Universidade do Minho
Departamento de Informática
Mestrado Integrado em Engenharia Informática

TP1 - Relatório Fase II

SMA: Sistema de Partilha de Bicicletas

Modelação, especificação e implementação

Grupo de trabalho 6

Miguel Miranda, A74726

Braga, 8 de Dezembro de 2017

1. Abstract

O contexto de criação do projeto foca-se no desenvolvimento e criação de agentes que constituam e simulem um Sistema de Partilha de Bicicletas. Estes podem representar estações de aluguer de bicicletas ou, os utilizadores deste tipo de serviços, que desempenham o processo de aluguer, desde a requisição à entrega dos referidos equipamentos.

O desenvolvimento do projeto foi dividida em duas fases, sendo que inicialmente foi criada uma arquitetura de agentes e de protocolos de comunicação entre os mesmos, que permitissem dar suporte aos contextos de interação do sistema. A fase de implementação passa pelo desenvolvimento, em linguagem *java* e com extensão ao ambiente de desenvolvimento de agentes *jade*, dos conceitos previamente modelados.

Conteúdo

2. Introdução

No presente relatório será descrito o processo desenvolvido para a concessão, estruturação e implementação do trabalho prático da unidade curricular de Agentes Inteligentes, que se foca na criação de um sistema de partilha de bicicletas (*SPB*).

Sendo esta implementação associada à utilização de agentes inteligentes para gerir o funcionamento do SPB, procura-se assim simular as interações entre as entidades que o corporizam. Para isso foram criados os agentes estação (*AE*), que representam os pontos físicos de requisição e devolução das bicicletas, e os agentes utilizadores (*AU*), que procuram simular o aluguer de bicicletas por parte dos clientes do sistema.

Nesta implementação de um sistema multi agente, destaca-se ainda a introdução de uma forma de negociação entre os dois principais agentes referidos. O agente estação, conforme as variações na sua taxa de ocupação, pode tentar convencer determinados utilizadores alvo a alterar o local de entrega das bicicletas. Nestas situações, cabe ao agente utilizador avaliar as eventuais vantagens das propostas feitas por uma determinada estação, sendo que este poderá aceitar uma proposta feita por uma estação, recusa-la ou fazer uma contra proposta. Este processo de persuadir os utilizadores a alterarem o seu destino é realizado através da oferta de descontos no preço da sua viagem.

No geral, foi idealizada uma arquitetura dividida em três módulos: dois que representam a existência e funcionalidades dos agentes *AE* e *AU* supracitados e, um terceiro, visto como a interface responsável pela tarefa de inicialização e visualização/interação da evolução do sistema.

O relatório do projeto apresenta assim uma explicação detalhada e pormenorizada dos componentes da arquitetura desenhada, assim como a referência às principais funcionalidades essenciais, associadas com o domínio do problema.

Levantada a arquitetura, serão apresentados os protocolos e formas de comunicação entre os agentes do sistema, e descrito o processo de implementação da arquitetura, com referência aos comportamentos que caracterizam os agentes do sistema.

Como comprovação das funcionalidades implementadas, será numa fase final realizada uma demonstração dos resultados obtidos e de alguns cenários de interação relevantes.

3. Arquitetura da solução implementada

3.1 Descrição geral

Tendo como principal objetivo a gestão de um sistema de aluguer de bicicletas, foram levantados como principais entidades as estações onde se localizam as bicicletas e os clientes que executam o aluguer e entrega de equipamentos nas mesmas.

Para permitir a monitorização de vários sensores virtuais, que ao longo do tempo capturam a localização *GPS* dos utilizadores que reservam bicicletas e comunicam esses dados às estações próximas, foi estruturada uma arquitetura que permitisse gerir estes sensores e a troca de mensagens entre as diferentes entidades, representadas sobre a forma de agentes.

Por uma questão de organização e pelas funcionalidades que desempenham, esta arquitetura pode ser divida em três subsistemas principais:

- Módulo Agente Estação;
- Módulo Agente Utilizador;
- Módulo Agente Interface;

Toda a implementação do projeto será realizada utilizando a linguagem *java* conjugada com as bibliotecas disponibilizadas pelo ambiente de desenvolvimento *JADE*.

A seguinte imagem procura representar conceptualmente as funcionalidades levantadas e que estão presentes na arquitetura final do sistema. Este esquema serve assim como base para compreender as seguintes secções, onde se abordará a forma de comunicação entre os agentes e algumas das classes implementadas.

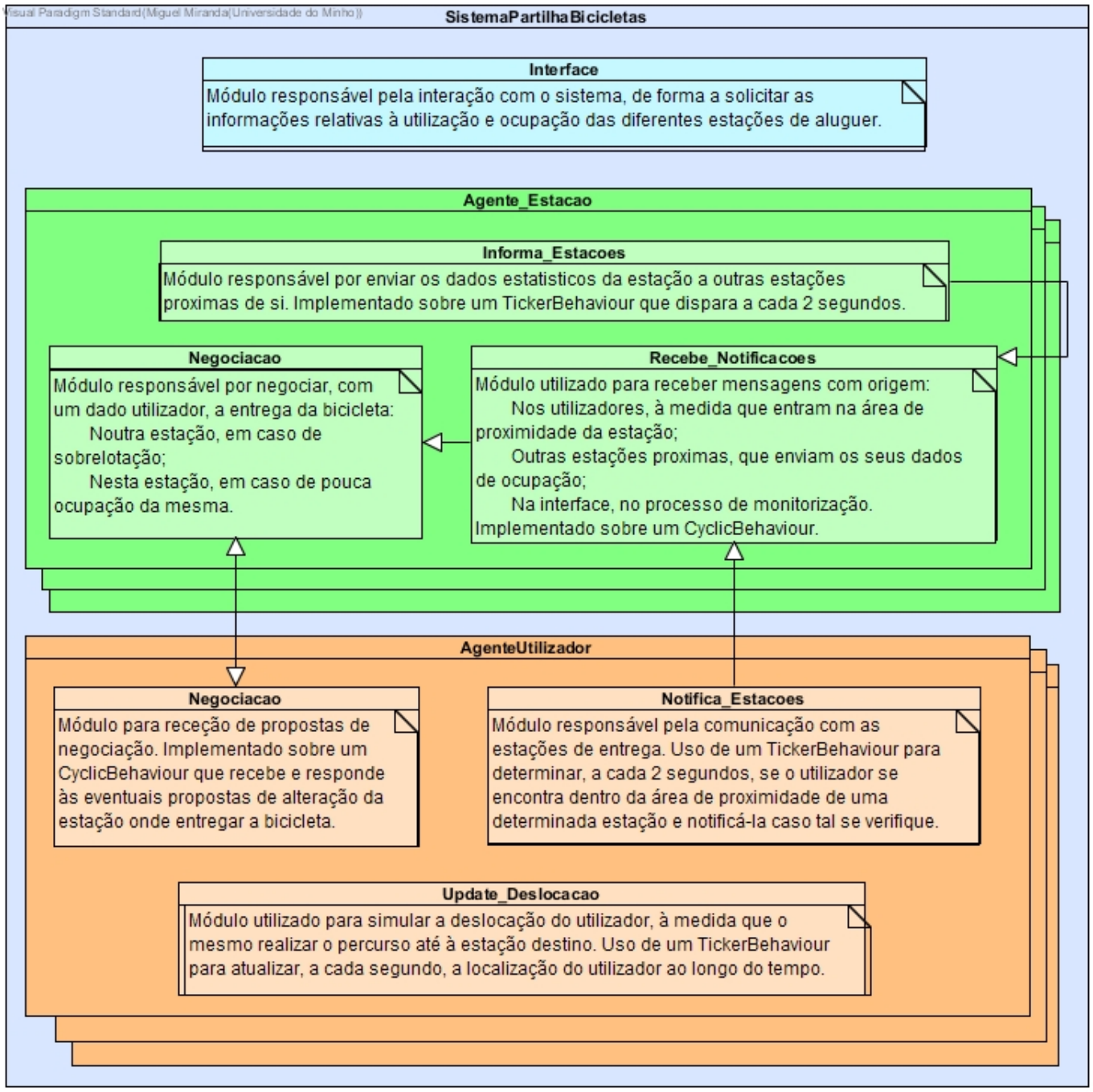

Figura 3.1: Esboço conceptual da arquitetura da solução implementada

3.1.1 Módulo *Sistema Partilha Bicicletas*

O módulo *SistemaPartilhaBicicletas (SPB)* é responsável pela inicialização e posterior interação com o sistema, à medida que o mesmo evolui, podendo ser visto como o agente *Interface* definido no enunciado do problema.

Cabe a este agente a criação de **N** estações iniciais, sendo este número de estações fixo e imutável ao longo do tempo, ou seja, uma vez criada uma estação de aluguer numa determinada localização, esta não deixará de existir ou alterar a sua localização. Este parâmetro **N** será solicitado ao utilizador, na fase de criação do sistema.

Além disto, este agente será também responsável por gerar, de forma regular e aleatória, os agentes utilizadores do sistema, simulando assim o fluxo de chegada de clientes que procuram utilizar estes serviços de aluguer de bicicletas.

Relativamente ao processo de interação com o sistema, serão implementados métodos que permitam observar a evolução da ocupação das estações ao longo do tempo, através da visualização das estatísticas de cada uma das estações.

Estas funcionalidades serão implementadas na classe designada *Agente_Interface_SPB*. O seguinte *system role* procura descrever de forma geral esta funcionalidade.

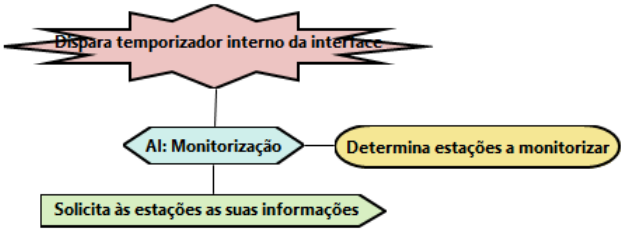

Figura 3.2: *System role* da tarefa de monitorização no agente *Interface*

3.1.2 Módulo *Agente Estação*

O módulo *AgenteEstação (AE)* representa o conjunto de classes onde se irão incorporar todas as funcionalidades e métodos necessários para gerir a utilização e recursos de uma estação de aluguer.

De uma forma geral, estas funcionalidades englobam:

- A gestão balanceada do número de bicicletas disponíveis na estação;
- A receção das mensagens com origem nos sensores das bicicletas dos utilizadores, à medida que estes entram na área de proximidade da estação;
- A receção de mensagens oriundas das estações próximas, que partilham as suas informações de ocupação;
- O atendimento dos pedidos da interface, que regularmente solicita os parâmetros de utilização da estação.
- A receção e resposta às propostas de negociação realizadas com utilizadores;

Será na classe *Recebe_Notificações* que se efetua o processamento de todas as mensagens recebidas, independente do remetente. À medida que se analisam as mensagens enviadas pelos sensores, caso se verifique um problema na ocupação atual da estação e o utilizador possa ajudar na resolução desse problema, será então criada uma instância da classe *Negociacao* para iniciar um diálogo com o remetente da mensagem.

Esta interação é realizada através da troca cooperativa de propostas com os agentes utilizadores, de forma a convencê-los a entregar a bicicleta na presente estação, em caso de baixa ocupação da mesma, ou entregar a bicicleta noutro ponto de entrega mais próximo, em caso de sobrelotação da estação inicialmente destinada para entrega do equipamento. Cada uma das estações, perante esta troca de mensagens entre os utilizadores, guarda um histórico de negociação associado ao cliente que recebe uma determinada proposta.

Ainda na classe *Recebe_Notificacoes* é feito:

- O tratamento das mensagens recebidas por parte do agente *Interface_SPB*, para solicitar à estação as suas informações de utilização;
- O tratamento das mensagens vindas das outras estações, no seu processo mútuo de troca de informações.

O envio de mensagens para as estações próximas, será realizado pela classe *Informa_Estacoes*. O conhecimento das estações mais próximas estará guardado numa lista que é atualizada ao longo do tempo e, no conteúdo das mensagens de informação, serão enviados os dados mais importantes guardados nas variáveis de uma estação.

O processo de informar as outras estações ou de responder aos pedidos do agente *Interface*, são descritos nos seguintes *system roles*. Devido à semelhança destes dois papeis, é possível notar que estes partilham os mesmos goals mas são desencadeados por motivos diferentes.

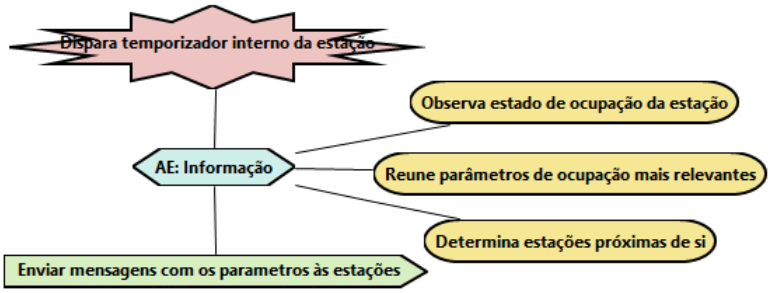

Figura 3.3: *System role* do processo de informação do agente estação para com outras estações

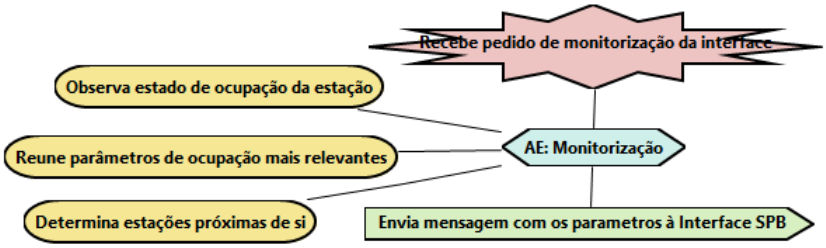

Figura 3.4: *System role* dos pedidos de monitorização realizados ao agente estação

3.1.3 Módulo *Agente Utilizador*

No módulo *Agente_Utilizador (AU)* serão implementadas as funcionalidades relacionadas com os sensores virtuais, que se encontram nas bicicletas dos utilizadores.

Quando um cliente realizar o aluguer de uma bicicleta, a partir do momento em que este inicia o seu percurso até uma estação destino, o sensor do equipamento alugado irá de forma regular:

- Determinar se o utilizador entrou dentro da área de proximidade de uma determinada estação, sabendo que a localização das estações e o seu raio de proximidade são parâmetros conhecidos ao sensor;
- Verificar se o utilizador já realizou mais de 3/4 do seu percurso.

