Sistemas de Partilha de Bicicletas - SPB

André Figueiredo, Luís Ferreira, Pedro Machado, and Rafael Lourenço

Universidade do Minho, Departamento de Informática, 4710-057 Braga, Portugal

Agentes Inteligentes, Trabalho Prático Nº2 - Fase 1

17 de Novembro de 2020 e-mail: {a84807,a86265,a83719,a86266}@alunos.uminho.pt

Resumo Neste relatório é apresentada uma primeira versão da arquitetura distribuída de um sistema multi-agente para um Sistema de Partilha de Bicicletas.
A criação de sistemas de transportes públicos ou partilhados é algo que tem ganho cada vez mais importância. Esta prática permite o descongestionamento e redução de trânsito (principalmente em áreas de grande densidade populacional) e a redução de emissões e de poluição. Para além disso, estudos afirmam que a utilização de bicicletas como meio de transporte para os locais de trabalho permite, por exemplo, reduzir os níveis de stress e melhorar a produtividade. [1]
Além disso, trata-se de um meio de transporte relativamente barato, que permite a prática de exercício físico e que tem iniciativas a serem criadas pelos governos, para ser, cada vez mais, o meio de transporte preferencial.

Keywords: Artificial Intelligence · Intelligent Agents · Multi-Agent Systems · Public Bike Share · Bicycle Sharing Systems · Balancing Bicycle Sharing Systems

1 Introdução

No âmbito desta Unidade curricular foi-nos proposta a modelação e criação de um Sistema de Partilha de Bicicletas (`SPB`), que permitisse o seu balanceamento dinâmico.
Um dos maiores problemas de um Sistema de Partilha de Bicicletas, é o desequilíbrio de bicicletas em certas estações ao fim de algum tempo, pois, sem intervenção ”inteligente”, algumas estações terão falta de bicicletas, enquanto que outras terão excesso delas, o que impede as devoluções de acontecer. Portanto, para existir um sistema que agrade aos utilizadores sem que se verifiquem quebras de receitas, devem ser criados mecanismos que permitam o balanceamento do sistema.
Para isso, é aqui proposta uma modelação de um sistema multi-agente para partilha de bicicletas entre utilizadores, onde é permitido realizar viagens entre as várias estações espalhadas por uma cidade. Nesta aplicação existirão três tipos de agentes - um agente que representará cada um dos utilizadores do sistema, outro que representará cada uma das estações (de forma a monitorizar o seu estado) e um agente interface responsável pela interação entres os dois agentes apresentados anteriormente. Os dois primeiros podem ser visto como uma espécie de sensores, por exemplo, sensores de `GPS` e de ocupação, enquanto que o último se trata de uma camada de representação/comunicação, de forma a facilitar a interação entre utilizadores e estações.
Desta forma, começámos por fazer uma breve introdução e descrição do problema, seguido da apresentação da arquitetura geral do sistema, descrição dos agentes, comportamento destes e a comunicação/negociação entre os vários agentes.

Quanto à modelação deste projeto, esta é baseada em metodologias Agent UML, para obter uma descrição detalhada dos protocolos de comunicação, processamento interno e interações entre agentes.

Em relação a Portugal, já foram implementados, pelo menos, seis Sistemas de Partilha de Bicicletas:

- beÁgueda, cidade de Águeda em 2011, conta com 4 estações e 20 bicicletas;
- b→AND, cidade de Anadia em 2014, conta com 11 bicicletas;
- Buga, cidade de Aveiro em 2000, conta com 1 estação e 300 bicicletas;
- biCas, cidade de Cascais em 2016, conta com 80 estações e 700 bicicletas;
- GIRA, cidade de Lisboa em 2017, conta com 74 estações e 700 bicicletas;
- biConde, cidade de Vila do Conde em 2014, conta com 10 estações e 60 bicicletas;

1.1 Descrição do problema

Analisando o enunciado, o principal objetivo deste sistema passa por gerir o SPB. Para isso, é necessário levantar certos requisitos, como, por exemplo, as entidades e funcionalidades que devem ser suportadas.
Quanto às entidades (”físicas”), existem dois tipos - estações e utilizadores.
Para permitir a monitorização dos sensores do sistema (por exemplo, GPS, entre outros), é necessário criar uma arquitetura que permita a gestão dos vários sensores e a comunicação entre as várias entidades (agentes).

Sendo assim, esta arquitetura pode ser divida em três subsistemas principais:

- **Interface**, responsável pela gestão do sistema, desde do registo de estações até à apresentação de dados, como estações pertencentes ao mesmo, entre outros;
- **Utilizador e sensores**, onde existem quatro funções principais, sendo estas:
  - *Pedido de Aluguer*, isto é, ter um mecanismo para permitir que seja estabelecida comunicação com o sistema e que possa iniciar o processo de aluguer;
  - *Negociação*, isto é, receber e deliberar sobre as melhores propostas;
  - *Atualização da posição do utilizador*;
  - E, consequentemente, *notificação da sua posição atual* ao sistema.
- **Estação**, onde existem três funções principais, sendo estas:
  - *Registo da estação*, para permitir que as estações sejam adicionadas durante a execução do programa;
  - *Negociação*, isto é, decidir que descontos oferecer aos utilizadores;
  - *Receção das posições dos utilizadores*, de forma a saber que utilizadores se encontram na sua APE e que poderão receber os seus descontos.

