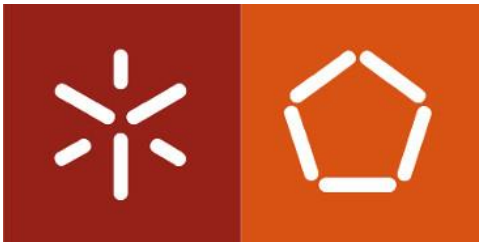
# UNIVERSIDADE DO MINHO



## Agentes e Sistemas Multiagente

Mestrado Integrado em Engenharia Biomédica

Sistemas Inteligentes

(1º Semestre/ Ano Letivo 2021/2022)

Grupo 2

Lara Alexandra Pereira Novo Martins Vaz (A88362)

Mariana Lindo Carvalho (A88360)

Tiago Miguel Parente Novais (A88397)

Braga

5 de janeiro de 2021

# Índice

# 1. Introdução

## 1.1. Inteligência Artificial e Sistemas Multiagente

A Inteligência Artificial (IA), segundo John McCarthy, é a ciência e a engenharia de tornar máquinas inteligentes, especialmente, programas de computador. A IA está relacionada com a tarefa de recorrer a computadores para entender a inteligência humana, no entanto a IA não precisa de se confinar a métodos que são biologicamente observáveis [1].

Relativamente à Inteligência Artificial Distribuída, esta pode ser definida como o estudo, a construção e a aplicação de sistemas em que diversas entidades computacionais (agentes ou Sistemas Multiagente - SMA) interagem e perseguem um conjunto de objetivos e/ou realizando um conjunto de tarefas. Por sua vez, um Sistema Multiagente compreende um conjunto agentes que cooperam por forma a solucionar um dado problema, que normalmente está além das suas capacidades individuais [2].

## 1.2. Agentes

Os agentes são entidades que habitam em ambientes complexos, sentem esse ambiente e atuam de modo autónomo, procurando executar um conjunto de tarefas para as quais receberam procuração [3]. Estes possuem uma série de caraterísticas que são comuns a todos os agentes:

- **Autonomia:** operam sem intervenção de outros agentes e controlam as suas ações e o seu estado de conhecimento interno;
- **Reatividade:** percecionam os eventos que ocorrem no seu universo de discurso e respondem adequada e atempadamente a mudanças ocorridas nesse ambiente;
- **Iniciativa:** tomam iniciativa, conduzindo as suas próprias ações mediante um comportamento dirigido por objetivos;
- **Sociabilidade:** relacionam-se com outros agentes, comunicando, competindo ou cooperando na resolução de problemas que lhes sejam colocados.

Existem ainda outras caraterísticas que são facultativas e cuja sua presença varia de acordo com o nível de inteligência do agente:

- **Mobilidade:** capacidade de se movimentar através da rede formada pelos seus pares, executando as tarefas de que foi incumbido;
- **Intencionalidade:** capacidade que o agente apresenta para a definição de objetivos e das estratégias para os atingir;
- **Aprendizagem:** capacidade que o agente apresenta de adquirir conhecimento. A atualização da base de conhecimento é feita através da assimilação de padrões de comportamento ou de preferências;
- **Competência:** um agente é competente quando conduz com sucesso e eficiência as tarefas de que é incumbido, estando a competência normalmente relacionada com a confiança depositada no agente por terceiros;
- **Veracidade:** um agente exibe veracidade quando não fornece, de forma intencional, informação falsa;
- **Racionalidade:** um agente racional não aceita realizar tarefas que avalie impossíveis de executar, contraditórias com os seus princípios ou quando não são compensados em termos do risco, custo ou esforço;
- **Benevolência:** um agente benevolente adota como seus os objetivos de terceiros, desde que estes não entrem em conflito com os seus princípios de natureza ética e/ou deontológica, o que significa que não realizarão todas as tarefas que lhes sejam atribuídas;
- **Emotividade:** certas características próprias do ser humano têm vindo a migrar e a estabelecer-se como parte constituinte dos agentes.
- **Credibilidade:** capacidade do agente de criar uma suspensão de "descrença", levando o utilizador a aceitar temporariamente que o agente está "vivo" ou que é um "personagem real".
- **Personalidade:** capacidade do agente de se comportar de forma individual, o que torna o agente distinguível de seus pares [4].