Se estas duas condições forem verificadas, será então enviada uma mensagem de notificação à estação sobre a qual se entrou na área de proximidade. Esta funcionalidade será implementada na classe designada por *Notifica_Estacoes*.

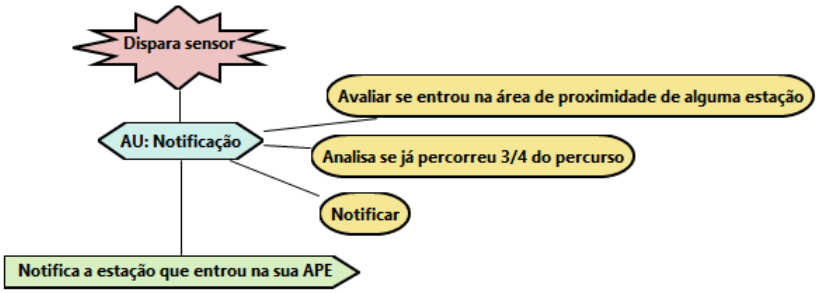

Figura 3.5: *System role* da tarefa de notificação no agente utilizador

Ao ser a implementação do sensor, no equipamento alugado, a determinar a sua localização perante a área de proximidade das estações, evita-se assim o envio de mensagens desnecessárias para as mesmas. Ou seja, só se notifica uma estação quando de facto estamos num cenário onde pode ser relevante para o agente estação entrar em possível negociação com o utilizador.

Tal como decorre nos agentes *Estação*, a capacidade de resposta a estes cenários de negociação, será nos utilizadores suportada pela criação de instâncias da classe *Negociacao*, guardando assim um histórico da troca de mensagens com uma determinada estação, até que um dos lados tome a posição de aceitar ou rejeitar a oferta.

Para simular a deslocação do utilizador, existe ainda a classe *Update_Deslocacao* que de forma regular vai incrementando a posição do utilizador, no seu percurso até à estação destino.

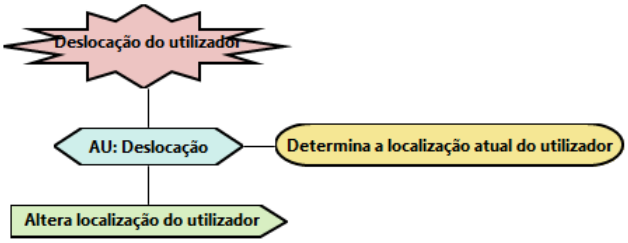

Figura 3.6: *System role* do processo deslocação no agente utilizador

3.2 Variáveis

Depois de apresentadas as principais classes do problema, serão nesta secção abordadas algumas das variáveis associadas a cada uma delas, de forma a permitir o desenrolar do sistema e guardar o estado atual de cada ator no sistema.

3.2.1 Classe *Agente Estação*

De uma forma geral, para a classe *Agente_Estacao* foram definidas as seguintes vaiáveis:

- **vizinhança**: Estrutura de dados onde se guardam e atualizam as informações relativas às estações conhecidas por uma dada estação. As informações são atualizadas no processo de troca de mensagens entre estações.

  Esta estrutura será útil quando, em situações de sobrelotação, é necessário escolher uma nova estação destino. Esta escolha não será aleatória mas sim tendo em conta os parâmetros de ocupação das estações próximas.

  De outra forma, uma estação poderia encaminhar um utilizador para uma estação também ela sobrelotada, o que levaria a uma nova realocação e processo de negociação com o utilizador. Estas deslocações recorrentes levariam, em contextos reais, ao aborrecimento e descontentamento do utilizador com o serviço.

- **negociacoes**: Estrutura de dados onde se guardam as negociações iniciadas pela estação. A estrutura mapeia o *AID* do utilizador que se abordou para uma estrutura do tipo *Negociacao* onde se guarda o histórico da negociação com o mesmo.

  Estas negociações podem já ter terminado ou ainda estar a decorrer. Nesta última situação, o agente *Estacao* atualiza o estado para "terminado"quando a mesma chegar a um fim;

- **estacao**: Estrutura de dados do tipo *Estacao* (ver secção 3.3.2).

3.2.2 Classe *Agente_Utilizador*

Para a entidade *Agente_Utilizador*, as variáveis definidas podem descrever-se da seguinte forma:

- **utilizador**: Estrutura de dados do tipo *Utilizador* (ver secção 3.3.1);

- **negociações**: Estrutura de dados onde se guardam as negociações que envolvem o utilizador, de forma semelhante ao processo feito pelas estações.

  Sendo as negociações realizadas com estações de aluguer, a estrutura mapeia o *AID* do agente *estação* que desencadeou a proposta, para uma estrutura do tipo *Negociacao* onde se guarda o estado da negociação com a mesma.

  Estas negociações podem já ter terminado ou ainda estar a decorrer. Nesta última situação, o agente *utilizador* vai atualizando o estado da negociação até que a mesma chegue ao fim.

3.3 Classes auxiliares

3.3.1 Classe Utilizador

Classe que procura representar os atributos que identificam um utilizador, nomeadamente:

- **origem**: variável do tipo *Estacao*, que identifica a estação onde o utilizador alugou o equipamento;

- **inicio**: variável do tipo *Posicao*, que identifica a localização onde o utilizador inicia o seu percurso. Esta posição coincide com a localização da estação origem;

- **destino**: variável do tipo *Estacao*, que identifica a estação onde o utilizador vai entregar o equipamento alugado.

  Esta estação pode mudar caso o utilizador aceite alguma proposta de negociação e altere o seu percurso inicial;

- **fim**: variável do tipo *Posicao*, que identifica a localização onde o utilizador termina o seu percurso. Esta posição coincide com a localização da estação destino;

- **atual**: variável do tipo *Posicao*, que identifica a posição atual do utilizador, à medida que o mesmo se desloca regularmente no seu percurso até à estação destino.

Relativamente os utilizadores, foi tomada a decisão de utilizar as três variáveis do tipo *Posicao* para que seja possível saber, em qualquer altura, qual a distância total que o utilizador vai percorrer e qual a distância já percorrida pelo mesmo.

A primeira é calculada através da distância entre as posições *inicio* e *fim* e a segunda através da distância entre *inicio* e *atual*. Estas duas medições serão úteis para determinar se o utilizador já percorreu ou não mais de 3/4 do seu percurso.

3.3.2 Classe Estacao

Classe que contém todos os atributos que identificam uma estação e caracterizam o seu estado atual, nomeadamente:

- **localização**: Cada estação de bicicletas terá uma posição, que corresponde a um tuplo com as coordenadas da mesma num referencial cartesiano.

  Esta localização será aleatória e criada pelo agente *Interface* no processo de inicialização do sistema;

- **area_proximidade**: Valor inteiro que representa o raio da circunferência que delimita a área circundante a uma estação.

  A partir deste valor determina-se se um determinado utilizador está ou não na área de proximidade da estação. Sendo **P(x,y)** a posição do utilizador e **r** o raio desta circunferência, o utilizador está na área de proximidade da estação se x^2 + y^2 <= r^2;

- **nr_bicicletas_atual**: Como o nome indica, para cada estação existirá um valor inteiro que representa o número de bicicletas que existem disponíveis para requisição num determinando instante;

- **capacidade_max**: Representa o número máximo de bicicletas que podem ser armazenadas simultaneamente numa estação.

  Se o *nr_bicicletas_atual* ultrapassar este valor, estaremos numa situação de sobrelotação da estação;

- **ocupacao_ideal**: Valor do tipo *double*, que representa a percentagem de bicicletas que idealmente deve existir na estação, de forma a dar resposta ao fluxo de cliente que a mesma recebe.

  Se a ocupação atual, extrapolada através dos parâmetros *nr_bicicletas _atual* e *capacidade_max*, for próxima da ocupação ideal, a estação tem sempre equipamentos disponíveis para a demanda da sua zona, levando à obtenção de um maior lucro.

  Se a ocupação atual for inferior a esta ocupação ideal, a estação vai procurar cativar utilizadores para realizar entregas no seu posto de aluguer.

- Além destas variáveis, outras farão parte da classe *Estação*, como forma de guardar estatísticas relativas à sua utilização:

- **nr_reservas**: contador do número total de reservas desempenhados na estação;
- **nr_entregas**: contador relativo ao número de entregas de bicicletas feitas na estação, seja por desvio do destino do utilizador ou porque de facto a estação já era o destino;
- **nr_props_aceites** e **nr_props_rejeitadas**: contadores do número de propostas feitas pela estação a utilizadores e que foram, respetivamente, aceites ou rejeitadas pelos utilizadores;
- **lucro**: Valor arrecadado pela estação até ao presente.

3.3.3 Classe Negociacao

Esta classe é utilizada nas variáveis do agente *Estacao* e agente *Utilizador*, que para um determinado AID de um agente mapeiam para uma instância desta classe *Negociacao*, como forma de arquivar o histórico de mensagens trocados entre dois agentes. Apresenta como variaveis:

- **user**: variável do tipo *Utilizador*, que representa o utilizador envolvido na negociação;
- **estacao**: variável do tipo *Estacao* que representa a estação envolvida na negociação;
- **historico**: lista com o conjunto de mensagens de negociação trocadas entre os dois agentes. No contexto atual do sistema, só a última mensagem desta lista é utilizada com relevo, mas em desenvolvimentos futuros, o histórico de negociações entre os agentes pode tomar influência na tomada de decisão;
- **terminada**: variável de teste usada para marcar o estado da negociação como ativa (a decorrer) ou terminada (recebeu AGREE ou REJECT);
- **desconto**: valor do último desconto trocado na negociação, evitando assim aceder ao histórico de propostas e realizar o parsing das mensagens guardadas;
- **propostas_trocadas**: contador do número de propostas trocadas entre os agentes. Quando este valor for superior a 5, os agentes desistem do processo de negociação.

3.3.4 Classe Posicao

Comum a praticamente todas as outras classes, existe ainda a classe *Posicao*, que contém duas variáveis, designadas **x** e **y**, como forma de retratar uma localização no referencial cartesiano.

Tendo sido apresentadas as principais variáveis, funcionalidades e formas de interação entre os agentes, a seguinte imagem procura representar o diagrama de classes que corporizam a solução final desenvolvida.

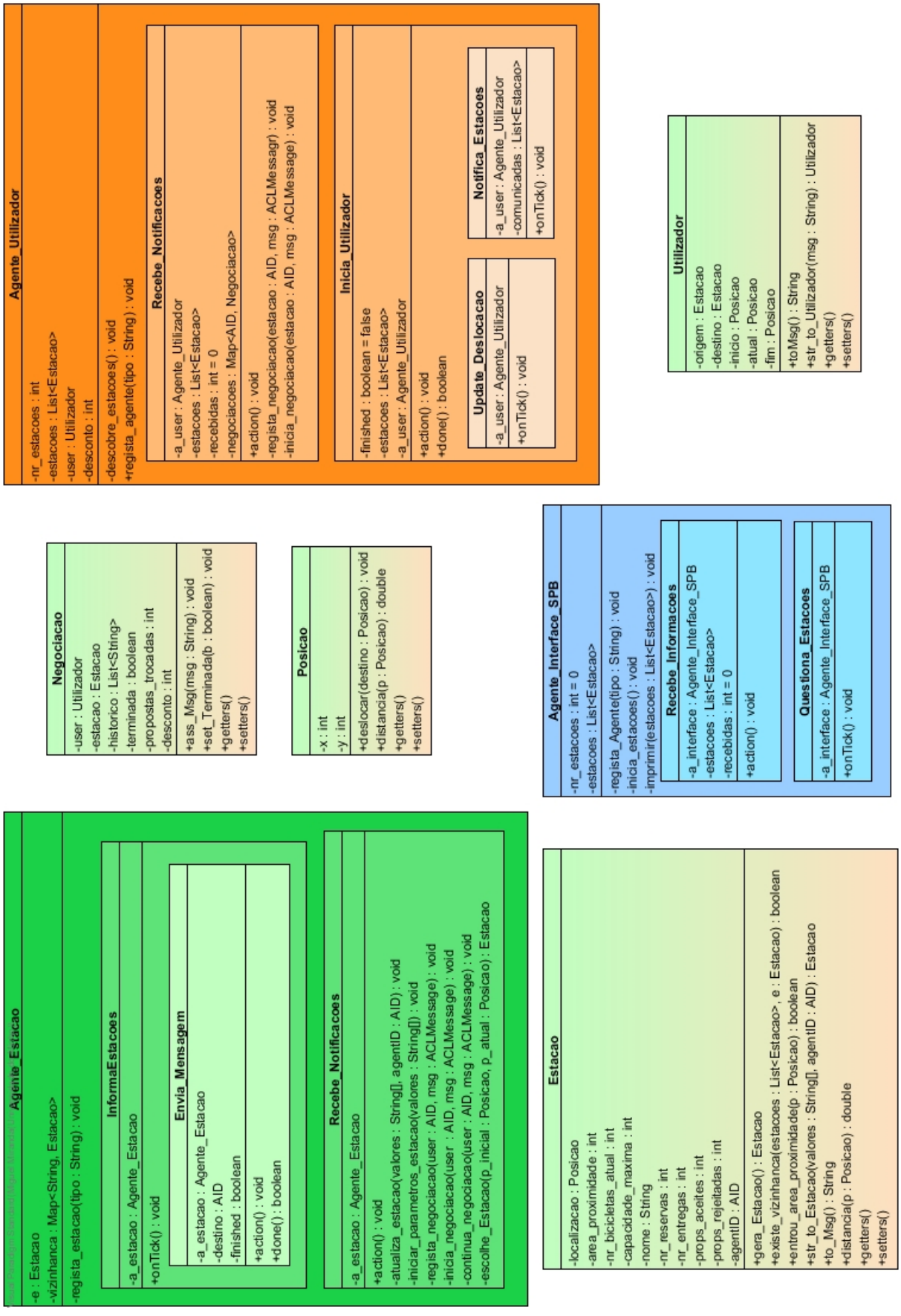

Figura 3.7: Diagrama classes da arquitetura implementada

3.4 Classificação da arquitetura

Sendo este contexto de aluguer de bicicletas um sistema bem definido a nível de intervenientes e funcionalidades necessárias, este será um sistema multi-agente fechado. Deste modo, a arquitetura apresentada será estática e os agentes interagem entre si de forma cooperativa, sendo que a sua linguagem e atos de comunicação serão concisos e descritos na secção seguinte.