Este sistema será, posteriormente, implementado recorrendo à linguagem Java em conjunto com a biblioteca Jade, podendo sofrer algumas alterações.

2 Agentes

2.1 Classes

O nosso sistema é constituído, essencialmente, por três agentes:

- **AI (Agente de Interface)**, este agente tem o intuito de gerir o sistema, comportando-se com uma ponte entre os utilizadores e estações. Deste modo, é imperativo que seja capaz de executar algumas tarefas, como adicionar/remover utilizadores ou estações, entre outros.
  Este agente é, portanto, responsável pelo registo da informação no sistema e, para tal, precisa de guardar uma lista com os vários utilizadores no sistema, bem como as estações pertencentes.
  Como se trata de um sistema onde são prestados serviços, é também necessário guardar as suas receitas/faturação e, como este agente possui a responsabilidade mais "global", mostra-se o melhor candidato para armazenar esta informação;
- **AE (Agente de Estação)** é uma extensão da classe auxiliar `InformAE`. Para além do identificador único (*id_AE*) e das suas coordenadas (*posicao*), tem de armazenar também o número de bicicletas disponíveis para aluguer, bem como o número de lugares reservados (isto é, número de utilizadores/bicicletas que se dirigem para a estação), sendo que estes dois valores não podem exceder a capacidade máxima, valor esse também armazenado na classe.
  Também é necessário armazenar uma lista com todos os potenciais clientes (isto é, os utilizadores dentro da sua `APE`), sendo que esta é atualizada sempre que um novo utilizador se encontra na `APE` da estação e no último $\frac{1}{4}$ do percurso da viagem. Para isso, é necessário que o `AU` envie a sua posição para a *AE*, para que esta saiba quais os utilizadores dentro da sua `APE` (esta é caracterizada pelo raio de ação de cada estação).
  De realçar que este agente tem funções para reservar/cancelar reservas de bicicletas e calcular descontos consoante o seu estado;
- **AU (Agente de Utilizador)**, caracterizado pelo seu estado, este é a extensão da classe `InformAU`, em que cada utilizador é identificado pelo *id_AU*, possui um estado que indica se se encontra em deslocamento ou em espera e, ainda, três posições - as coordenadas do agente no início da viagem (*pos_inicial*), as coordenadas da estação para a qual se dirige (*pos_dest*) e as coordenadas atuais (*pos_atual*). De realçar que a posição atual é constantemente atualizada ao longo da viagem.
  O `AU` guarda, ainda, o preço da viagem, a distância a percorrer e a velocidade no seu trajeto, bem como se já aceitou ou não algum desconto oferecido.
  Em termos de capacidade, este irá deliberar sobre as várias propostas que lhe são feitas (utilizando valores *random* que caracterizam a predisposição de aceitar/negociar certo tipo de propostas, bem como regras previamente estipuladas), calcular a percentagem de trajeto já percorrido, atualizar o seu destino na eventualidade de aceitar uma proposta alternativa, entre outros.

Sendo assim, estas três classes, juntamente com as classes auxiliares (definidas no *package* de negócio), irão permitir um funcionamento adequado do sistema.