## 1.3. Comunicação num SMA

A comunicação é essencial em ambientes onde existem diversas tarefas que concorrem para um dado fim de forma cooperante ou concorrente, como é o caso dos

SMA. Assim, é necessário assegurar a existência de suportes físicos, ou seja, de estruturas de rede para a propagação dos sinais de transporte de informação através do meio e de suportes de interação, que correspondem à pilha protocolar que assegura a troca das mensagens (sobre o suporte físico) e a sua compreensão [4].

### 1.3.1. Linguagens de Comunicação

Para que a comunicação seja possível, é também necessária a existência de uma linguagem de comunicação, que configura a forma pela qual é comunicado o conteúdo de uma mensagem [5]. Deste modo, para os agentes poderem interagir e comunicar entre si, é necessária a existência de uma linguagem comum, uma forma de interpretar o conhecimento trocado, a capacidade de trocar esse conhecimento e o conteúdo semântico deve ser independente do agente. De entre as principais linguagens que têm sido utilizadas para a comunicação entre agentes destacam-se a KQML (Knowledge Query Manipulation Language) e a FIPA ACL (Foundation for Intelligent Physical Agents - Agent Communication Language) [5].

### 1.3.2. Atos de Discurso

Os agentes são entidades com propriedades antropomórficas, e por isso, a comunicação entre seres racionais é um modelo de referência para a comunicação entre agentes. Uma análise restrita da comunicação humana é dada pela teoria designada por SAT (Speech Act Theory), que reduz a linguagem humana aos elementos indicativos de ação. A SAT analisa o discurso humano segundo três vetores:

- **Locução:** a articulação do emissor;
- **Elocução:** o significado pretendido para a articulação dada;
- **Perlocução:** a ação resultante da articulação no recetor.

A SAT recorre também ao termo “performative” para indicar o tipo especial de elocução que implica ação [5].

### 1.3.3. Ontologias

A troca de informação entre agentes exige mais que um formato comum e uma linguagem de comunicação comum. Estas entidades, especialistas na resolução de um determinado problema, autónomas e heterogéneas, têm de concordar num modelo conceptual da realidade, que permita capturar o conhecimento próprio dessa mesma realidade e que a permita compreender, isto é, acordar numa ontologia comum [6].

Uma ontologia fornece por outras palavras o vocabulário de representação para o domínio em questão e um conjunto de definições e axiomas que restringem o significado dos termos nesse vocabulário, de forma a permitir uma interpretação consistente e única. A adesão a uma ontologia comum garante a consistência (a mesma expressão tem o mesmo significado em qualquer agente) e a compatibilidade (qualquer conceito é designado pela mesma expressão em qualquer agente) da informação presente no sistema [7].

## 1.4. JADE

O Java Agent Development Framework (JADE) é um software desenvolvido em Java que permite a construção de aplicações baseadas em agentes, sendo compatível com as especificações FIPA. Mais do que um conjunto de APIs, o JADE é visto como um middleware para agentes, fornecendo uma plataforma de agentes e uma arquitetura de desenvolvimento.

Os programadores necessitam apenas de se concentrar nos mecanismos inteligentes e no funcionamento interno dos seus agentes, deixando o transporte de mensagens, a codificação/descodificação de estruturas de dados (marshalling/unmarshalling) e a gestão geral ao sistema subjacente. A versão 2.x do JADE é compatível com as especificações FIPA2000, implementando o Agent Management System (AMS), o Directory Facilitator (DF) e o Agent Communication Channel (ACC). A passagem de mensagens segue a FIPA-Agent Communication Language (FIPA-ACL) [8].

As principais vantagens do JADE verificam-se ao nível da interoperabilidade, do uso de um standard bem fundamentado, da portabilidade do código e de um conceito neutro de agente [8].

### 1.4.1. Containers

Um Container corresponde a cada instância que é colocada a correr no ambiente de execução do JADE, sendo que cada container pode conter vários agentes. Um conjunto de containers ativos designa-se por Platform.

O 1º container a ser inicializado numa plataforma deve ser um main-container e posteriormente, os restantes containers (não principais) devem ser informados sobre o "host" e a "port" para conseguirem encontrar o seu main-container. Se eventualmente outro main-container for ativado, o JADE assume que este constitui uma nova plataforma, para a qual, novos containers normais podem-se registar. Posto isto, um main-container difere dos containers normais, pois contém 2 tipos de agentes fundamentais:

- **Agent Management System (AMS):** fornece o serviço de identificação, isto é, permite que cada agente da plataforma tenha um nome único. Também representa a entidade na plataforma capaz de criar ou matar agentes;
- **Director Facilitator (DF):** fornece um serviço de Yellow Pages, permitindo que os agentes encontrem outros agentes necessários para atingir os seus objetivos [8].