Comparando especificamente a arquitetura apresentada com as arquiteturas *standard* de agentes, esta pode ser enquadrada numa arquitetura híbrida, por possuir características de arquiteturas deliberativas e arquiteturas reativas. Esta conjugação de diferentes formas de ação permite também tornar a modelação final mais funcional e adequada à forma de interação dos agentes no contexto real.

Existe assim um subsistema deliberativo/cognitivo, que pode ser visto no processo de negociação iniciado pelos agentes *Estacao* e, nos agentes *Utilizador*, que recebem ofertas de descontos em situações excecionais e decidem em torno disso executar ou não uma determinada ação, beneficiando ambos os agentes.

Uma visão mais reativa da forma como um agente perceciona e atua no ambiente, pode ser observada no comportamento condicional dos sensores. Estes são responsáveis por determinar a localização do utilizador e, caso se verifiquem as condições já abordadas, o sensor notifica uma dada estação que o utilizador se encontra na sua área de proximidade.

4. Especificação dos protocolos de comunicação

Considerando os três principais módulos identificados na arquitetura do sistema, será nesta secção descrita de que forma se estrutura e realiza a comunicação entre os mesmos.

A comunicação entre estes agentes baseia-se assim na troca de mensagens, que seguem a estrutura e especificação da norma *FIPA Agent comunication language (ACL)*. Dado o contexto do sistema de partilha de bicicletas, existem situações distintas que levam à troca de mensagens:

- O processo de **inicialização**, quando na fase de arranque do sistema de aluguer se atribuem valores aos atributos de cada estação;
- Os processos de **notificação** de estações, quando o sensor no equipamento de um determinado utilizador comunica com estações de aluguer;
- O processo de **informação** entre estações, onde os agentes *Estação* partilham mutuamente as suas principais características de utilização;
- Os processos de **negociação** entre agentes *Estação* e agentes *Utilizador*, nas situações em que uma estação procura gerir o seu nível de ocupação;
- O processo de **monitorização**, quando se procura solicitar as informações de cada estação e observar a evolução do sistema de aluguer no geral.

A ocorrência destes eventos origina assim a necessidade de associar diferentes tipos de *performatives* às mensagens, assim como a estruturação do formato do seu conteúdo, consoante o contexto sobre o qual a mesma é enviada. Estes aspetos serão assim especificados com mais detalhe nos seguintes tópicos, apresentando ainda um esquema em *Agent UML* para cada uma das situações de comunicação entre agentes.

4.1 Inicialização de estações

No processo de arranque do sistema, o agente *Interface* envia uma mensagem a cada uma das estação, onde no seu conteúdo estão os valores que a estação deve atribuir à suas variáveis, como por exemplo o número de bicicletas inicial, a capacidade máxima ou a ocupação ideal.

Estas mensagens são marcadas com a *performative* ***INFORM*** e o conteúdo inicia-se pela palavra "iniciar", seguido dos valores dos diferentes parâmetros.

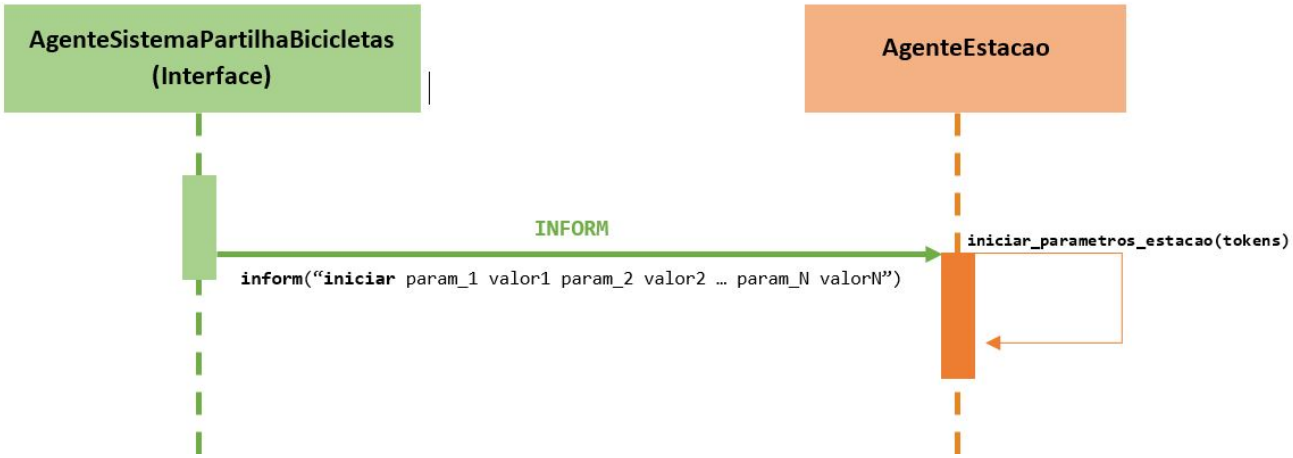

Figura 4.1: Protocolo comunicação das mensagens de inicialização

4.2 Notificação de estações

Como já referido, à medida que um utilizador se desloca ao longo do seu percurso, um sensor equipado na sua bicicleta irá regularmente determinar qual a sua posição atual e se o mesmo se encontra dentro da área de proximidade de alguma das estações. Caso esta condição se verifique, e o utilizador já tenha efetuado mais de 3/4 do seu percurso, este componente terá que ser capaz de comunicar com as estações sobre as quais o utilizador se encontra próximo.

Esta comunicação só é então realizada quando de facto está reunido um conjunto de condições que permitam à estação entrar em possível negociação com o cliente. As notificações podem ser vistas como um simples alerta para estação, estruturado numa mensagem que não precisa de confirmação ou resposta por parte da estação ao equipamento que a enviou.

Dentro do conteúdo da mensagem, iniciado pela palavra "cliente enter", será passado à estação as coordenadas da posição inicial, atual e destino utilizador, como forma da mesma conhecer o percurso do utilizador e de posteriormente determinar, a titulo de exemplo, qual a estação mais próxima a que o pode reencaminhar, em caso de sobrelotação.

Outras formas de notificar estações ocorrem quando:

- O utilizador inicia o seu percurso numa determinada estação e a informa que vai realizar o aluguer de um bicicleta, pegando assim numa das suas bicicletas disponíveis.

- O utilizador termina o seu percurso numa determinada estação, informando a mesma da entrega do equipamento alugado.

  Nesta situação, o utilizador informa ainda a estação sobre a sua posição inicial e final, como forma da mesma calcular aproximadamente a distancia percorrida pelo mesmo e lhe cobrar o valor do aluguer. Também nestas mensagens, é indicado à estação destino o desconto que o utilizador deve receber, caso alguma outra estação o tenha encaminhado para esta nova estação de entrega com a proposta de um desconto.

Estas notificações serão marcadas com a *performative* ***INFORM***, como forma de identificar ao agente recetor que o conteúdo da mensagem é uma proposição verdadeira e que o mesmo ainda não possui conhecimento sobre a informação que lhe está a ser enviada, sendo assim algo relevante para que o mesmo atualize o seu conhecimento.

O seguinte esquema procura descrever estas três situações de notificação por parte dos agentes *Utilizador* para os agentes *Estacao*.

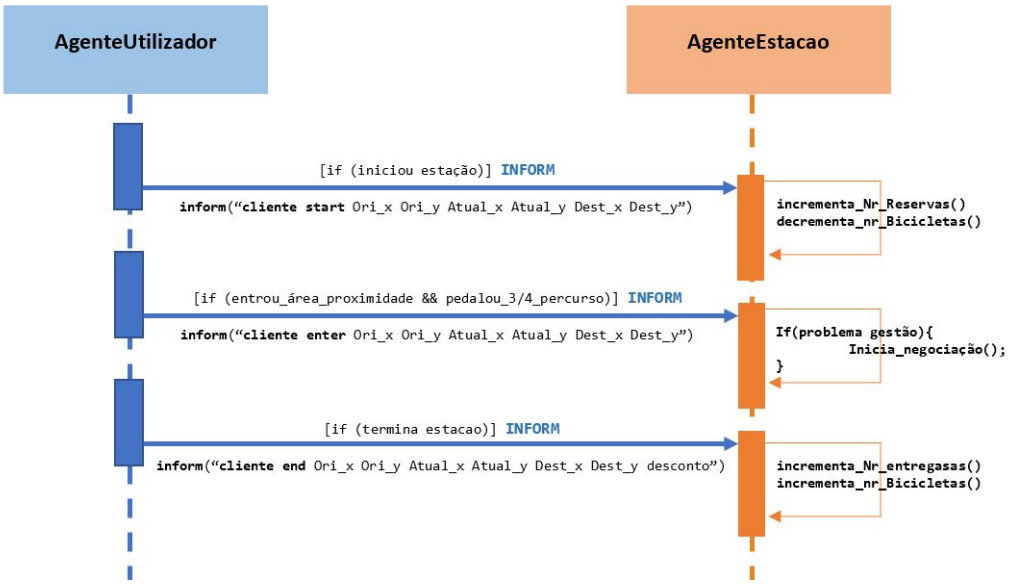

Figura 4.2: Protocolo comunicação das mensagens de notificação

4.3 Informação entre estações

Como forma de atualizar o conhecimento que uma estação tem sobre outras estações próximas de si, cada estação irá de forma recorrente informar outras estações sobre os seus parâmetros de utilização.

O envio e tratamento destas mensagens permite assim ter um conhecimento sobre as estações que evolui à medida que o próprio sistema é utilizado de forma paralela nos diferentes pontos de aluguer.

Pelo significado semântico do conteúdo da mensagem, este tipo de mensagens irá também utilizar a *performative* ***INFORM***, mas o conteúdo da mensagem é iniciado pela palavra "informar", seguido por um conjunto de parâmetros que representam os principais dados de utilização da estação.

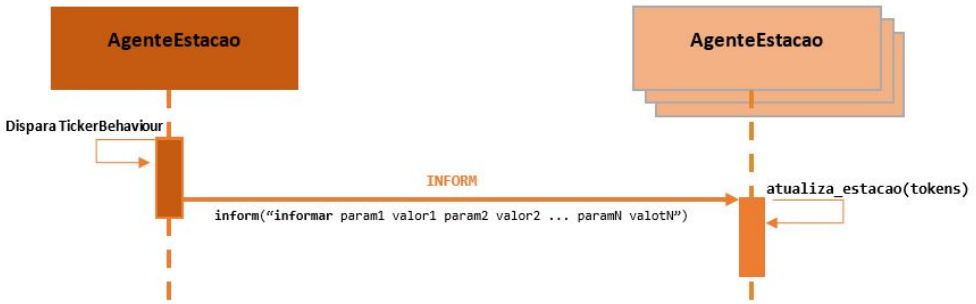

Figura 4.3: Protocolo comunicação das mensagens de informação

4.4 Negociação com utilizadores

Os cenários de negociação entre agentes surgem dentro do processo de gestão da ocupação de cada uma das estações de aluguer de bicicletas. A partir das notificações que uma estação vai recebendo à medida que novos utilizadores entram na sua área de proximidade, esta irá averiguar se lhe é interessante cativar algum utilizador para si ou pedir que o mesmo se dirija a outra estação para realizar a entrega.

No contexto apresentado, foram levantadas duas situações principais que levam ao processo de negociação:

- (1) A ocorrência de sobrelotação da estação, impossibilitando a entrega de mais bicicletas nesse local e, portanto, é pedido aos utilizadores que as entreguem numa outra estação próxima.

  De referir que neste reencaminhamento de utilizadores, a estação sobrelotada escolhe a nova estação tendo por critério aquela que é mais próxima do utilizador;

- (2) O cenário oposto, quando a estação se encontra com falta de bicicletas perante a procura média que a mesma serve. Nesta situação a estação procura cativar utilizadores a alterarem ligeiramente o seu destino e entregarem a sua bicicleta nesta nova estação, com défice de equipamentos para alugar.

Independentemente da razão pela qual surja a negociação, o utilizador é inicialmente confrontado com uma proposta. Esta proposta pode ser vista como um tuplo onde se indica ao utilizador a nova ação a realizar e uma motivação/oferta para que o mesmo aceite desempenha-la. A nova ação solicitada ao utilizador será a de entregar a bicicleta numa nova estação destino, seja pelos motivos (1) ou (2) citados acima.

A recompensa será esboçada na forma de um desconto no preço da sua viagem, sendo este tanto maior quando maior for a necessidade da estação em receber bicicletas na sua localização. Por omissão e questões de simplificação, os utilizadores pagam tanto mais quando maior for a sua distancia total percorrida e os descontos representam sempre valores inteiros, como por exemplo, 10% ou 30% de desconto.
.

Estas mensagens com a proposta inicial serão assim identificadas pela *performative* ***PROPOSE*** e são enviadas pelas estações de aluguer aos utilizadores alvo.

No lado dos utilizadores, perante a receção de uma proposta de uma dada estação, deve ser avaliada a eventual vantagem do desconto oferecido e a localização da nova estação, podendo o agente responder da seguinte forma:

- Confirma que realiza a alteração da estação destino, aceitando o valor de desconto oferecido.

  Esta situação é identificada pela *performative* ***AGREE*** e o conteúdo da mensagem é o mesmo da mensagem com a proposta recebida;

- Realiza uma contra oferta, onde se demonstra o interesse em aceitar a nova ação, mas apenas perante um desconto mais elevado.

  Esta situação é marcada com a *performative* ***CALL FOR PROPOSAL*** e o conteúdo da mensagem apresenta a mesma estrutura da mensagem recebida, mas no campo referente ao desconto coloca um valor maior;

- Rejeita a proposta, sem procurar negociar outros valores de desconto. Esta situação pode ocorrer porque de facto se rejeita a proposta ou porque já se trocaram mais de 5 contra propostas com a estação e o utilizador desiste de tentar negociar.

  Esta situação utiliza a *performative* ***REJECT_PROPOSAL*** e por uma questão de significado do conteúdo, este contem apenas a palavra "rejeita".

Novamente na perspetiva do agente *Estacao*, se perante a realização de uma proposta a resposta recebida for:

- Marcada com a *performative* ***AGREE***, reconhece o término da negociação com aquele o *Utilizador* e incrementa o contador de propostas aceites;

- Marcada com a *performative* ***REJECT_PROPOSAL***, reconhece o término da negociação com o agente *Utilizador* e incrementa o contador de propostas rejeitadas;

- Marcada com a *performative* ***CFP***, reconhece que a negociação teve uma nova iteração e contra proposta por parte do agente *Utilizador*. Neste cenário uma estação toma uma das três possíveis decisões já descritas acima, quando o agente *Utilizador* recebia a mensagem inicial com a *performative* ***PROPOSE***, ou seja, pode aceitar, rejeitar ou realizar uma outra contra proposta.