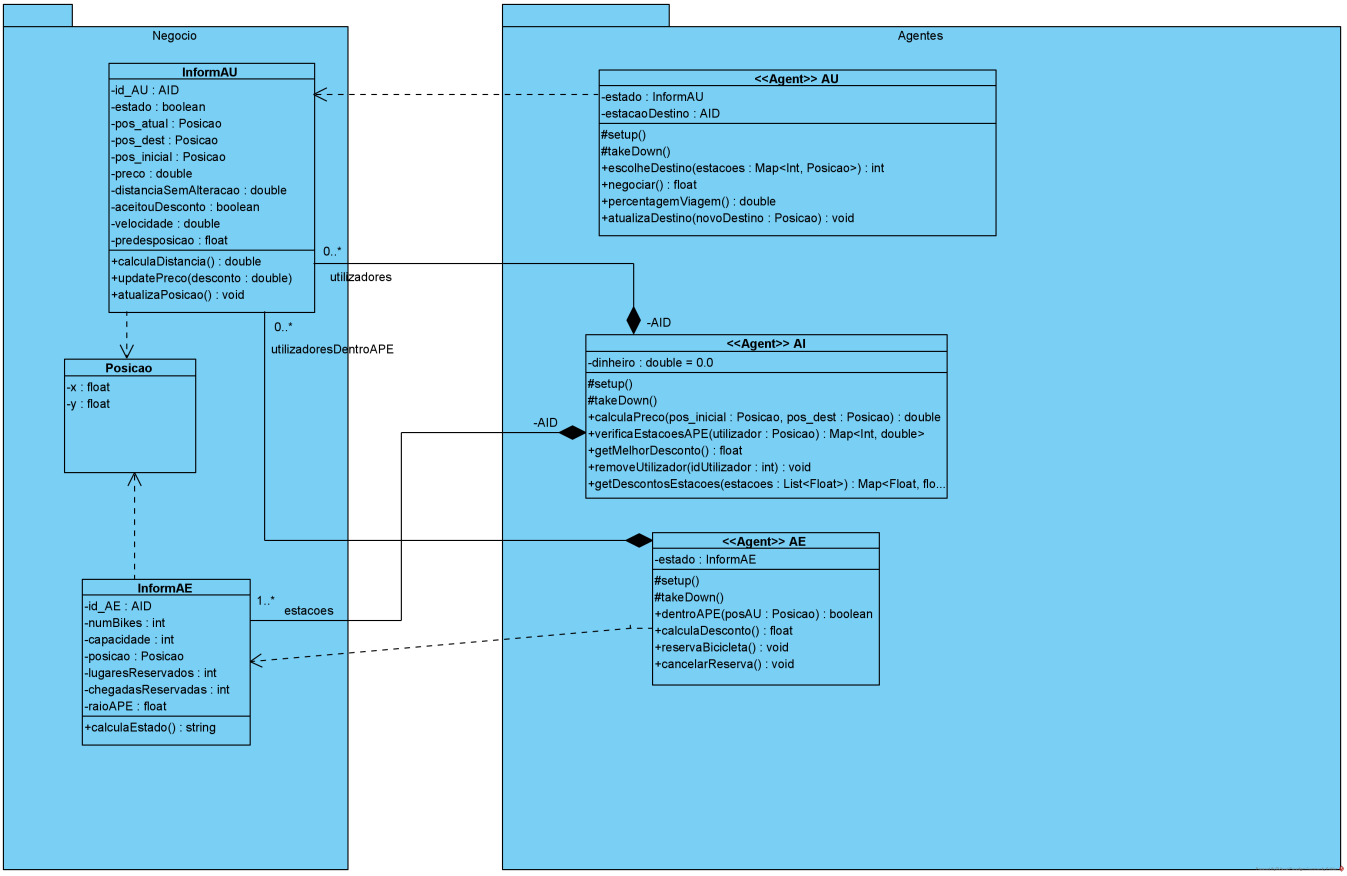

Figura 1. Diagrama de Classes do sistema

Quanto às propostas oferecidas, estas têm associado um desconto, que será um *float* de 0 a 1, sendo esta é a percentagem de desconto sobre o preço da viagem. O facto de o `AI` guardar e atualizar os vários descontos, faz com que apenas seja necessário verificar que estações podem apresentar-se como uma alternativa para o utilizador.
Outra particularidade do nosso sistema prende-se no facto de, quando uma estação é registada, esta não estará sempre na sua capacidade máxima. Isto permite que no início do funcionamento do sistema, exista alguma flexibilidade para os utilizadores circularem entre estações. Outra particularidade é a proporcionalidade entre a capacidade da estação e o raio da sua `APE`.

2.2 *Performatives*

Visto que o objetivo do trabalho prático ser a simulação de um SPB, é de esperar que os vários agentes comuniquem (para o bom funcionamento do sistema) e o balanceamento das bicicletas pelas várias estações. Para representar estas interações recorremos a um diagrama de *comunicação* que reflete o tipo de mensagens que podem ser trocadas dentro do sistema nas diversas situações.
Estas comunicações são baseadas nas normas FIPA, *performatives*, disponíveis na plataforma JADE.[2]
Então, as comunicações entre as várias entidades utilizam as seguintes *performatives*:

- **AI**:
  - ACCEPT_PROPOSAL / REJECT_PROPOSAL, para indicar se a estação foi adicionada ao sistema;
  - PROPOSE, para indicar/propor o preço previsto ao AU;
  - INFORM, para informar o AU das estações do sistema;
- **AE**:
  - SUBSCRIBE, para iniciar o pedido de registo no sistema;
  - INFORM / FAILURE, para indicar a disponibilidade da estação;
  - PROPOSE, para enviar os descontos a oferecer aos utilizadores na sua APE;
- **AU**:
  - REQUEST, para iniciar o processo de aluguer e escolher estação final, entre outros;
  - ACCEPT_PROPOSAL / REJECT_PROPOSAL, para indicar se o preço previsto é aceitável ou não, entre outros;
  - INFORM, para atualizar o sistema da sua posição atual, entre outros;
  - CONFIRM, confirmar o fim da viagem;
  - CANCEL, para informar uma estação para cancelar a reserva.

Quando for recebida uma mensagem que um dos agentes não sabe processar, poderá ser enviada uma mensagem com a *performative* UNKNOWN.

