {
      e.printStackTrace();
    }
}
```

6.6 Demonstração

- **Mapa 2D**

Com o objetivo de exibir os resultados obtidos na simulação, decidimos desenvolver uma interface gráfica 2D em Swing, que pretende ilustrar a participação dos vários agentes na simulação.

Desta forma, o mapa desenvolvido tem uma dimensão 100 por 100, cujos agentes estação estão representados a azul e os agentes utilizador estão representados a amarelo. É ainda relevante notar que ao longo da simulação os agentes estação permanecem sempre na mesma posição enquanto os agentes utilizadores se vão movimentando de forma aleatória até saírem do sistema.

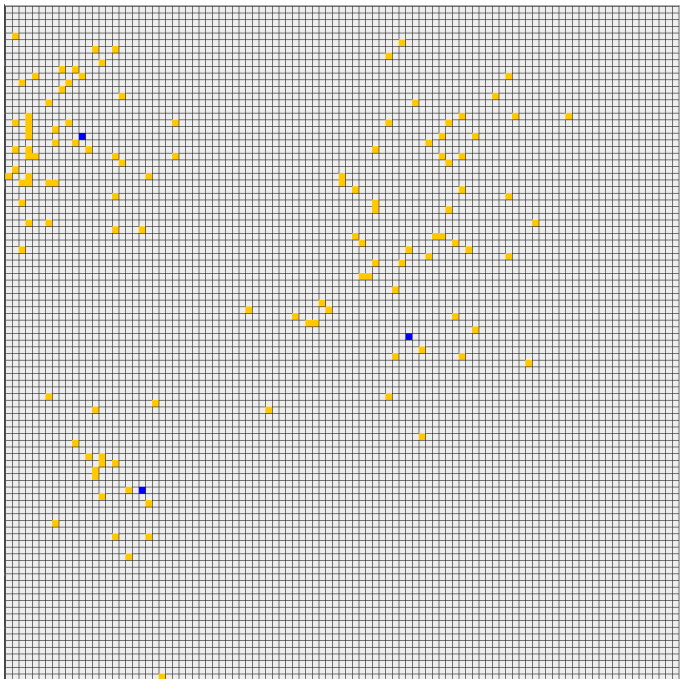

Figura 11. Mapa 2D

6.7 Estatísticas

De forma a apresentar a análise de performance dos agentes utilizador e estação, optámos por fazê-lo através de gráficos (recorrendo à API JFreeChart) e tabelas.

- **Tabelas**

ID	Position	Status
Client91	alugada	(87,39)
Client90	fim percurso	(32,27)

ID	Position	Capacity	Available	APE
Station2	(22,29)	21	21	14m raio
Station3	(77,62)	19	19	7m raio
Station0	(11,85)	15	15	6m raio
Station1	(32,27)	19	18	12m raio
Station4	(96,28)	18	18	7m raio

Figura 12. Resultados estatísticos

- **Gráficos**

Optámos por desenvolver três gráficos, um dos quais de barras representante do tempo médio, que para além de apresentar o tempo médio do percurso de todos os cliente que já passaram pelo sistema, apresenta também o tempo mínimo e máximo que um percurso já teve. Um gráfico circular que representa os alugueres ativos por estação e por último um gráfico de linhas que representa o número de clientes ativos ao longo do tempo.