O seguinte esquema, com notação UML, procura descrever esta interação de negociação entre os agentes estação e agentes utilizador. No esquema é tida em conta a eventual situação de uma proposta se desenrolar noutras contra propostas sucessivas, até que um dos agentes aceite ou desista de negociar.

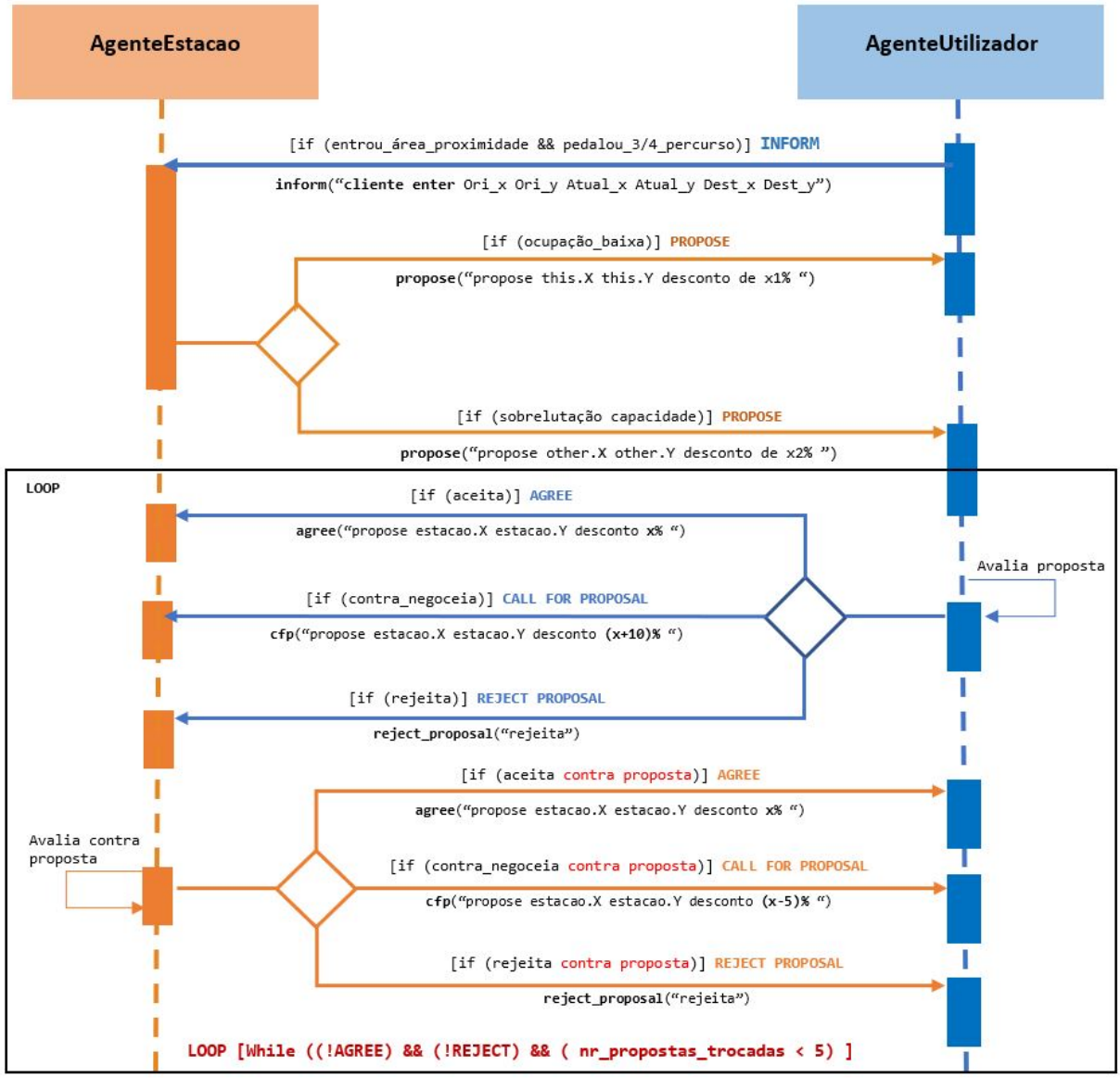

Figura 4.4: Protocolos comunicação das mensagens de negociação

4.5 Monitorização sistema aluguer

Considerando ainda as funcionalidades associadas à visualização do estado de ocupação e utilização de cada uma das estações, existe ainda um outro tipo distinto de mensagens, referentes aos pedidos que o agente *Interface_SPB* realiza a cada estação.

Estas mensagens corporizam assim a requisição de informações às estações de aluguer, e portanto, serão denotadas com a *performative* ***REQUEST*** e o seu conteúdo contém apenas a palavra "informar". Estas características numa mensagem são entendidas pelos agentes *Estacao* que as recebem como uma solicitação das suas informações de utilização, que o agente estação vai mantendo e atualizando ao longo do tempo.

Desta forma, cada uma das estações responde ao agente *Interface* informando no conteúdo da mensagem de resposta os valores das suas variáveis mais relevantes no contexto.

Este tipo de mensagens são unicamente enviadas pelo agente *Interface* SPB aos agentes *Estação*, sendo que com as posteriores respostas recebidas, a interface processa as informações para as imprimir no terminal.

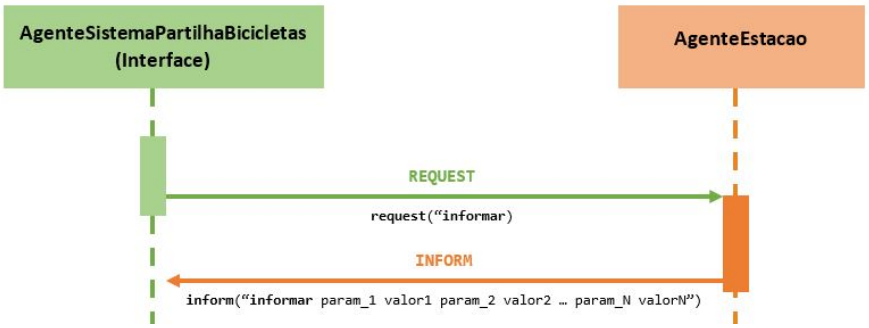

Figura 4.5: Protocolo comunicação das mensagens de monitorização

5. Implementação

Neste capítulo será abordado, de forma detalhada, a metodologia usada para implementar o funcionamento do sistema de aluguer de bicicletas, através do desenvolvimento dos agentes *Interface*, *Estação* e *Utilizador*.

Tal como já referido, a implementação das funcionalidades da arquitetura foram desenvolvidas em *Java*, com recurso às funcionalidades de agentes disponibilizadas pelo sistema *Jade*. Para uma melhor compreensão dos aspetos técnicos aqui referidos, aconselha-se a revisão do esquema da arquitetura implementada (secção 3), tendo esta sido revista e aumentada ao longo do desenvolvimento da fase de implementação.

5.1 Módulo *SistemaPartilhaBicicletas - Interface*

Na modelação inicial da arquitetura do sistema de partilha de bicicletas, foi definido que as funcionalidades de criação e monitorização do sistema ficariam associadas ao sub-sistema *SistemaPartilhaBicicletas*, no qual se vem a inserir o agente *Interface*.

Para ser possível criar os diferentes agentes neste sub-sistema, será aqui o ponto de arranque do sistema de aluguer, através do método *main* e do uso da classe *MainContainer*, para realizar a integração com o ambiente de agentes *Jade*.

Este arranque inicial dos agentes do sistema, é feito da seguinte forma:

1. É solicitada a introdução do número de estações que se pretendem criar no sistema e assim que um valor inteiro seja introduzido, são então criados **N** agentes *Estação*.

   Esta inicialização dos agentes *Estação* arranca apenas com os agentes que vão representar uma estação de aluguer, mas só mais tarde lhes será atribuída uma localização e valores aos seus atributos, através de uma comunicação inicial com o agente *Interface*;

2. Criados os agentes *Estação*, é iniciado o agente *Interface*;

3. Por fim, é solicitado que se introduza o número de utilizadores que se pretende que sejam criados para simular a utilização do sistema de aluguer. Introduzido um valor inteiro, serão gradualmente criados os **N** agentes *Utilizador*, com intervalos aleatórios entre [0, 2] segundos.

5.1.1 Agente *Interface*

Assim que o agente *Interface* é inicializado, o mesmo regista-se no *Directory Facilitator* associado à função de *Interface*. A sua primeira tarefa é realizar o processo de atribuição de valores aos agentes *Estação* já criados, mas sem quaisquer valores nas suas variáveis.

Para isso, o agente começa por utilizar as informações contidas no *Directory Facilitator* para descobrir quem são os agentes existentes, registados com a funcionalidade de *Estação*. Para cada um dos agentes *Estação* encontrados, é gerada uma variável do tipo *Estacao*, através do método *gera_estacao*. Este método cria de forma aleatória e dentro do contexto do sistema a desenvolver, valores para cada um dos atributos de uma estação de aluguer. A nível da localização das estações, assumiu-se um mapa com dimensão inteiras 100x100.

Com estes valores iniciais, os mesmos são comunicados à respetiva estação que os irá utilizar, através de uma mensagem com a estrutura descrita na secção 4.1. Tendo gerado e atribuído valores iniciais a cada um dos agentes *Estacao*, o agente *Interface* termina assim o seu estado inicial de criação do sistema e passa a desenrolar a tarefa de monitorização.

Para isso, são criados dois comportamentos:

- Um *TickerBehaviour*, implementado na classe *Questiona_Estacoes*, responsável por a cada 10 segundos enviar uma mensagem a cada um dos agentes *Estacao* registados no *Directory Facilitator*.

  Estas mensagens servem para requerer a cada estação as suas informações e dados atuais e estão apresentadas com detalhe na secção secção 4.5.

  Como este processo de envio de mensagens a cada estação pode ser feito de forma paralela, sempre que disparar o *TickerBehaviour* é utilizado um *ParallelBehaviour* para comunicar com todas as estações de forma simultânea.

- Um *CyclicBehaviour*, implementado na classe *Recebe_Informações*, responsável por realizar o tratamento das mensagens recebidas no agente *Interface* e com remetente nos agentes *Estacao*.

  No caso de ter disparado o *TickerBehaviour* descrito no ponto anterior, quando forem recebidas todas as mensagens de resposta por parte dos agentes *Estacao*, as informações são recolhidas dessas mensagens e disponibilizadas no terminal na forma de uma tabela, onde são listados alguns dos principais campos de utilização de uma estação.

  O conteúdo e informações impressas nesta tabela serão apresentados com detalhe e exemplos na secção de demonstração do sistema (secção 6).

A seguinte imagem procura descrever de forma geral o diagrama de classes deste módulo do sistema.

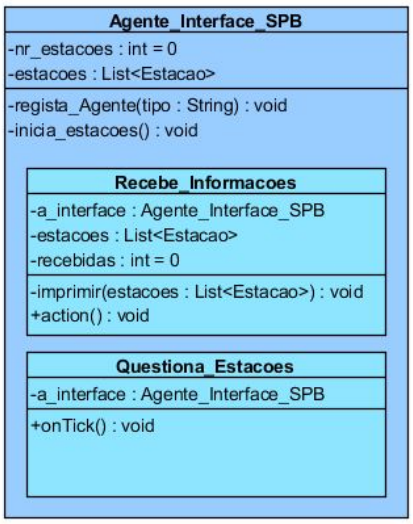

Figura 5.1: Diagrama classes Agente *Interface*

5.2 Módulo *Agente Estação*

O módulo *AgenteEstação* corporiza no sistema a representação de um posto de aluguer de bicicletas. Implementado também sobre a forma de agentes, estes apresentam como objetivos a gestão de uma estação de aluguer, focando-se no controlo do número de equipamentos disponíveis para alugar.

Com esta gestão, os agentes *Estacao* procuram evitar cenários de sobrelotação de entregas ou a falta de equipamentos disponíveis para satisfazer a chegada de utilizadores. A nível de implementação, os agentes com estas funções apresentam dois comportamentos:

- Um *TickerBehaviour*, implementado na classe *Informa_Estacoes*, que irá a cada 3 segundos enviar uma mensagem a todos os agentes *estação*, de forma a que cada estação conheça o estado atual das estações à sua volta.

  Como este processo de envio de mensagens a cada estação pode ser feito de forma paralela, sempre que disparar o *TickerBehaviour* é utilizado um *ParallelBehaviour* para comunicar com todas as estações de forma simultânea.

  Como referido no processo de modelação inicial, esta comunicação permite que cada estação tenha uma visão presente e geral do sistema onde se inclui, podendo assim, em cenários de sobrelotação, escolher qual a estação mais próxima e com determinadas características que a tornem favorável para receber os clientes que a estação não pode receber.

  De outra forma, a escolha de uma estação para onde encaminhar um utilizador em casos de sobrelotação seria aleatória, podendo o utilizador ser encaminhado para uma estação também ela sobrelotada. Tais cenários levariam

ao descontentamento do utilizador, que desvalorizaria o serviço do sistema de aluguer.

A estrutura destas mensagens pode ser revisto na secção 4.3.

- Um *CyclicBehaviour*, implementado na classe *Recebe_Notificacoes*, responsável por realizar o tratamento das mensagens recebidas por parte dos agentes *Utilizador*, *Interface* e outros agentes *Estacao*.

  A nível do significado das mensagens recebidas por parte dos agentes *Estacao*, as mesmas podem ser:

  - Marcadas com a *performative* ***INFORM*** e o seu conteúdo iniciado por:
    - *"iniciar"*, quando na fase inicial do sistema, o agente *Interface* envia uma mensagem com os valores a atribuir aos atributos do agente *Estacao*.
      Nesta situação, através do parsing do conteúdo da mensagem, é utilizado o método *iniciar_parametros_estacao* para atribuir os valores da mensagem aos valores dos atributos do agente *Estacao*;
      A estrutura destas mensagens é apresentada na secção 4.1.
    - *"informar"*, no caso das mensagens trocadas intra estações, para que cada um conheça o estado atual das restantes.
      Nestas situações, novamente com o parsing do conteúdo da mensagem, é utilizado o método *atualiza_estacao* para através dos valores retirados do conteúdo da mensagem atualizar o conhecimento sobre a estação que remeteu a mensagem;
      A estrutura destas mensagens é apresentada na secção 4.3.
    - *"cliente start"*, quando um utilizador inicia o seu percurso numa determinada estação de aluguer e avisa-a desse procedimento. Nesta situação o número de clientes é incrementado e o número de equipamentos disponíveis para aluguer é decrementado;
    - *"cliente end"*, quando um utilizador termina o seu percurso na estação final onde entrega a bicicleta alugada. Nesta situação, o número de entregas e de bicicletas disponíveis na estação são incrementados em uma unidade. É ainda aumentando o dinheiro recolhido pela estação, conforme o preço que o cliente paga pelo aluguer;
    - *"cliente enter"*, quando um utilizador determina que entrou dentro da área de proximidade de uma estação e notifica-a sobre a sua presença.
      Será com estas mensagens que a estação decide, conforme o seu estado de ocupação, se inicia ou não um processo de negociação com o utilizador;
      A estrutura destes três últimos formatos de mensagem são apresentados na secção 4.2.
  - Marcadas com a *performative* ***AGREE, REJECT_PROPOSAL, CFP*** em situações de resposta a um processo de negociação previamente iniciado. A estrutura destas mensagens é apresentada na secção 4.4.