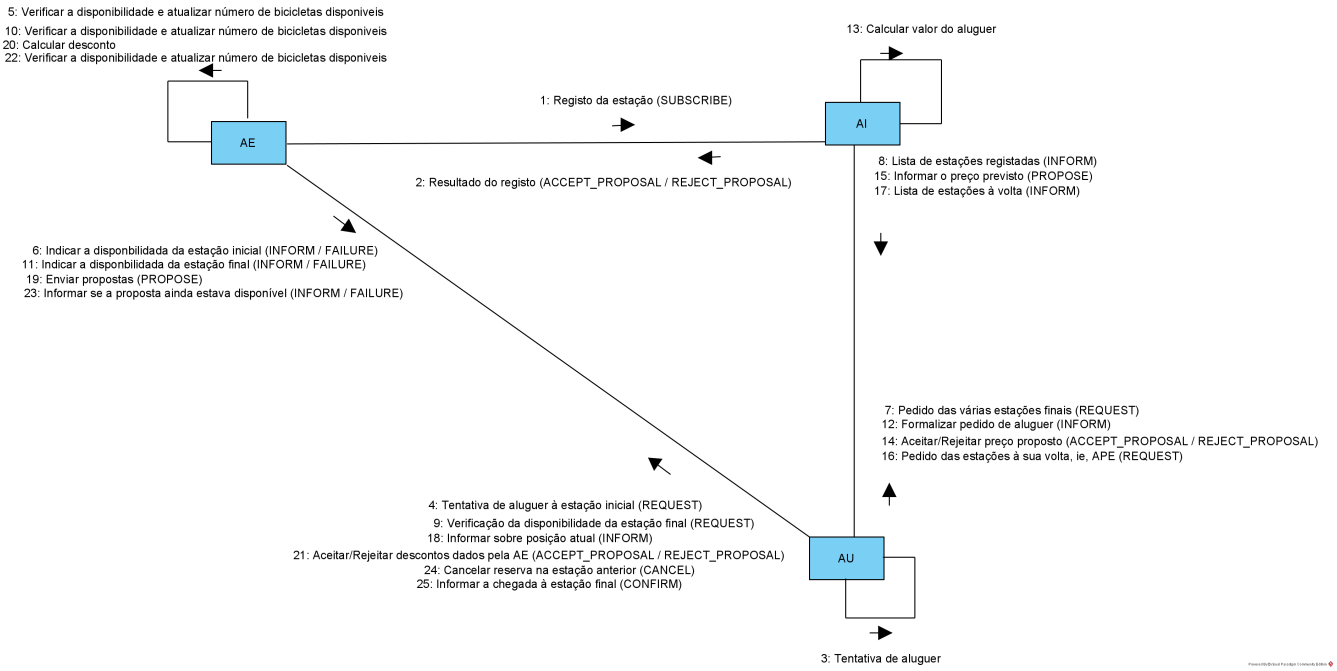

Figura 2. Diagrama de Comunicação do sistema

2.3 *Behaviours*

Na arquitetura deste sistema, são várias as tarefas que terão de ser executadas repetidamente como, por exemplo, o processamento das comunicações do `AI`. Todavia, também existem procedimentos que apenas são executados uma vez, por exemplo, uma tentativa de registo de estação. Desta forma, será necessário definir alguns *behaviours* para cada agente. Portanto, para cada agente, terão de ser implementados os seguintes *behaviours*:[1]

- **AI**:
  - `CyclicBehaviour`, para processar as várias mensagens vindas quer de `AUs` quer de `AEs`. Este deve processar continuadamente para responder a todos os pedidos.
- **AE**:
  - `OneShotBehaviour`, para o registo da estação, pois trata-se de uma ação que apenas deve ser executada uma vez;
  - `CyclicBehaviour`, para poder responder aos pedidos de aluguer e atualizar a sua ocupação, assim como manter descontos atualizados consoante a sua ocupação e negociar com os `AUs`.
- **AU**:
  - `OneShotBehaviour`, para tentar iniciar um pedido de aluguer e, visto também se tratar de uma ação que apenas deve ser executada uma vez, caso seja rejeitado, o agente sai do sistema;
  - `TickerBehaviour`, para atualizar a sua posição durante a viagem e manter a comunicação com o sistema, como por exemplo, informar sobre a sua posição e negociar com o `AEs`.

O seguinte diagrama demonstra o processo interno para os vários agentes pertencentes ao sistema e, como pode ser constatado, grande parte dos estágios dos agentes tratam-se de ações recursivas, isto é, procedimentos repetitivos.