### 1.4.2. Páginas Amarelas

Um serviço de "páginas amarelas" permite que os agentes publiquem um ou mais dos serviços que prestam, para que outros os agentes possam encontrá-los. O serviço de páginas amarelas no JADE (conforme especificação FIPA) é prestado por um agente denominado DF (Facilitador de diretoria). Cada plataforma compatível com FIPA hospeda um agente DF padrão. Outros agentes DF podem ser ativados e vários agentes DF (incluindo o padrão) podem ser federados para fornecer um único catálogo de páginas amarelas distribuído [9].

Sendo o DF um agente, é possível interagir com ele como trocando mensagens ACL usando uma linguagem de conteúdo adequada (a linguagem SL0) e uma ontologia adequada (a ontologia de gestão de agentes FIPA) de acordo com a especificação FIPA. A fim de simplificar essas interações, o JADE fornece a classe jade.domain.DFService através da qual é possível publicar e pesquisar serviços por meio de chamadas de métodos [9].

Um agente que deseje publicar um ou mais serviços deve fornecer ao DF uma descrição que inclui o seu AID (Identificador do Agente), possivelmente a lista de linguagens e ontologias que outros agentes precisam saber para interagir com ele e a lista de serviços publicados. Para cada serviço publicado, é fornecida uma descrição que inclui o tipo de serviço, o nome do serviço, as linguagens e ontologias necessárias para explorar esse serviço e uma série de propriedades específicas do serviço. As classes DFAgentDescription, ServiceDescription e Property, incluídas no pacote jade.domain.FIPAAgentManagement, representam as três abstrações mencionadas [9].

### 1.4.3. Agentes e a sua Comunicação

Os Agentes são criados através da extensão da classe Agent no IDE Eclipse. Contudo, é necessário programar o seu setup() de modo a adicionar-lhe behaviours, que consistem em comportamentos que o agente irá realizar assim que se encontrar vivo [9].

A comunicação entre agentes é assíncrona, isto é, existe uma queue para cada agente em que sempre que um agente recebe uma mensagem de outro, esta é adicionada à queue existente, notificando assim o agente [9].

As mensagens trocadas entre os agentes têm um formato específico que seguem as normas FIPA. Este formato contém vários campos importantes, tais como o agente que manda a mensagem, a sua lista de recetores, juntamente com o objetivo da comunicação (performative), o seu conteúdo e a linguagem e ontologia usadas. Todos os campos anteriormente referidos surgem para a correta interação entre os agentes e para a construção de um Sistema de Multiagentes eficiente [9].

### 1.4.4. Behaviours

Como supramencionado, os behaviours representam o comportamento do agente. Estes podem ser de vários tipos, como por exemplo, OneShotBehaviours, CyclicBehaviors, SimpleBehaviours, TickerBehaviours e os WakerBehaviours. Os ParallelBehaviours e SequentialBehaviours são usados em sistemas mais complexos [7][8].

De seguida, serão explicados os três behaviours mais utilizados e que possivelmente serão utilizados no desenvolvimento do Sistema Multiagentes deste projeto.

#### 1.4.4.1. OneShotBehaviours

Os OneShotBehviours fazem com que a ação do agente seja executada uma única vez. Normalmente, após executar essa ação, o agente é "morto", contudo, este pode continuar a existir como um agente residual que apenas execução uma determinada ação [8].

#### 1.4.4.2. CyclicBehaviours

Estes behaviours são implementados desde que o agente nasce e essa implementação permanece enquanto o agente estiver ativo. Deste modo, o agente estará sempre a realizar a mesma ação de forma cíclica [8].

#### 1.4.4.3. TickerBehaviours

São comportamentos que o agente possui de x em x tempo. São implementados tendo em conta o tempo que o agente poderá demorar a agir novamente. Os TickerBehaviours são considerados um subtipo dos CyclicBehaviours, uma vez que são comportamentos cíclicos executados periodicamente, de acordo com um valor definido pelo utilizador [8].

## 1.5. Linguagem de Modelagem Unificada (UML)

A UML tem como objetivo definir uma linguagem de modelação visual padronizada, na qual é estabelecida um esquema visual para a implementação de sistemas complexos através da utilização de um conjunto integrado de diagramas, nos quais são retratados os comportamentos e estrutura de um sistema [10].