A implementação do processo de negociação e tratamento das mensagens que o corporizam, serão tratadas com mais detalhe no capitulo seguinte;

– Marcadas com a *performative* ***REQUEST*** e o seu conteúdo iniciado pela palavra "informar", quando se trata do agente *Interface* a solicitar as informações de utilização da estação. A estrutura destas mensagens é apresentada na secção 4.5.

A seguinte imagem procura descrever de forma simplificada o diagrama de classes do sub-sistema representado num agente *Estacao*

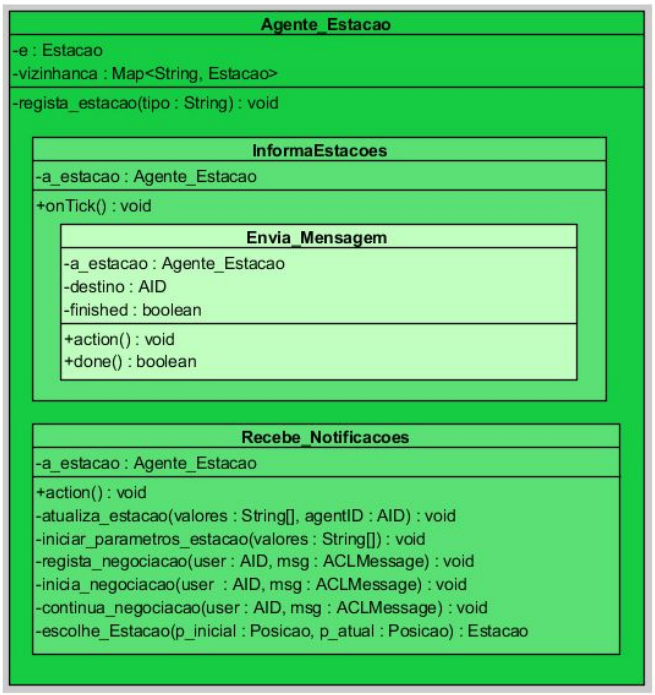

Figura 5.2: Diagrama classes Agente *Estação*

5.3 Módulo *AgenteUtilizador*

No subsistema *AgenteUtilizador* foram implementadas as funcionalidades que permitissem simular a afluência de utilizadores ao sistema, nomeadamente:

- O aluguer de equipamentos e deslocação do utilizador ao longo do seu percurso;
- A aceitação, negação ou negociação das possíveis propostas enviadas pelos agentes *Estacao* aos utilizadores na sua área de proximidade, em cenários de sobrelotação ou necessidade de repor o stock de bicicletas disponíveis.

Além de representar o ator utilizador do sistema, num contexto mais especifico este módulo implementa também o funcionamento dos sensores que acompanham os equipamentos alugados. Estes sensores têm como tarefas determinar a localização atual do utilizador ao longo do seu percurso e, caso o mesmo intercete a área de proximidade de uma estação, o controlador do sensor *GPS* deve notificar a estação sobre a qual se entrou na área de proximidade.

Assim que o agente que representa o utilizador é iniciado, o mesmo regista-se no *Directory Facilitator* associado à função de utilizador. Como se procura criar alguma aleatoriedade na escolha das estações origem e destino do utilizador, antes da criação dos comportamentos que efetivamente modulam o comportamento de um utilizador do sistema, são feitos alguns procedimentos iniciais, nomeadamente:

- A criação de um *CyclicBehaviour*, implementado na classe *Recebe_Notificacaoes*, responsável por realizar o tratamento das mensagens recebidas no agente *Utilizador*;

  As mensagens recebidas, são sempre enviadas por parte dos agentes *Estacao* e podem ser marcadas com as *performatives* ***AGREE***, ***PROPOSE***, ***REJECT_PROPOSAL*** *ou* ***CALL FOR PROPOSAL***.

  Estando estas mensagens relacionadas com o processo de negociação entre agentes, o processamento e tratamento das mesmas será abordado com mais detalhe na secção seguinte;

- A chamada do método *descobre_Estacoes*, que recolhendo no *Directory Facilitator* a referência aos agentes que desempenham a função de *Estacao*, envia a cada um deles uma mensagem a solicitar as suas informações de utilizador.

  Novamente, pelo facto desta tarefa poder ser realizada de forma simultânea, o envio das mensagens é processado dentro de um comportamento paralelo, que realiza esta tarefa de enviar uma mensagem a cada uma das estações descobertas.

  Esta mensagem, marcada com a *performative* ***REQUEST*** e cujo conteúdo inicia pela palavra *"informar"*, é apresentada com mais detalhe na secção anterior, relativa aos agentes *Estação*.

Depois da iteração do método *descobre_Estacoes*, quando no *CyclicBehaviour* referido se tiver recebido todas as mensagens de resposta às mensagens enviadas, o agente *Utilizador* conhece então todas as estações existentes no sistema e pode delas escolher duas, como origem e destino, respetivamente. Este processo é feito com recurso a um *SimpleBahaviour* implementado na classe *Iniciar_Utilizador*.

De uma forma geral, este comportamento desempenha aquilo que podia ser feito no método *setup* do agente *utilizador*, caso este conhecesse as estações de aluguer mal é iniciado. Como tal não acontece, é preciso primeiro descobrir as estações antes de determinar uma origem e destino, e arrancar com os comportamentos que simulam a atuação do utilizador no sistema de aluguer.

Este *SimleBehaviour* irá assim desempenhar as seguintes ações:

- Das estações encontradas, serão aleatoriamente selecionadas duas distintas, que repectivamente serão as estações destino e origem do agente *Utilizador*;

  Apesar de momentos antes o agente *Utilizador* receber mensagens contendo as informações mais recentes das estações existentes, este escolhe uma estação de origem aleatória, sem analisar se a mesma tem um stock de bicicletas disponível para aluguer. Se não houver um número positivo de equipamentos para alugar, o cliente termina a sua iteração no sistema, sem realizar reservas.

- À estação origem selecionada, é enviada uma mensagem para a informar que um novo cliente requisitou nela uma bicicleta. Este processo é feito enviando uma mensagem estampada com a *performative* ***INFORM*** e com o conteúdo da mensagem contendo *"cliente start"*, tal como já especificado na secção 4.2.

- Estando a configuração inicial do utilizador realizada, iniciam-se dois comportamentos para simular o seu papel de ação no sistema:

  – Um *TickerBehaviour* implementado na classe *Update_Deslocacao*, responsável por a cada meio segundo realizar a deslocação da posição atual do utilizador, em direção à estação destino do mesmo.

    Quando neste comportamento se verificar que a deslocação do utilizador, coloca a sua posição atual no mesmo local da estação onde irá entregar a bicicleta, então o sensor do equipamento envia uma mensagem à estação para a informar que o utilizador chegou ao fim da sua viagem e entregou o seu equipamento na estação.

    Se considerarmos que as estações funcionam de forma autónoma, sem a presença física de funcionários, estas mensagens permitem ao sistema de gestão de cada estação de aluguer determinar a chegada dos clientes que nelas devolvem bicicletas.

    Assim que o utilizador entrega a bicicleta numa estação, é assumido que o pagamento é feito de forma automática e virtual, por exemplo através de uma aplicação do sistema de aluguer. Desta forma o utilizador termina aqui o seu papel no sistema, levando ao término do agente que o representa.

  – Outro *TickerBehaviour*, implementado na classe *Notifica_Estações*, responsável por determinar, a cada 2 segundos, se o utilizador entrou na área de proximidade de alguma estação. Caso tal condição se verifique, o mesmo notifica a estação da sua presença no seu raio de proximidade.

A seguinte figura representa assim o diagrama de classes de um agente do tipo *Utilizador*.

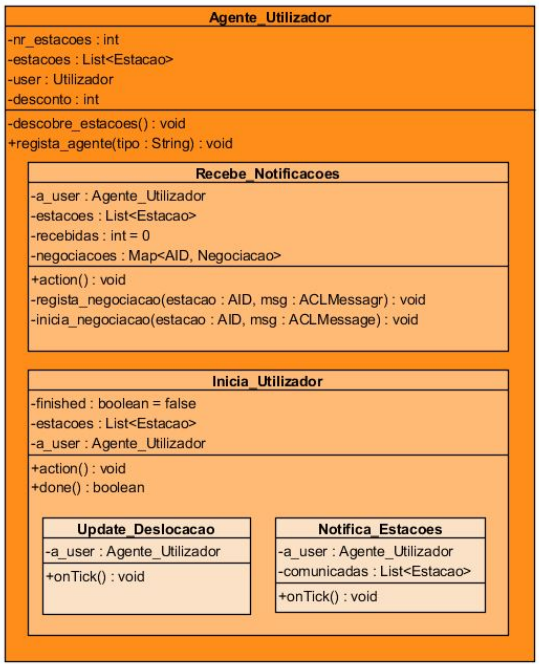

Figura 5.3: Diagrama classes Agente *Estação*

5.4 Processo de negociação

Apesar do processo de negociação já ter sido falado com algum detalhe na secção alusiva aos protocolos de comunicação utilizados nesse contexto (secção 4.4), o tema é agora apresentado na perspetiva de implementação desta funcionalidade entre os agentes.

Alguns dos aspetos aqui focados, relativos ao tratamento das mensagens de negociação trocadas, serão comuns a ambos os agentes *Estacao* e *Utilizador*, justificando por esse motivo a criação desta secção.

Na situação em que os agentes entrem em troca de contra propostas sucessivas, existe um número máximo de 5 propostas trocadas até que um dos lados desista da negociação, por não entendimento e não convergência de ambas as partes.

5.4.1 Negociação na perspetiva de Utilizador

Tal como já referido, no lado do agente *Utilizador* teremos um *CyclicBehaviour* implementado na classe *Recebe_Notificaoes*, responsável por realizar o tratamento das mensagens enviadas ao agente.

Quando uma determinada mensagem recebida for marcada com a *performative PROPOSE*, *AGREE*, *CFP* ou *REJECT_PROPOSAL* estamos perante um cenário de inicio ou continuação de uma negociação com uma das estações. Nesta primeira abordagem, independentemente do tipo de *performative*, é utilizado o método designado por *regista_notificacao* para realizar efetivamente o tratamento do conteúdo da mensagem.

Neste método *regista_notificacao* existem três condições que são testadas:

- Se a *performative* da mensagem recebida for *AGREE*, então trata-se de uma situação em que a estação aceitou a contra proposta feita pelo agente.

  Relembrando que o agente tem uma variável onde guarda o histórico de negociações realizado com cada estação que o contactou com uma proposta (secção 3.2.2), o utilizador tem assim acesso à última proposta trocada com a estação que aceitou a sua proposta. Acendendo ao conteúdo desta ultima proposta, o agente *Utilizador* altera a sua estação destino e leva uma *tag* com o valor do desconto que lhe foi oferecido.

  Nas situações em que uma estação cativa um agente, a própria estação guarda também o histórico de negociações e desconto atribuído ao agente, sabendo que desconto lhe aplicar quando o mesmo chegar até si.

  Contudo, nas situações em que o agente é encaminhado para uma nova estação, é o utilizador que informa esta nova estação que outra estação lhe ofereceu um desconto.

- Se a *performative* da mensagem for *REJECT_PROPOSAL*, então o agente reconhece que a estação não aceitou a sua contra proposta. Nestas situações o utilizador não realiza alterações ao seu percurso e marca a negociação com esta estação como terminada.

- Caso a *performative* seja *PROPOSE* ou *call for propose CFP*, é utilizado o método *inicia_negociacao* para determinar uma resposta à proposta recebida.

O método *inicia_Negociacao* é assim utilizado quer para dar uma resposta a uma proposta inicial de uma estação, como para dar continuidade à negociação em casos de de troca de contra propostas. Este método procura simular a tomada de decisão do agente, sendo que:

- Com uma probabilidade de 15% o utilizador rejeita a proposta/contra proposta recebida, respondendo com um *REJECT_PROPOSAL*;

- Com uma probabilidade de 25% o utilizado realizar uma contra proposta, solicitando um desconto de mais 10% sobre o valor indicado na mensagem recebida.

  Nestas situações o agente responde ao agente *Estacao* com uma *CALL FOR PROPOSAL*;

- Com a restante probabilidade de 60%, o utilizador aceita a proposta ou contra proposta recebida, realizando os mesmo passos já enunciados quando recebe uma mensagem com a performative *AGREE*.

Estas probabilidades procuram modelar um comportamento dos agentes *Utilizador* potencialmente colaborativo com as estação, mas que em algumas situações também realizam contra propostas e rejeições.

Independente do tipo de mensagem de negociação recebida, no final de uma iteração a mesma é sempre adicionada à variável do tipo *Negociacao* que guarda a evolução do processo de negociação entre o agente *Utilizador* com uma determinada estação.

5.4.2 Negociação na perspetiva de Estação

De forma muito semelhante aos utilizadores, o agente *Estacao* terá também um *CyclicBehaviour* implementado na classe designada por *Recebe_Notificacoes*, responsável por realizar o tratamento das mensagens que chegam ao agente.

Quando uma determinada mensagem recebida for marcada com a *performative AGREE*, *CFP* ou *REJECT_PROPOSAL*, o agente *Estacao* está perante um cenário de continuação de uma negociação previamente feita a um agente *Utilizador*.

Nesta situações, é também utilizado um método designado por *regista_notificacao* para realizar efetivamente o tratamento do conteúdo da mensagem.

Neste método *regista_notificacao*, na perspetiva de um agente *Estacao*, existem três condições que são testadas:

- Se a *performative* da mensagem recebida for *AGREE*, então trata-se de uma situação em que o utilizador aceitou a última proposta que lhe foi realizada.