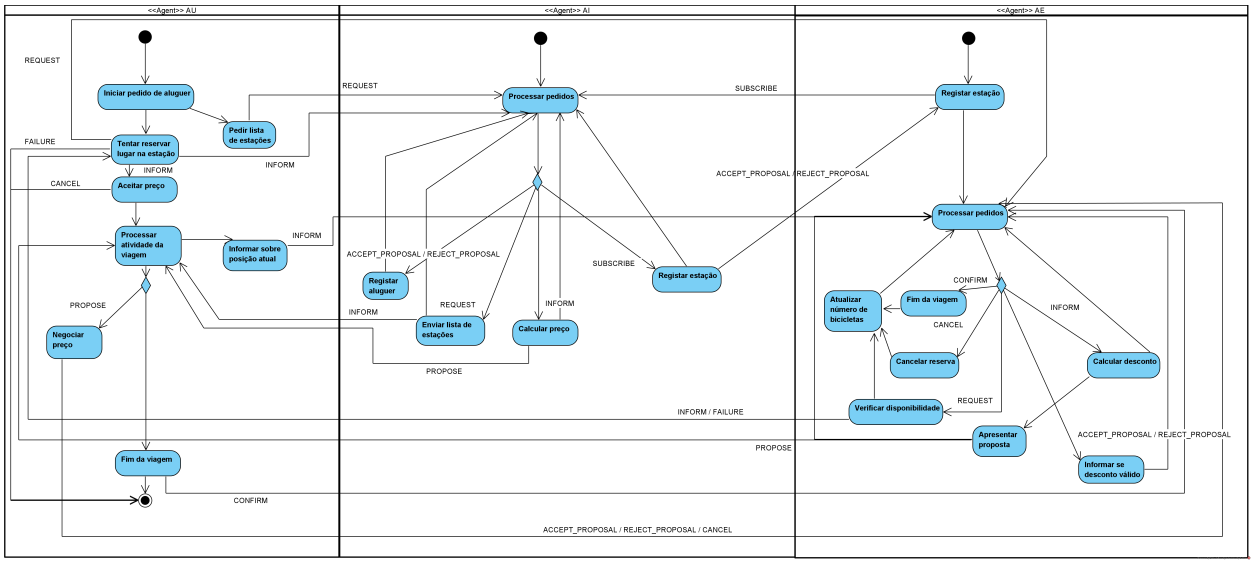

Figura 3. Diagrama de Atividade do sistema

[1] Possivelmente terão de ser adicionados novos *behaviours* para separar comportamentos dos vários agentes

3 Interação/Comunicação entre agentes

3.1 Pedido de aluguer e interação durante a viagem

Para o processo de aluguer, o Agente Utilizador (AU) inicializa a *App* ao lado da estação, de modo a poder utilizar o identificador associado à ela (por exemplo, um *QRCode*), de modo a estabelecer ligação com o Agente Estação (AE) para iniciar o processo de aluguer permitindo, assim que o AU prove que se encontra lá (1).

De seguida, caso não haja bicicletas, o AE informa o AU (2) e este sai do sistema (3), caso contrário o AE reserva uma bicicleta decrementando o seu número (4).

Depois da interação com o AE, o AU entra em comunicação com o AI para informar do início de aluguer (5). O AI, então, calcula a lista de estações de destino possíveis (6) e envia-a ao AU (7).

Neste momento, o AU escolhe a estação de destino pretendida e comunica a decisão ao AI (8) e, este último, calcula o preço da viagem pretendida (9), enviando de volta uma mensagem com o valor para o AU (10).
Neste ponto o AU tem duas opções:

- Recusar o preço apresentado, informando o AI (13) e, consequentemente, enviando mensagem ao AE de origem para cancelar a reserva da bicicleta anteriormente realizada (14) (e, por sua vez, o AE aumenta o número de bicicletas disponíveis (15)) e sai do sistema (16);
- Aceitar o preço (11) e o AI atualizar a posição do AU para a posição da estação inicial, adicionando-o ao *Map* de utilizadores (12).

Se o AU continuar no sistema, este vai tentar reservar lugar na estação destino (17). Caso a resposta seja negativa (18), informa a estação inicial para cancelar a reserva da bicicleta (19), o que vai fazer com que esta aumente o número de bicicletas disponíveis (20) e, por consequência, provoca a sua saída do sistema (21). Caso existam lugares livres na estação de destino, esta reserva um lugar para a bicicleta (22) e o AU comunica ao AI que pode iniciar a viagem (23).

Durante a viagem, o AU atualiza sempre a sua posição, assim como a percentagem percorrida da viagem (24). Quando esta percentagem passar os 75%, o AU vai passar a enviar mensagens, regularmente, às AEs, quando entra nas suas APEs, para dar essa informação, iniciando, assim, um processo de negociação entre eles (25/26), que será apresentado noutro diagrama.

Quando finalmente chegar ao destino, informa o respetivo AE de destino (27), levando-o a aumentar o seu número de bicicletas disponíveis (28). Informa também o AI, enviando o respetivo pagamento (29). Por fim, o AI remove o AU do *Map* de utilizadores em viagem (30) e informa AU sobre o fim do aluguer (31) e este sai do sistema (32).