Os focos principais que justificam a utilização dos diagramas UML são:

- As aplicações complexas necessitam da colaboração e do planeamento de várias equipas e, portanto, exige uma forma clara e concisa na comunicação entre estas;
- Os empresários não compreendem código, pelo que a UML se torna essencial de modo a comunicar a estes os requisitos, funcionalidades e processos do sistema;

- Existe uma economia de tempo quando é permitido às equipas visualizar os processos, interações entre utilizadores e a estrutura estática do sistema [11].

### 1.5.1. Diagramas UML

A UML está vinculada ao design e à análise orientados a objetos, fazendo uso de elementos e formas de associações entre eles para formar diagramas. Os modelos em UML podem ser classificados como:

- **Modelos estáticos:** aplicados em diagramas de classes e de pacotes, que descrevem a semântica estática de dados e mensagens. Dentro do desenvolvimento do sistema, os diagramas de classes são usados de duas maneiras diferentes, para dois propósitos diferentes. Primeiro, estes podem modelar um domínio de um problema conceitualmente. Uma vez que são de natureza conceitual, eles podem ser apresentados aos clientes. Em segundo lugar, os diagramas de classes podem modelar a implementação de classes. Relativamente aos diagramas de pacotes, estes agrupam classes em pacotes conceptuais para apresentação e consideração;
- **Modelos dinâmicos:** incluem diagramas de interação (ou seja, diagramas de sequência e colaboração/comunicação), gráficos de estado e diagramas de atividades;
- **Casos de uso:** a especificação de ações que um sistema ou classe pode executar interagindo com atores externos;
- **Modelos de implementação:** como modelos de componentes e diagramas de implementação, que descrevem a distribuição de componentes em diferentes Plataformas;
- **Object constraint language (OCL):** é uma linguagem formal simples para expressar mais semântica dentro de uma especificação UML. Pode ser usada para definir restrições no modelo, invariáveis, pré e pós-condições de operações e caminhos de navegação dentro de uma rede de objetos [12] [13] [14] [15] [16] [17].

Como neste trabalho apenas serão utilizados os diagramas de classes, de colaboração/comunicação e de atividades, então, estes serão abordados mais detalhadamente.

#### 1.5.1.1. Diagrama de Classes

Um diagrama de classes é um tipo de diagrama estrutural que contêm os blocos principais da construção de um método orientado a objetos. Este tipo de diagrama é utilizado para modelar o domínio do problema e a implementação das classes de agentes [10].

Os principais objetivos deste tipo de diagrama são a representação de vários aspetos relacionados com conceitos de programação orientada a objetos, uma implementação mais eficiente e rápida do design e análise da aplicação e serve de base para os diagramas de componentes e implementação [10].

#### 1.5.1.2. Diagrama de Colaboração/Comunicação

O diagrama de colaboração, ou diagrama de comunicação, apresenta uma representação de clara compreensão do sistema a ser desenvolvido. Assim, consiste numa sequência de interações ordenadas pela sua ordem de execução [11].

#### 1.5.1.3. Diagrama de Atividades

O diagrama de atividades é utilizado para modelar o aspeto comportamental de processos. Neste diagrama, uma atividade é modelada como uma sequência estruturada de ações, controladas potencialmente por nós de decisão e sincronismo. Os diagramas de atividades são compostos por partições (*swimlanes*), sendo que estas servem para indicar as diferentes ações que são executadas por diferentes agentes dentro de um processo [18].

## 1.6. Motivações e objetivos

A Indústria Farmacêutica é uma área comercial bastante competitiva, complexa e de investimentos avultados em pesquisa, desenvolvimento e produção de produtos farmacêuticos. Este é um dos mercados mais rentáveis e inovadores da atualidade, sendo que nele atuam grandes corporações mundiais, que investem continuamente no desenvolvimento e comercialização de seus produtos.

Hoje em dia, e especialmente, com o envelhecimento da população e com a nova era pandémica, a procura e a demanda por medicamentos tem vindo a aumentar consideravelmente, tornando-se assim mais provável a ausência ou baixa de stock de medicamentos nas farmácias. Contudo, esta é uma situação extremamente indesejável, uma vez que, muitos dos doentes necessitam de tomar imediatamente os fármacos. Deste modo, seria desejável a existência de mecanismos de IA para prever a quantidade e o tipo de produtos vendidos num determinado período. Por exemplo, de mês a mês, o algoritmo deveria verificar a quantidade de medicamentos vendidos de cada tipo e assumir que no mês seguinte seriam vendidas essas mesmas quantidades. Se não existisse em stock quantidades mínimas, iguais às vendidas no mês passado, então esses mecanismos de IA permitiriam que a farmácia encomendasse automaticamente mais medicamentos até ter o stock máximo.