  A estação pode assim incrementar o seu número de propostas aceites e marcar a negociação com o utilizador como terminada.

- Se a *performative* da mensagem for *REJECT_PROPOSAL*, então o agente reconhece que o utilizador não aceitou a sua contra proposta. Neste caso o número de propostas rejeitadas é incrementado numa unidade e a negociação é também marcada como terminada.

- Caso a *performative* seja *CFP*, é utilizado o método *continua_negociacao* para determinar uma resposta à contra proposta recebida.

  Este método é a nível de funcionalidades e tomada de decisão equivalente ao método *inicia_negocicao* implementado no lado do utilizador e já apresentado.

6. Demonstração resultados

Nesta secção serão apresentados algumas exemplos relevantes do funcionamento do sistema, procurando assim validar a forma como os agentes de facto interagem e colaboram dentro do contexto do sistema.

6.1 Arranque do sistema

Como já referido, o arranque do sistema baseia-se na introdução do número de estações de aluguer que se querem criar e, posteriormente, do número máximo de utilizadores que vão passar pelo sistema. Ao passo que as estações são iniciadas todas em simultâneo numa iteração inicial, os utilizadores são criados ao longo do tempo, simulando assim o fluxo de chegada de clientes às diversas estações.

Estes últimos entram no sistema em conjuntos de 6 utilizadores paralelos, gerados em intervalos de tempo que variam entre 0 a 2 segundos, até se atingir o número máximo de utilizadores introduzido.

Apesar de não terem sido testados limites, das simulações realizadas, pode ser introduzido um qualquer número de estações assim como de utilizadores.

A seguinte imagem procura representar esta configuração inicial.

```
Agent container MainContainer@192.168.1.122 is ready.
---------------------------------------------
[2017-11-23 21:32:42] Quantas estações de aluguer deseja iniciar: 9
[2017-11-23 21:32:44] Agente_Estacao_0 iniciou.
[2017-11-23 21:32:44] Agente_Estacao_1 iniciou.
[2017-11-23 21:32:44] Agente_Estacao_2 iniciou.
[2017-11-23 21:32:44] Agente_Estacao_3 iniciou.
[2017-11-23 21:32:44] Agente_Estacao_4 iniciou.
[2017-11-23 21:32:44] Agente_Estacao_5 iniciou.
[2017-11-23 21:32:44] Agente_Estacao_6 iniciou.
[2017-11-23 21:32:44] Agente_Estacao_7 iniciou.
[2017-11-23 21:32:44] Agente_Estacao_8 iniciou.
Agente_Interface: iniciado.
[2017-11-23 21:32:45] Quantos utilizadores deseja criar:
500
[2017-11-23 21:32:46] Agente_Utilizador_0 iniciou.
[2017-11-23 21:32:46] Agente_Utilizador_3 iniciou.
[2017-11-23 21:32:46] Agente_Utilizador_5 iniciou.
[2017-11-23 21:32:46] Agente_Utilizador_1 iniciou.
[2017-11-23 21:32:46] Agente_Utilizador_2 iniciou.
[2017-11-23 21:32:46] Agente_Utilizador_4 iniciou.
```

Figura 6.1: Exemplo da configuração de arranque do sistema

6.2 Formato dos resultados apresentados

Tal como referido, uma das funcionalidades do sistema de aluguer recai sobre a possibilidade de monitorizar a evolução do mesmo, à medida que é utilizado por utilizadores. Este processo é assim desempenhado pelo agente *Interface*, que a intervalos de tempo regulares, solicita os dados mais recentes a cada uma das estações e lista as informações recebidas na forma de uma tabela.

A seguinte imagem procura assim representar a forma e conteúdo dessa tabela, numa simulação do sistema com 9 estações de aluguer e um fluxo continuo de 500 utilizadores.

Nome	Px	Py	Raio	C atual	C max	Reservas	Entregas	Oc ideal	Oc atual	Prop A	Prop R	% Aceites	Prejuizos	Lucro
Estacao_2	17	39	29	1	17	31	17	0.75	0.06	18	8	0.67	0	506.57
Estacao_1	93	9	35	37	38	54	64	0.83	0.97	61	18	0.76	3	1962.04
Estacao_5	46	29	23	20	24	57	57	0.76	0.83	103	23	0.81	0	1226.56
Estacao_3	67	9	32	23	24	50	57	0.83	0.96	109	30	0.78	4	1521.34
Estacao_6	87	73	25	7	38	47	24	0.79	0.18	29	9	0.74	0	845.29
Estacao_0	59	35	20	20	40	63	48	0.82	0.5	95	19	0.83	0	890.41
Estacao_4	35	83	33	29	30	54	67	0.79	0.97	79	16	0.82	9	1961.18
Estacao_7	63	66	20	17	18	56	68	0.78	0.94	84	15	0.84	10	1420.37
Estacao_8	48	60	24	16	17	55	65	0.79	0.94	134	24	0.84	9	1400.92

Figura 6.2: Exemplo tabela de resultados após simulação do sistema de aluguer

Relativamente às informações da tabela, as linhas são referentes às estações criadas no sistema e as colunas indicam os principais atributos que caracterizam o estado atual dessa estação. O significado destes atributos é o seguinte:

- **Nome**: Nome atribuído à estação de aluguer;
- **Px** e **Py**: Valores das coordenadas *x* e *y* que representam a localização da estação no referencial cartesiano;
- **Raio**: Valor atribuído ao raio usado para determinar a área de proximidade da estação;
- **C Atual**: Indica a capacidade de bicicletas atual da estação, ou seja, o número de equipamentos disponíveis para aluguer naquele instante;
- **C Max**: Capacidade de armazenamento máxima da estação;
- **Reservas**: Número de reservas realizadas na estação, ou seja, o número de clientes que alugaram uma bicicleta naquela estação;
- **Entregas**: Número de entregas realizadas na estação, seja por motivos de deslocações ou porque de facto a estação era o destino do utilizador;
- **Oc ideal**: Valor percentual que representa a ocupação ideal da estação, com base no fluxo de clientes que a mesma recebe. A sua ocupação atual deve ser próxima deste valor, para garantir que não ocorrem perda de utilizadores por falta de equipamentos disponíveis;
- **Oc Atual**: Ocupação atual da estação, calculada através do número de bicicletas atual e da capacidade máxima da mesma;
- **Prop A**: Indica o número de propostas de negociação iniciadas pela estação e aceites por parte do utilizadores que as receberam ou, contra propostas realizadas pelos utilizadores e aceites pela estação;
- **Prop R**: Indica o número de propostas realizadas a utilizadores e rejeitadas

por parte dos mesmos ou rejeições realizadas por parte da própria estação quando recebe contra propostas de utilizadores;
- **% Aceites**: Percentagem que através das duas últimas colunas, indica a percentagem de propostas aceites.
- **Prejuizos**: Representa a contagem de situações em que a estação, quando recebe um determinado cliente, fica num estado de sobrelotação, tendo que ser um dos seus funcionários a deslocar a bicicleta devolvida para outra estação com capacidade de armazenamento disponível;

Sobre os dados referidos na tabela, convém referir que:

- O somatório da coluna *Reservas* não tem necessariamente que ser igual ao número de clientes criados no sistema. Isto deve-se às situações em que o utilizador inicia o seu percurso numa estação sem stock de bicicletas disponível e, portanto, não é feita nenhuma reserva na mesma e o utilizador termina a sua atuação no sistema de aluguer;

- O somatório da coluna *Entregas*, será obrigatoriamente igual ao valor do somatório da coluna *Reservas*, pois não são consideradas no sistema situações de extravio de equipamentos ou perdas de clientes a meio do seu percurso até uma estação destino.

6.2.1 Negociação em caso de sobrelotação

Nas situações de sobrelotação, as estações de aluguer procuram deslocar os utilizadores que tinham inicialmente como destino esta estação, para outras estações que tenham capacidade para os receber.

Os seguintes exemplos procuram descrever negociações que surgiram neste contexto.

- Situação em que se realiza uma proposta de deslocação e o utilizador aceita realizar essa alteração no seu destino. De salientar que esta proposta é realizada pela estação que inicialmente era o destino do utilizador, mas que quando este entra na sua área de proximidade e a mesma verifica que está em estado de sobrelotação, procura encaminhar o utilizador para outra estação.

  Neste caso especifico, alem do utilizador aceitar a proposta da estação_3, é ainda posteriormente confrontado com uma proposta de atração da estação_5, que tambem aceita. Neste caso é possivel ver que o utilizador termina de facto na estação_5, mas os descontos nao são acumuláveis, ficando apenas o último valor trocado nas mensagens de negociação.

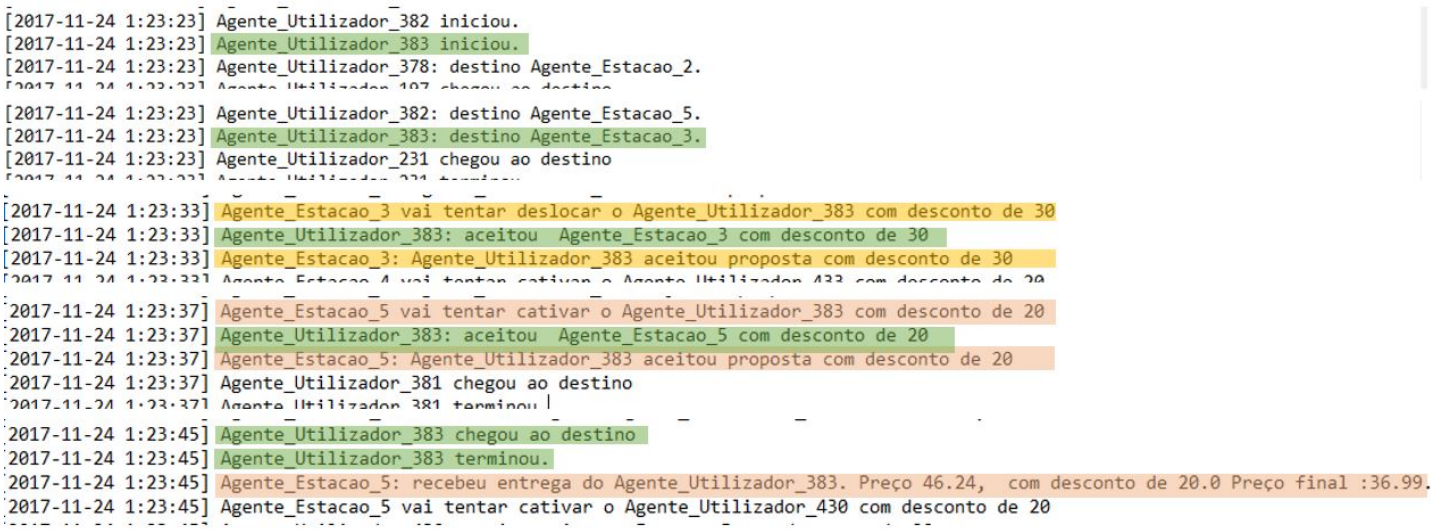

Figura 6.3: Exemplo de uma proposta de deslocação aceite por um utilizador

- Cenário oposto, em que se realiza uma proposta de deslocação e o cliente a recusa.

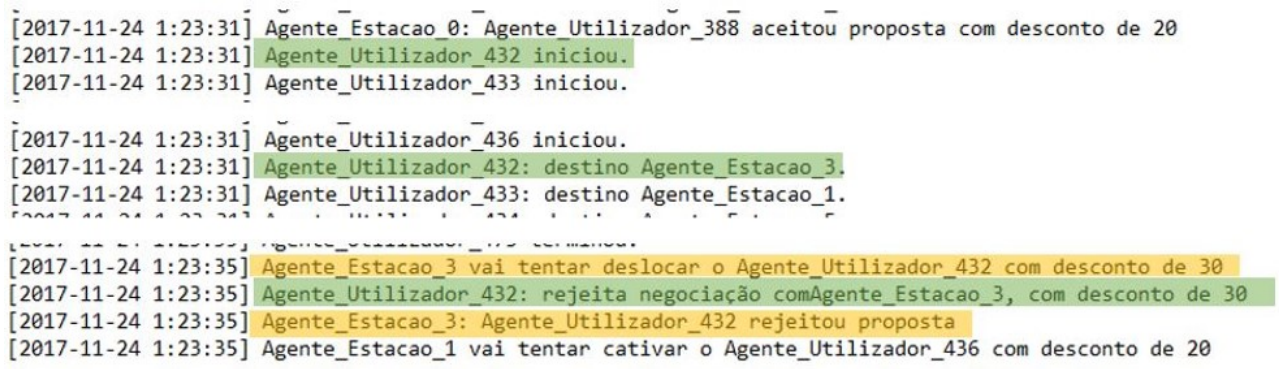

Figura 6.4: Exemplo de uma proposta de deslocação rejeitada por um utilizador

- Cenário em que se realiza uma uma proposta para deslocar um utilizador, e este entra num processo de contra propostas, solicitando um aumento maior.

  De notar no exemplo abaixo que o utilizador_413 tinha como destino inicial a estação_3. Pelo seu percurso rejeitou uma proposta da estação_1, que estaria num cenário com falta de equipamentos.

  Quando o utilizador entra na área de proximidade da estação destino, a mesma está sobrelotada e procura desloca-lo. O utilizador responde com uma contra proposta solicitando um desconto de 40% face aos 30% inicialmente propostos.