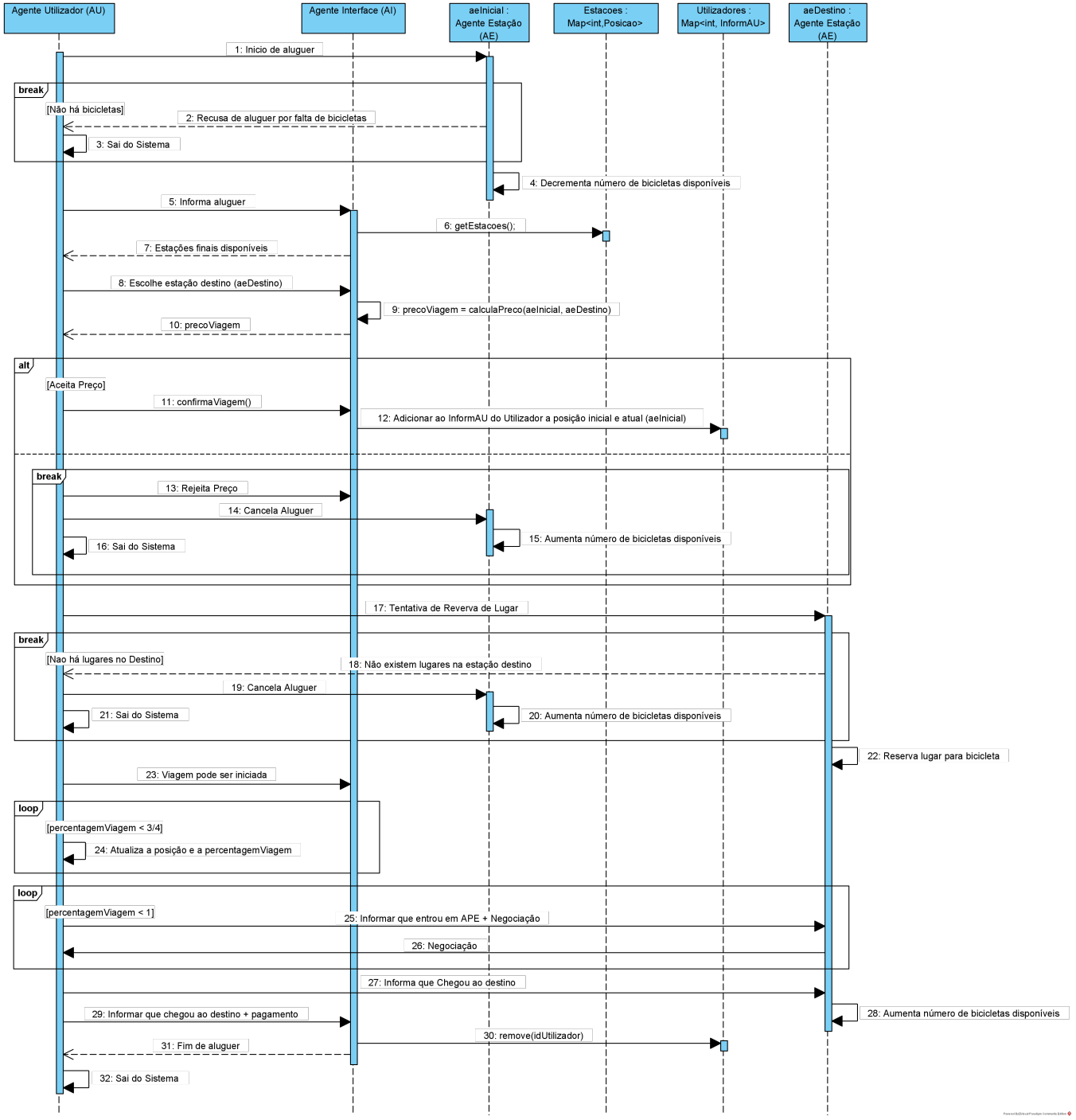

Figura 4. Diagrama de Sequência de um aluguer

3.2 Registo de estação

O processo de registo de uma estação é um processo bem mais simples. Aqui o `AE` apenas procura pelo `AI` nas "páginas amarelas" (`DFService`) e envia um pedido de registo com a informação necessária, capacidade máxima, número de bicicletas atual, entre outros.
De seguida, se o pedido for válido, o `AI` adiciona esta estação à sua lista, caso contrário, rejeita-o. Por fim, informa a estação do resultado do pedido.