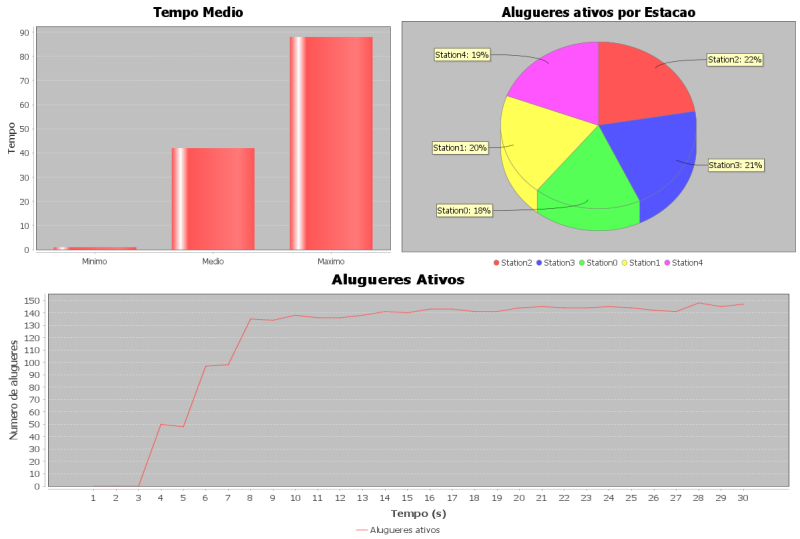

Figura 13. Resultados estatísticos

7 Conclusão e Trabalho Futuro

Concluído o projeto, apercebermo-nos que foi adquirido conhecimento sobre os conceitos de Agentes, tendo estes sido aplicados quer na modelação quer posteriormente na implementação deste problema real. Para inicializar este projeto começamos por investigar quais as solução existentes e em desenvolvimento no mercado, de modo a termos uma noção de como poderíamos desenvolver este projeto.

Assim, começamos a idealizar como iríamos estrutura-lo, através da modelação UML, que tornou possível demonstrar a estratégia por nós definida. Apesar desta ter sofreu pequenas alterações de uma fase para a outra, uma vez que inicialmente procuramos tentar simplificá-la ao máximo e ao mesmo tempo obter os melhores resultados possíveis.

Como tal, as etapas mais trabalhosas foram a coordenação dos agentes, uma vez que pretendíamos minimizar a troca de mensagens entre eles. Por exemplo, para avaliar se um agente está ou não dentro da área de proximidade de uma estação é necessário a estação saber qual a localização atual do cliente, no entanto, como este está em constante movimento aleatório, precisaria de estar constantemente a receber mensagens de todos os clientes e este estar constantemente a enviar mensagens para todas as estações, o que sobrecarregaria o sistema desnecessariamente.

Outro problema associado foi a tomada de decisões do cliente, pois para tentarmos colocar algum realismo no projeto, queríamos que os clientes não tomassem sempre a mesma decisão, como tal foi necessário pensar em todas as alternativas existentes para cada caso.

Em última instância, consideramos que este foi um projeto bem conseguido em parte tendo em conta as estatísticas e resultados obtidos, contudo, gostaríamos de ter melhorado certos pontos, como um utilizador poder escolher a estação mais perto, mas para tal era necessário ou trocar ainda mais mensagens entre o utilizador e as estações ou cada utilizador guardar a posição de todas as estações disponíveis, no entanto consideramos que não deveria ser trabalho do utilizador verificar qual a estação mais perto de si, mas sim ser a estação a dar mais incentivo ao utilizador.

Referências

1. FISHMAN, ELLIOT. 2016., *"Bikeshare: A Review of Recent Literature." Transport Reviews 36(1):92–113.*,
2. OLIVEIRA, GUILHERME N., JOSE L. SOTOMAYOR, RAFAEL P. TORCHELSEN, CLÁUDIO T. SILVA, AND JOÃO L. D. COMBA. 2016., *"Visual Analysis of Bike-Sharing Systems." Computers and Graphics (Pergamon) 60:119–29.*,
   http://dx.doi.org/10.1016/j.cag.2016.08.005
   Last accessed 27 Nov 2020
3. FISHMAN, ELLIOT, SIMON WASHINGTON, AND NARELLE HAWORTH. 2013. , *"Bike Share: A Synthesis of the Literature." Transport Reviews 33(2):148–65.*,
4. WANG, MINGSHU AND XIAOLU ZHOU. 2017, *"Bike-Sharing Systems and Congestion: Evidence from US Cities." Journal of Transport Geography 65(May):147–54*,
5. GIOVANNI CAIRE, *"Jade Tutorial : Jade Programming for beginners" Transport Reviews 36(1):92–113.*,
6. LUÍS FILIPE DO NASCIMENTO LIBERATO, *O potencial dos sistemas de bicicletas partilhadas: uma contribuição para a construção do panorama português*,
   FEUP, U.PORTO, Janeiro de 2018