  Neste caso a estação_3 acabou por aceitar esta contra oferta do utilizador. Na tabela, é possível ver que de facto a estação 3 está com uma ocupação acima da sua capacidade máxima, tendo mais um equipamento acima daqueles que consegue armazenar.

```
[2017-11-24 1:23:28] Agente_Utilizador_412 iniciou.
[2017-11-24 1:23:28] Agente_Utilizador_413 iniciou.
[2017-11-24 1:23:28] Agente_Utilizador_408: destino Agente_Estacao_7.
[2017-11-24 1:23:28] Agente_Utilizador_409: destino Agente_Estacao_4.
[2017-11-24 1:23:28] Agente_Utilizador_412: destino Agente_Estacao_3.
[2017-11-24 1:23:28] Agente_Utilizador_413: destino Agente_Estacao_3.
[2017-11-24 1:23:28] Agente_Estacao_5 vai tentar cativar o Agente_Utilizador_362 com desconto de 20
[2017-11-24 1:23:32] Agente_Utilizador_409: Agente_Estacao_4 aceitou CFP, com desconto de 30.
[2017-11-24 1:23:32] Agente_Estacao_1 vai tentar cativar o Agente_Utilizador_413 com desconto de 20
[2017-11-24 1:23:32] Agente_Utilizador_413: rejeita negociação comAgente_Estacao_1, com desconto de 20
[2017-11-24 1:23:32] Agente_Estacao_1: Agente_Utilizador_413 rejeitou proposta
[2017-11-24 1:23:32] Agente_Utilizador_264 chegou ao destino
[2017-11-24 1:23:48] Agente_Estacao_5: recebeu entrega do Agente_Utilizador_406. Preço 46.24,  com desconto de 20.0 Preço final :36.99.
[2017-11-24 1:23:48] Agente_Estacao_3 vai tentar deslocar o Agente_Utilizador_413 com desconto de 30
[2017-11-24 1:23:48] Agente_Utilizador_413: realiza CFP À Agente_Estacao_3, com desconto de 40
[2017-11-24 1:23:48] Agente_Estacao_3: continuou negociacao com Agente_Utilizador_413
[2017-11-24 1:23:48] Agente_Utilizador_413: Agente_Estacao_3 aceitou CFP, com desconto de 40.
[2017-11-24 1:23:48] Agente_Estacao_7 vai tentar cativar o Agente_Utilizador_420 com desconto de 20
[2017-11-24 1:23:50] Agente_Estacao_1: recebeu entrega do Agente_Utilizador_495. Preço 24.41,  com desconto de 20.0 Preço final :19.53.
Nome      Px Py Raio C atual C max Reservas Entregas Oc ideal Oc atual Prop A Prop R % Aceites Prejuizos Lucro
----      -- -- ---- ------- ----- -------- --------- -------- -------- ------ ------ ---------- --------- -----
Estacao_4 87 26 28   25      32    64       64        0.79     0.78     67     9      0.87       0         2013.4
Estacao_0 4  54 15   1       17    36       22        0.82     0.06     32     6      0.82       0         510.7
Estacao_2 65 38 24   34      35    54       58        0.75     0.97     101    37     0.73       0         1382.68
Estacao_1 20 15 19   40      42    43       48        0.79     0.95     55     11     0.82       0         1268.27
Estacao_6 81 58 19   8       32    39       22        0.82     0.25     23     7      0.74       0         621.26
Estacao_7 14 76 28   25      42    55       45        0.75     0.6      50     12     0.79       0         1387.64
Estacao_5 31 33 18   9       24    60       49        0.77     0.38     59     10     0.84       0         1278.04
Estacao_3 59 53 28   32      31    53       60        0.81     1.03     83     20     0.8        14        1436.63
Estacao_8 0  1  29   6       18    44       35        0.83     0.33     46     12     0.78       0         1083.63

[2017-11-24 1:23:50] Agente_Estacao_5 vai tentar cativar o Agente_Utilizador_456 com desconto de 20
```

Figura 6.5: Exemplo de uma proposta de deslocação contra negociada por um utilizador

6.2.2 Negociação em caso de *underlotação*

Nas situações em que uma estação tenha falta de bicicletas para alugar, face ao valor que representa a ocupação ideal que a mesma deve manter, esta vai procurar cativar utilizadores que se encontrem dentro da sua área de proximidade. Perante estas propostas, os utilizadores podem ou não aceitar o processo de negociação iniciado pela estação.

Os seguintes exemplos procuram exemplificar alguns dos cenários que se desenrolaram neste contexto.

- Situação em que se realiza uma proposta de atração a um utilizador, e este aceita.

  Tal como assinalado na seguinte figura, o utilizador_1 tem como destino inicial a estação_4. Contudo, perante a aceitação de uma proposta de desconto realizada pela estação_8, que o tenta cativar, o utilizador decide aceitar a proposta e alterar o seu percurso.

  Momentos mais tarde, e porque nenhuma outra estação calhou de colocar propostas ao utilizador ou porque este as recusou, o utilizador entrega efetivamente a bicicleta na estação_8, recebendo o desconto que lhe foi prometido. De notar que pelo facto da estação estar num cenário de *underlotação* (menos importante) o desconto oferecido é inicialmente de apenas 20%.

```
[2017-11-24 0:55:9] Agente_Utilizador_1 iniciou.
[2017-11-24 0:55:9] Agente_Utilizador_0 iniciou.
[2017-11-24 0:55:9] Agente_Utilizador_0: destino Agente_Estacao_2.
[2017-11-24 0:55:9] Agente_Utilizador_1: destino Agente_Estacao_4.
[2017-11-24 0:55:9] Agente_Utilizador_3: destino Agente_Estacao_3.
[2017-11-24 0:55:11] Agente_Utilizador_13: destino Agente_Estacao_1.
[2017-11-24 0:55:11] Agente_Estacao_8 vai tentar cativar o Agente_Utilizador_1 com desconto de 20
[2017-11-24 0:55:11] Agente_Utilizador_1: aceitou  Agente_Estacao_8 com desconto de 20
[2017-11-24 0:55:11] Agente_Estacao_2 vai tentar cativar o Agente_Utilizador_3 com desconto de 20
[2017-11-24 0:55:11] Agente_Utilizador_14: destino Agente_Estacao_2.
[2017-11-24 0:55:11] Agente_Utilizador_3: aceitou  Agente_Estacao_2 com desconto de 20
[2017-11-24 0:55:11] Agente_Estacao_8: Agente_Utilizador_1 aceitou proposta com desconto de 20
[2017-11-24 0:55:11] Agente_Estacao_7 vai tentar cativar o Agente_Utilizador_5 com desconto de 20
[2017-11-24 0:55:24] Agente_Utilizador_1 chegou ao destino
[2017-11-24 0:55:24] Agente_Utilizador_1 terminou.
[2017-11-24 0:55:24] Agente_Utilizador_21 chegou ao destino
[2017-11-24 0:55:24] Agente_Utilizador_21 terminou.
[2017-11-24 0:55:24] Agente_Estacao_7 vai tentar cativar o Agente_Utilizador_36 com desconto de 20
[2017-11-24 0:55:24] Agente_Estacao_0: recebeu entrega do Agente_Utilizador_26. Preço 20.25,  com desconto de 30.0 Preço final :14.18.
[2017-11-24 0:55:24] Agente_Utilizador_36: realiza CFP À Agente_Estacao_7, com desconto de 30
[2017-11-24 0:55:24] Agente_Estacao_8: recebeu entrega do Agente_Utilizador_1. Preço 34.13,  com desconto de 20.0 Preço final :27.3.
[2017-11-24 0:55:24] Agente_Estacao_4: recebeu entrega do Agente_Utilizador_21. Preço 22.0,  com desconto de 25.0 Preço final :16.5.
```

Figura 6.6: Exemplo de uma proposta aceite por um utilizador, cativado por uma estação

- Cenário oposto, em que se realiza uma proposta e o cliente recusa-a.

```
[2017-11-24 0:56:0] Agente_Utilizador_327 iniciou.
[2017-11-24 0:56:0] Agente_Utilizador_328 iniciou.
[2017-11-24 0:56:0] Agente_Utilizador_329 iniciou.
[2017-11-24 0:56:0] Agente_Utilizador_324: destino Agente_Estacao_1.
[2017-11-24 0:56:0] Agente_Utilizador_325: destino Agente_Estacao_4.
[2017-11-24 0:56:0] Agente_Utilizador_326: destino Agente_Estacao_8.
[2017-11-24 0:56:0] Agente_Utilizador_327: destino Agente_Estacao_4.
[2017-11-24 0:56:0] Agente_Utilizador_328: destino Agente_Estacao_2.
[2017-11-24 0:56:0] Agente_Utilizador_329: destino Agente_Estacao_6.
[2017-11-24 0:56:16] Agente_Utilizador_327: aceitou  Agente_Estacao_1 com desconto de 20
[2017-11-24 0:56:16] Agente_Estacao_1 vai tentar cativar o Agente_Utilizador_328 com desconto de 20
[2017-11-24 0:56:16] Agente_Utilizador_328: rejeita negociação comAgente_Estacao_1, com desconto de 20
[2017-11-24 0:56:16] Agente_Estacao_1: Agente_Utilizador_327 aceitou proposta com desconto de 20
[2017-11-24 0:56:16] Agente_Estacao_1: Agente_Utilizador_328 rejeitou proposta
[2017-11-24 0:56:16] Agente_Estacao_1 vai tentar cativar o Agente_Utilizador_287 com desconto de 20
```

Figura 6.7: Exemplo de rejeição de uma proposta para cativar um utilizador

- Cenário em que uma estação tenta cativar um utilizador e este realiza uma contra-proposta, onde se pode verificar o pedido de aumento do desconto de 20% para 30%.

  Neste caso a estação aceita esta contra proposta, sendo que o utilizador entrega a bicicleta momentos mais tarde nesta estação, pagando um valor com o desconto acordado.

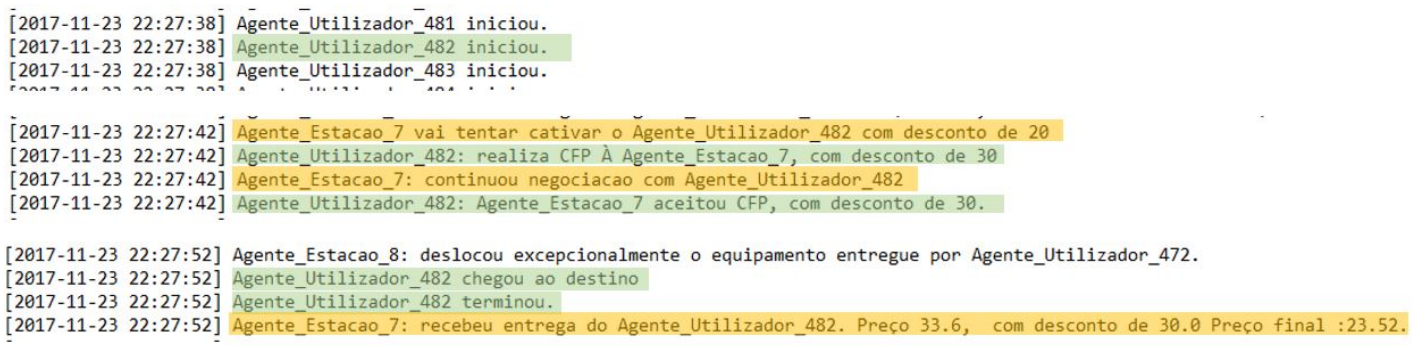

Figura 6.8: Exemplo de um cenário de contra-negociação num processo de atração de utilizadores

6.3 Situações excecionais

- Podem ocorrer situações em que uma estação, por estar com défice de bicicletas disponíveis face àquilo que devia ser a sua ocupação ideal, vai tentar cativar utilizadores para si.

  Contudo, ao realizar este processo, a estação não tem conhecimento do eventual fluxo de utilizadores que por si só já se dirige a esta estação como destino. Esta situação pode levar a cenários em que, apesar de se terem cativado utilizadores para devolverem bicicletas numa dada estação, quando o utilizador chegar a essa estação, a mesma já se encontra sobrelotada, porque entretanto chegou um número inesperado de utilizadores.

  Nestas situações, o utilizador não é novamente desviado para outro ponto de entrega, sendo a estação responsável por de algum modo levar as bicicletas em excesso para outras estações que as possam recolher.

  A seguinte imagem procura descrever a ocorrência desta situação.

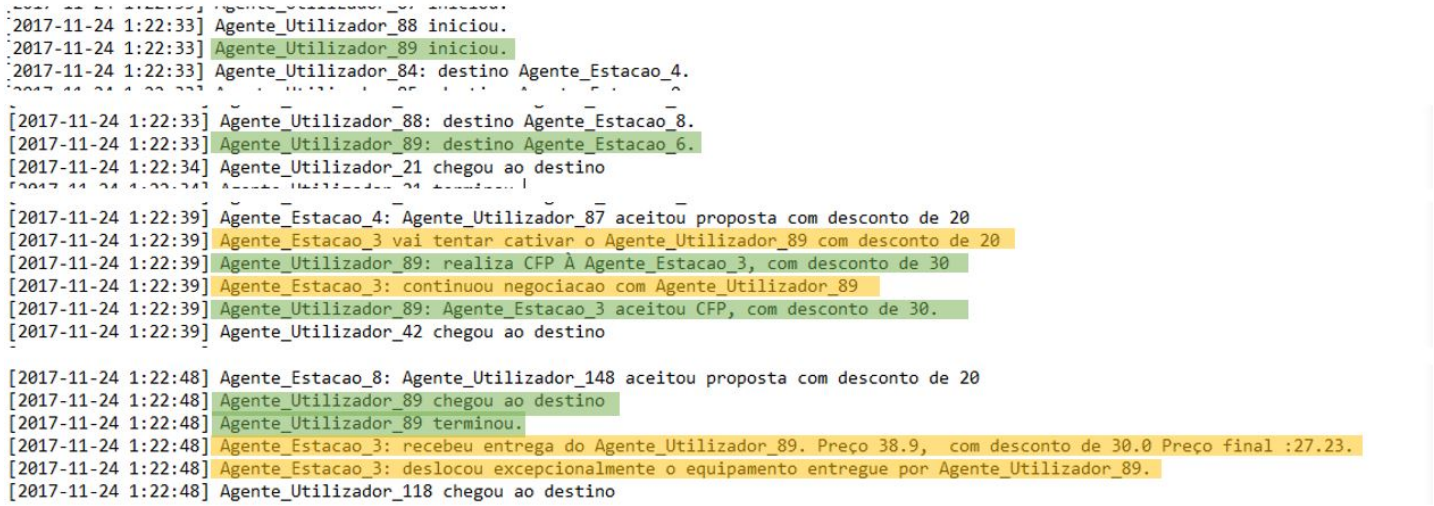

Figura 6.9: Exemplo de um cenário inesperado de sobrelotação

- Situações em que um utilizador entra no sistema, escolhe uma determinada estação de origem mas a mesma não tem bicicletas disponíveis para alugar. Nestas situações o utilizador termina a sua atuação no sistema e não reserva nenhum equipamento, pois não existem equipamentos na estação onde ele se encontra.