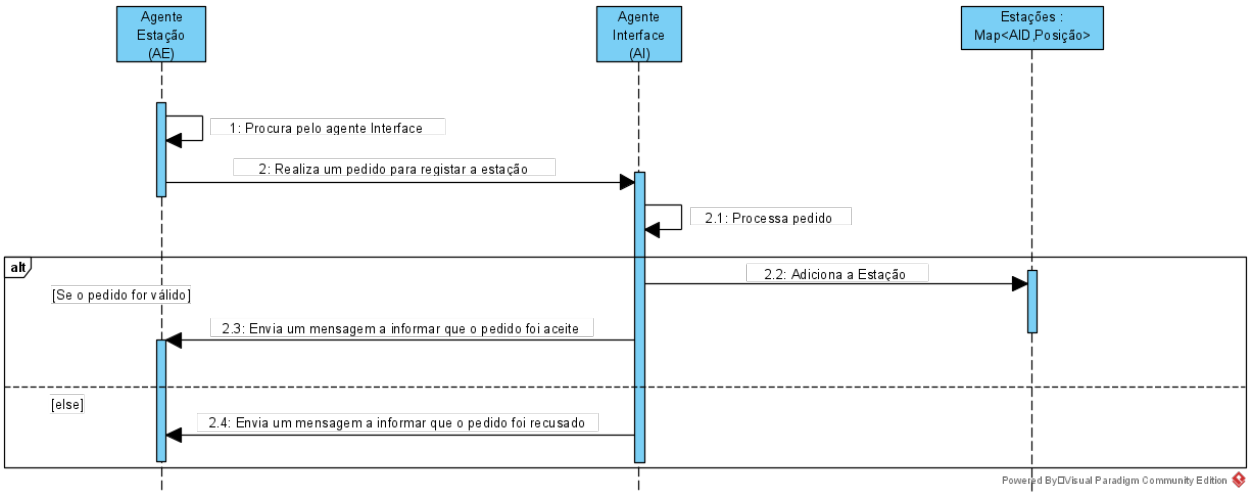

Figura 5. Diagrama de Sequência de um registo de uma estação

3.3 Negociação

Quanto ao processo de negociação, este tem um papel muito relevante na *performance* do sistema. Existe uma certa '*Coopetição*', (uma estratégia de negócios baseada na Teoria dos Jogos), isto é, os `AEs` terão de competir para garantir o aluguer, quando têm poucas bicicletas disponíveis, apresentando propostas e descontos muito vantajosos que aliciem os clientes. Assim, cada estação tem como objetivo evitar a escassez de recursos (neste caso, bicicletas) de forma a poder continuar a operar. Por outro lado, quando estiver congestionada, esta deverá oferecer propostas menos vantajosas com vista a manter o equilíbrio do número de bicicletas existentes nos sistemas.
Consequentemente, esta competição resulta, de certa forma, em cooperação, pois as estações trabalham para um bem comum - o balanceamento e descongestionamento do sistema. Ainda assim, os `AE` podem ser vistos como agentes competitivos, uma vez que procuram maximizar a sua utilidade (visto terem apenas acesso ao seu estado e não ao estado de outras `AEs` ou até do sistema), ainda assim, têm certos comportamentos cooperativos.

Outra situação onde se aplica a dualidade 'competição *vs.* cooperação' é na apresentação de descontos ao `AU`, já que um `AU` e `AE` tentarão encontrar propostas que beneficiem ambos. Assim, um `AU` poderá optar por manter o trajeto, com vista a chegar à estação inicialmente pretendida. No entanto, também poderá decidir cooperar com a estação que lhe propõe o desconto e deslocar-se para um novo destino, equilibrando o congestionamento do sistema. Esta negociação está representada no diagrama seguinte.

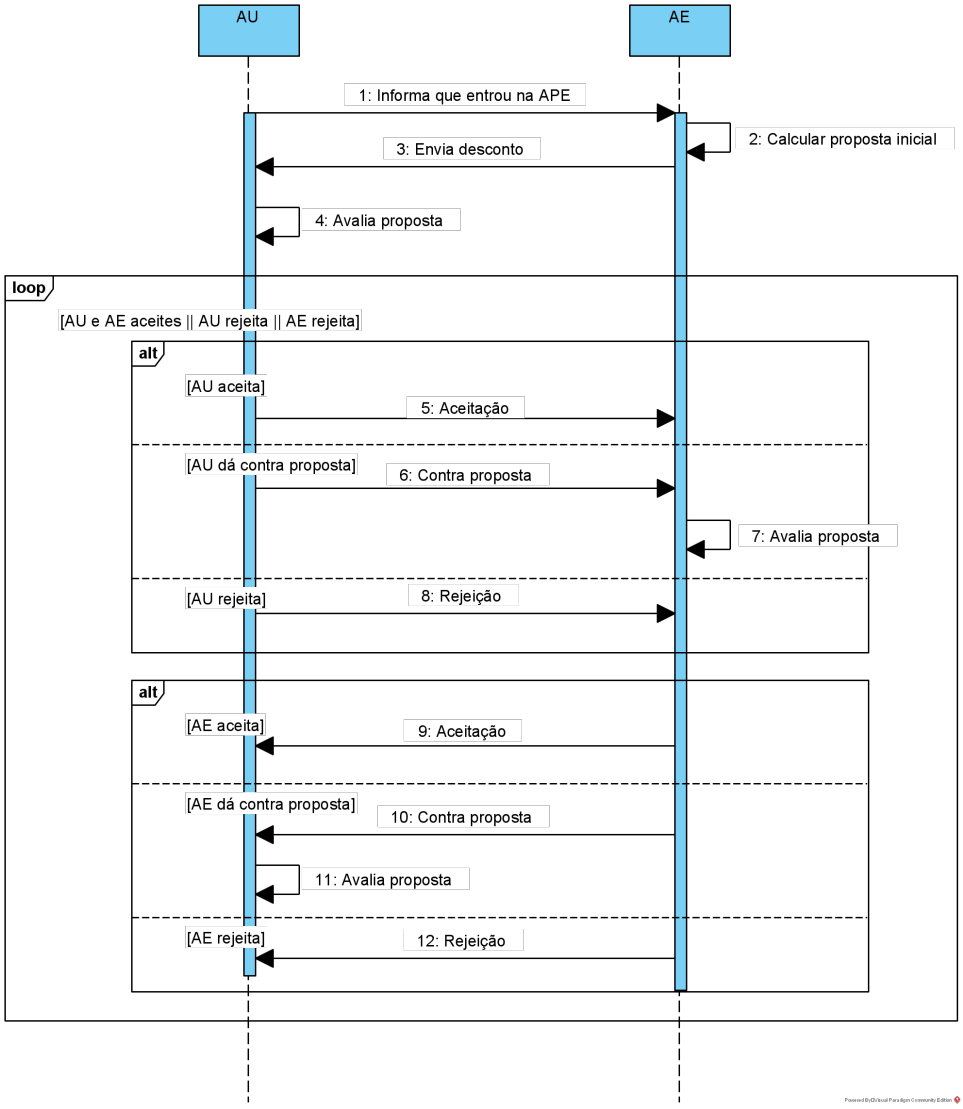

Figura 6. Diagrama de Sequência da negociação entre estação e utilizador

O início de uma negociação é efetuado quando o `AU` entra na `APE` de um determinado `AE`. Aí, o `AU` informa o respetivo `AE` que entrou na sua `APE` (1), o que provoca um cálculo de proposta inicial por parte deste (2), sendo esta enviada ao `AU` (3) que vai, prontamente, avaliá-la (4).

A partir daqui, inicia-se o processo de negociação propriamente dito, onde o `AU` tanto pode aceitar (5), rejeitar (8) ou dar uma contra proposta (6), o que vai levar a uma avaliação por parte do `AE` (7) que, por sua vez, pode aceitar (9), rejeitar (12) ou dar uma contra proposta (10), desencadeando uma nova avaliação por parte do `AU`, voltando ao início deste processo de negociação. Este é terminado assim que os dois entrem em acordo, ou seja, ambos aceitem a proposta ou, então, logo que uma das partes rejeite uma proposta.

4 Conclusão

Como foi referido ao longo do presente relatório, o objetivo principal é o desenvolvimento e especificação dos vários agentes e da interação e comunicação entre os mesmos. Assim sendo, realizámos, inicialmente, uma descrição geral do problema, onde foi feito um levantamento de requisitos e analisado cada componente útil para o sistema. De seguida, foi feita a descrição de cada agente, das suas funcionalidades e comportamento.
As várias secções serão acompanhadas por diagramas `AUML`, para permitir uma melhor compreensão de cada etapa elaborada.

O planeamento do sistema multi-agente que visa monitorizar e implementar um serviço `SPB` é, sem dúvida, uma fase muito importante para o próximo passo - a implementação da simulação do sistema.
As tarefas mais importantes são o mecanismo de balanceamento de bicicletas entre as várias estações e o equilíbrio do processamento pelos vários agentes, pois, se o `AI` for sobrecarregado e tiver demasiadas responsabilidades, pode tornar-se um *bottleneck* para o sistema, pelo que, a decisão da atribuição de descontos deve ser feita pelas `AEs`.

Após alguma investigação, concluímos que este tipo de sistema se trata de um sistema de *Automated stations*, arquitetura esta que consiste no aluguer e receção de bicicletas entre as várias estações, todas pertencentes ao mesmo sistema. As estações necessitam de estar equipadas com um tipo de bloqueio especial, de forma a que apenas libertem bicicletas através do controlo de um computador.

Existem várias vantagens para o meio ambiente num sistema deste tipo, tais como a redução de emissões de poluição e ruído, bem como a melhoria do bem estar da população, reduzindo o congestionamento em áreas urbanas e aumentando a sua mobilidade, sendo, ainda, uma boa forma de praticar exercício físico e um meio de transporte mais saudável. Todavia, existem algumas desvantagens como a criação de estações (que é, normalmente, feita à custa da redução de lugares de estacionamento disponíveis para automóveis), o risco de vandalismo, acidentes ou roubos sendo, por isso, necessário implementar mecanismos para responsabilizar o utilizador pelos seus atos, bem como a implementação de alguns sistemas de segurança nas estações e bicicletas.
Outra preocupação que deve ser tida em conta é a questão da segurança, pois é necessário que a estação possa fornecer algum tipo de equipamento, como capacetes, entre outros.

Em suma, para um sistema deste estilo funcionar, é, então, necessário garantir certas condições, como manter o sistema equilibrado (isto é, manter um certo equilíbrio nas quantidades de bicicletas disponíveis em cada estação, de modo a que nenhuma estação fique congestionada com bicicletas e sem clientes ou sem bicicletas para alugar e bastante procura) para permitir uma melhor *performance*. Visto que o sistema a desenvolver deverá ser automatizado, é também necessário garantir a interação entre as várias estações do sistema e consequente colaboração.

Referências

1. Brutus, S., Javadian, R. & Panaccio, A.J. (2017); '**Cycling, car, or public transit: a study of stress and mood upon arrival at work**'. *International Journal of Workplace Health Management*, vol. 10, no. 13.
2. **FIPA** (*Foundation for Intelligent Physical Agents*). Disponível em: *http://www.fipa.org*
3. **Introdução a um sistema de partilha de bicicletas**. Disponível em: *https://en.wikipedia.org/wiki/Bicycle-sharing_system*
4. Kloimüllner, C., Papazek, P., Hu, B. & Raidl, G. (2014). '**Balancing Bicycle Sharing Systems: An Approach for the Dynamic Case**'. *Evolutionary Computation in Combinatorial Optimisation*.