  No exemplo abaixo, os utilizadores 265 e 273 iniciam-se em instantes muito próximos na mesma estação de origem. Como a estação_2 não tem bicicletas disponíveis naquele instante, ambos terminam o seu papel no sistema.

```
[2017-11-24 1:23:3] Agente_Utilizador_264 iniciou.
[2017-11-24 1:23:3] Agente_Utilizador_265 iniciou.
[2017-11-24 1:23:3] Agente_Utilizador_266 iniciou.
[2017-11-24 1:23:3] Agente_Utilizador_264: destino Agente_Estacao_0.
[2017-11-24 1:23:4] Agente_Utilizador_265: selecionou a  Agente_Estacao_2 que nao tem stock disponivel!
[2017-11-24 1:23:4] Agente_Utilizador_265 terminou.
[2017-11-24 1:23:4] Agente_Utilizador_266: destino Agente_Estacao_4.
[2017-11-24 1:23:4] Agente_Utilizador_271 iniciou.
[2017-11-24 1:23:4] Agente_Utilizador_272 iniciou.
[2017-11-24 1:23:4] Agente_Utilizador_273 iniciou.
[2017-11-24 1:23:4] Agente_Estacao_3: Agente_Utilizador_204 aceitou proposta com desconto de 20
[2017-11-24 1:23:4] Agente_Utilizador_273: selecionou a  Agente_Estacao_2 que nao tem stock disponivel!
[2017-11-24 1:23:4] Agente_Utilizador_273 terminou.
[2017-11-24 1:23:4] Agente_Utilizador_274: destino Agente_Estacao_1.
```

Figura 6.10: Exemplo do cenário de ausência de equipamentos para alugar

Um outro exemplo da mesma situação, onde é possível ver que o utilizador_485 tenta iniciar na estação_5, mas a mesma não tem stock disponível. Por não ter stock disponível, é possível ver que a estação procura cativar o utilizador_396 que por acaso entrou na sua área de proximidade.

Na tabela, é possível confirmar que de facto a capacidade da estação atual está a zero, assim como a sua ocupação atual.

```
[2017-11-24 1:23:36] Agente_Utilizador_484: destino Agente_Estacao_7.
[2017-11-24 1:23:36] Agente_Utilizador_485: selecionou a  Agente_Estacao_5 que nao tem stock disponivel!
[2017-11-24 1:23:36] Agente_Utilizador_485 terminou.
[2017-11-24 1:23:37] Agente_Utilizador_392 chegou ao destino
[2017-11-24 1:23:40] Agente_Estacao_2: Agente_Utilizador_491 aceitou proposta com desconto de 20
[2017-11-24 1:23:40] Agente_Estacao_5 vai tentar cativar o Agente_Utilizador_396 com desconto de 20
[2017-11-24 1:23:40] Agente_Utilizador_396: aceitou  Agente_Estacao_5 com desconto de 20
[2017-11-24 1:23:40] Agente_Estacao_5: Agente_Utilizador_396 aceitou proposta com desconto de 20
```

Nome	Px	Py	Raio	C atual	C max	Reservas	Entregas	Oc ideal	Oc atual	Prop A	Prop R	% Aceites	Prejuizos	Lucro
Estacao_0	4	54	15	2	17	35	22	0.77	0.12	30	6	0.81	0	510.7
Estacao_4	87	26	28	15	32	63	53	0.81	0.47	65	7	0.89	0	1729.86
Estacao_2	65	38	24	23	35	54	47	0.82	0.66	97	36	0.72	0	1088.72
Estacao_3	59	53	28	33	31	52	60	0.79	1.06	79	19	0.8	14	1436.63
Estacao_7	14	76	28	17	42	54	36	0.8	0.4	45	11	0.79	0	1102.89
Estacao_5	31	33	18	0	24	60	40	0.82	0.0	51	8	0.85	0	1037.99
Estacao_6	81	58	19	1	32	39	15	0.83	0.03	18	7	0.69	0	403.14
Estacao_1	20	15	19	28	42	43	36	0.79	0.67	49	11	0.8	0	926.24
Estacao_8	0	1	29	2	18	42	29	0.82	0.11	45	12	0.78	0	832.84

Figura 6.11: Exemplo do cenário de ausência de equipamentos para alugar

7. Dificuldades encontradas e soluções aplicadas

Nesta secção será apresentada uma breve referência às várias dificuldades que apareceram durante o decorrer do desenvolvimento do projeto.

As dificuldades encontradas serão listadas junto com a sua resolução, descrevendo o motivo que justificou a tomada das decisões na fase de implementação do sistema.

- Na modelação do sistema, foi definido que qualquer um dos agentes teria uma componente que permitisse a receção de mensagens. A ideia inicial seria que quando uma estação desejasse entrar num processo de negociação com um dado utilizador, a mesma criava um canal de comunicação especificamente orientado para a negociação com esse utilizador.

  Como a receção e tratamento das mensagens recebidas pelo agente Estação é implementada num *CyclicBehaviour*, não é possível a criação de novos *CyclicBehaviours* sempre que se deseja iniciar uma negociação com um novo utilizador, dado que estes comportamentos teriam a mesma função de receção de mensagens.

  A solução passou assim pela criação a classe Negociação (secção 3), onde é possível guardar um histórico de mensagens trocadas entre dois agentes, assim como outros parâmetros relevantes no contexto de negociação do *SPB*. No lado dos agentes Utilizador e Estação, existe assim uma variável *negociações* (secção 3.2) que mapeia o AID do agente para uma instância da classe *Negociacao*, permitindo assim saber a que proposta um determinando agente está a responder.

- Quando um processo de negociação entre uma estação e um utilizador termina com sucesso, a estação regista na sua variável *negociacoes*, na posição referente ao utilizador em causa, o desconto que foi acordado entre ambas as partes.

  Este processo permite que, em cenários em que uma estação cativa utilizadores até si, quando estes cheguem até ela a estação possa analisar se aquele utilizador existe na sua variável *negociacoes* e, caso exista, retirar dela o valor de desconto aceite entre ambos os agentes.

  Neste casos, o utilizador paga um preço com o desconto acordado e o sistema comporta-se de forma esperada.

  Contudo, no cenário oposto de sobrelotação, uma estação procura afastar os utilizadores que nela vão entregar equipamentos. Isto significa que, quando uma negociação neste contexto terminar com o acordo de uma das partes, o facto da estação que iniciou a proposta de deslocação registar o último

desconto associado ao utilizador, não será relevante, porque não será mais esta estação a receber o utilizador.

A solução poderia passar por fazer a estação que deslocou o utilizador comunicar com a estação que selecionou como novo destino do mesmo, informando algo como "Estação **X**, caso receba o utilizador **K** ofereça-lhe **N** de desconto". A solução mais simples passou por incluir no utilizador o valor do desconto que lhe seja oferecido.

Se um utilizador não aceitar ou não se cruzar com nenhuma proposta, o seu desconto será naturalmente 0 mas, nas situações em que o mesmo aceita uma proposta de deslocação, o utilizador carrega consigo o valor de desconto que lhe foi oferecido. Quando o utilizador chegar ao destino, informa assim a estação que, apesar de ela não ter conhecimento nem ter sido originado nela, outra estação lhe propôs um desconto caso entregasse ali o equipamento alugado.

- Numa abordagem inicial, ao fim da criação de algumas dezenas de utilizadores, era possível visualizar alguma descredibilidade nos dados apresentados na tabela de monitorização, pois algumas estações apresentavam valores de capacidade atual negativos.

  Tal problema decorria no facto de um utilizador selecionar uma estação de origem aleatoriamente e nela iniciar o seu percurso até à estação destino.

  Contudo, não faz sentido reservar uma bicicleta e decrementar o stock em mão de uma estação quando a estação já não tem efetivamente bicicletas disponíveis para o fluxo de clientes que chegam até si num dado momento.

  A solução passou pela introdução de um condição de controlo, onde caso o utilizador se inicie numa estação sem stock de bicicletas disponível, o utilizador não pode efetuar uma reserva e termina o seu papel de atuador no sistema de aluguer.

  Desta forma, é simulado o comportamento real de um cliente e é mantida a coerência na evolução dos dados de uma estação. (ver secção 6.3)

- Quando um utilizador tem como destino uma dada estação, e esta se encontra num estado de sobrelotação, assim que o utilizador entre na área de proximidade da estação, esta vai tentar desloca-lo para outra estação próxima e com capacidade de recolher equipamentos.

  Contudo, caso o cliente não aceite nenhuma proposta, a estação terá que inevitavelmente receber a entrega do utilizador.

  Numa outra perspetiva, quando uma estação tem um défice no número de bicicletas disponível, esta vai tentar cativar utilizadores para se deslocarem até si. Em algumas simulações realizadas, foi possível observar que em determinados momentos, uma estação cativava utilizadores para recuperar a sua ocupação atual mas, quando alguns destes chegavam à estação para entregar a bicicleta alugada, a estação já estava num estado de sobrelotação. Nesta situação, dado que a estação foi quem cativou o utilizador com uma

proposta de desconto, não faria sentido manda-lo posteriormente para outra estação, tentando oferecer-lhe outro desconto.

Nestes dois casos, quer por negação de propostas de deslocação como por não previsão do número de chegadas de utilizadores, a estação vai irreversivelmente aumentar o seu nível de sobrecarga.

Nestes casos, foi assumido que é da responsabilidade da estação descolar as bicicletas em excesso. Este processo pode ser realizado por um funcionário que desloca a bicicleta para uma estação que tenha capacidade para a receber.

Uma solução mais complexa passaria por introduzir um novo nível de comunicação entre as estações, para que estas pudessem comunicar entre si quando deslocam utilizadores para outras estações, criando assim um efeito de previsão de chegadas nas estações que foram definidas como novo destino dos utilizadores realocados.

Na implementação atual, considera-se que um funcionário da estação pega na bicicleta impossível de armazenar e leva-a para outra estação. Este processo consome um período de 3 segundos, que devido ao ritmo da simulação do sistema permite que outras bicicletas se acumulem na estação, até à "chegada"deste funcionário fictício.

Este processo decrementa o número de bicicletas na estação, reduzindo assim gradualmente a sua sobrelotação, à medida que este "funcionário"realiza a tarefa marcada como *deslocação excecional*. (ver secção 6.3)

- Não tanto como dificuldade, mas como limitação do sistema de aluguer na implementação atual, existe o seguinte cenário: Uma estação **A** pode ver o seu stock atual a ser reduzido à medida que novos utilizadores iniciem nela o seu percurso.

  Contudo, se nenhum utilizador com origem noutra estação se cruzar na área de proximidade da estação **A** e, se nenhuma outra estação em caso de sobrelotação, encaminhar utilizadores para a estação **A**, esta nunca terá possibilidade de incrementar o seu stock disponível de equipamentos e melhorar a sua ocupação.

  Uma possível solução poderia passar por alargar, nestas situações em que o stock está muito próximo de zero, a área de proximidade da estação **A**, tentando dessa forma aumentar a probabilidade de detetar utilizadores no seu percurso e com isso iniciar processo de negociação para os cativar a entregar os equipamentos naquela estação.

8. Conclusão e trabalho futuro

Como referido ao longo do presente relatório, o objetivo principal do projeto enquadra-se no desenvolvimento e especificação de uma estrutura de interação e comunicação entre agentes. Neste sentido, foi inicialmente feita uma descrição geral do problema, de forma a levantar as entidades e funcionalidades que devem estar presentes num sistema de aluguer de bicicletas. Compreendido o domínio do projeto, é de seguida explicada a arquitetura idealizada para a resolução do mesmo assim como todo o processo de implementação da mesma.

No capítulo referente à arquitetura da solução implementada, além de uma descrição dos seus componentes, são apresentados esquemas e diagramas que permitem a melhor compreensão dos módulos elaborados. Para estes, foi ainda apresentado e explicada cada variável das classes que constituem estes módulos. No fecho da modelação da arquitetura do projeto, é ainda realizada uma ponte entre a arquitetura desenvolvida e as características das arquiteturas de agentes existentes.

Na seguinte secção é descrita a especificação dos protocolos de comunicação, consoante os cinco processos principais de diálogo entre os agentes: Inicialização, Notificação, Informação, Negociação e Monitorização. Para cada tipo de ato de comunicação foram associadas diferentes *performatives*, como forma de descrever o sentido e conteúdo da mensagem. Também como forma de análise visual destes processos, foram criados esquemas ilustrando o funcionamento da comunicação entre agentes nas diferentes situações, através de diagramas com notação *Agent UML*.

Cada um dos módulos criados foi detalhadamente exposto no capítulo referente à implementação do projeto. Para cada sub-sistema, foi feita uma descrição técnica do seu funcionamento, pormenorizando os comportamentos e a execução dos agentes que os integram. Além disto, é explicado com relevo todo o processo de negociação entre agentes, sendo este o aspeto fulcral do funcionamento do sistema de partilha de bicicletas.

Por último, é apresentada uma secção com a demonstração dos resultados obtidos, procurando comprovar o funcionamento do programa implementado. Neste ponto, destaca-se a ilustração da ocorrência dos diferentes atos de comunicação e negociação apresentados, através de excertos de execução do sistema em tempo real.

Como aspetos positivos, destaca-se o facto de nesta implementação inicial o sistema de aluguer conseguir dar resposta a uma procura substancialmente alta e chegada aleatória de utilizadores, conseguindo fazer uma gestão sustentada das estações com base nos principais cenários assinalados.

Como trabalho futuro, assinala-se a introdução de mais algumas condições na execução dos agentes, como forma de lidar com algumas situações excecionais levantadas na demonstração do sistema. Neste aspeto, podem ainda ser feitas algumas otimizações no processo de troca de mensagens entre estações, dado que uma estação não precisa de conhecer todas as estações do sistema, mas sim

apenas a sua vizinhança.

Isto permite reduzir eventuais custos com a troca de mensagens entre agentes que se encontrem a grandes distâncias. Esta otimização da troca de mensagens foi contudo tida em conta nos agentes utilizadores, onde os sensores que equipam as bicicletas só comunicam de facto com as estações que verificam um conjunto de condições. Numa abordagem mais descuidada, os sensores poderiam sempre comunicar com todas as estações, e seriam as estações a determinar se o utilizador podia ou não ser alvo de uma negociação.

Bibliografia

[1] Fipa.org. (2002). *FIPA Communicative Act Library Specification* [online] Available at: http://www.fipa.org/specs/fipa00037/SC00037J.html [Accessed 7 Nov. 2017].

[2] Girardi, R., Ferreira, S. L. C., (2002). Arquiteturas de Software baseadas em agentes. [online] Available at:

http://arquivo.ulbra-to.br/ensino/43020/artigos/anais2002/EIN2002/EIN-2002-Arquiteturas.pdf
[Accessed 7 Nov. 2017].

[3] Bauer, B., Müller, J. P., Odell, J. (2001). *Agent UML: A formalism for specifying multiagent software systems*

[4] Bellifemine, F. L., Caire, G., Greenwood, D. (2007). *Developing multi-agent systems with JADE (Vol. 7)*

[5] James Odell, H. Van Dyke, Bernhard Bauer. (2000). *Extending UML for agents*