Além disso, verifica-se que as pessoas atualmente compram mais medicamentos, cujo preço pode variar consideravelmente entre farmácias. Por isso, de forma a evitar gastos desnecessários por parte dos doentes, seria muito vantajoso a existência de, por exemplo, uma aplicação que permitisse comparar os preços de um determinado medicamento para todas as farmácias, que informasse o doente de quais as farmácias em que o medicamento se encontra disponível e também que indicasse qual a farmácia com o melhor preço. Isto permitiria que o cliente procedesse a uma compra informada, isto é, conhecendo todas as suas opções e podendo escolher comprar a mais barata, se assim o desejar.

Assumindo que o cliente utiliza a aplicação acima referida, esta dá ainda ao doente a liberdade de decidir se quer comprar o produto online e recebê-lo no seu domicílio ou se prefere encomendar o produto e ir levantá-lo na farmácia ou ainda se pretende simplesmente deslocar-se à farmácia e proceder à compra presencial do produto. Mais uma vez, verifica-se que o uso de aplicações baseadas em algoritmos de Inteligência Artificial facilitam o processo de compra de medicamentos efetuados pelos consumidores, tornando-o mais cómodo, rápido e barato (se assim o utente desejar).

Tendo por base essas motivações, neste trabalho, foi proposto o desenvolvimento de um Sistema Multiagente constituído por um agente **Gestor**[1] responsável por analisar

[1] Os nomes dos agentes e das classes utilizados no SMA desenvolvido, foram destacados a negrito, de modo a facilitar a leitura e compreensão deste documento.

os pedidos recebidos pelos **Cidadaos**[2] e gerir, de forma autónoma e inteligente, a compra e venda de medicamentos aos **Cidadaos**. O **Gestor** deverá também analisar os históricos de vendas mensais como forma de antecipar as necessidades das **Farmacias**. Para além disso, o sistema deve ainda proceder ao envio dos medicamentos ao respetivo domicílio dos **Cidadaos**.

Assim, foi concebida uma arquitetura baseada em agentes para a melhoria dos serviços de saúde para **Cidadaos**. Neste caso, a arquitetura do sistema é híbrida, uma vez que parte da arquitetura é centralizada, existindo um agente centralizado (**Gestor**) para gerir as compras entre **Cidadaos** e **Farmacias**. Contudo, entre os **Fornecedores** e as **Farmacias**, a arquitetura é distribuída, pois, não existe nenhum agente responsável pela comunicação entre **Fornecedores** e **Farmacias**. Para desenvolver o SMA em causa, foram utilizados diagramas Agent UML, que permitiram formalizar os protocolos de interação entre agentes. Os diagramas de classes, de colaboração e de atividades foram elaborados com o auxílio do API Visual Paradigm.

Posto isto, os principais objetivos deste projeto são:

- Aumentar a qualidade dos serviços de saúde e melhorar a sua acessibilidade com o uso da tecnologia digital;
- Melhorar a economia das farmácias, bem como tornar o trabalho no setor de saúde mais eficiente;
- Contribuir para a prevenção eficaz de doenças;
- Fortalecer a igualdade dos cidadãos;
- Economizar o tempo dos pacientes, assim como os gastos em produtos farmacêuticos.

De salientar que os diagramas apresentados podem sofrer alterações mais tarde, aquando da execução do código.

[2] Apesar de não existir nenhum agente designado Cidadaos, como a palavra no plural visa representar vários agentes do tipo Cidadao, então, nomes de agentes no plural também estão assinalados a negrito.

# 2. Sistema Multiagente no setor Farmacêutico

## 2.1. Levantamento de requisitos do projeto

Numa fase inicial, foi elaborado o esquema representado na Figura 1, que representa todos os agentes envolvidos, bem como o objetivo da cada um no Sistema Multiagente a desenvolver. Os agentes a implementar são: agente **Cidadao**, agente **Farmacia**, agente **Interface**, agente **Gestor** e agente **Fornecedor**.

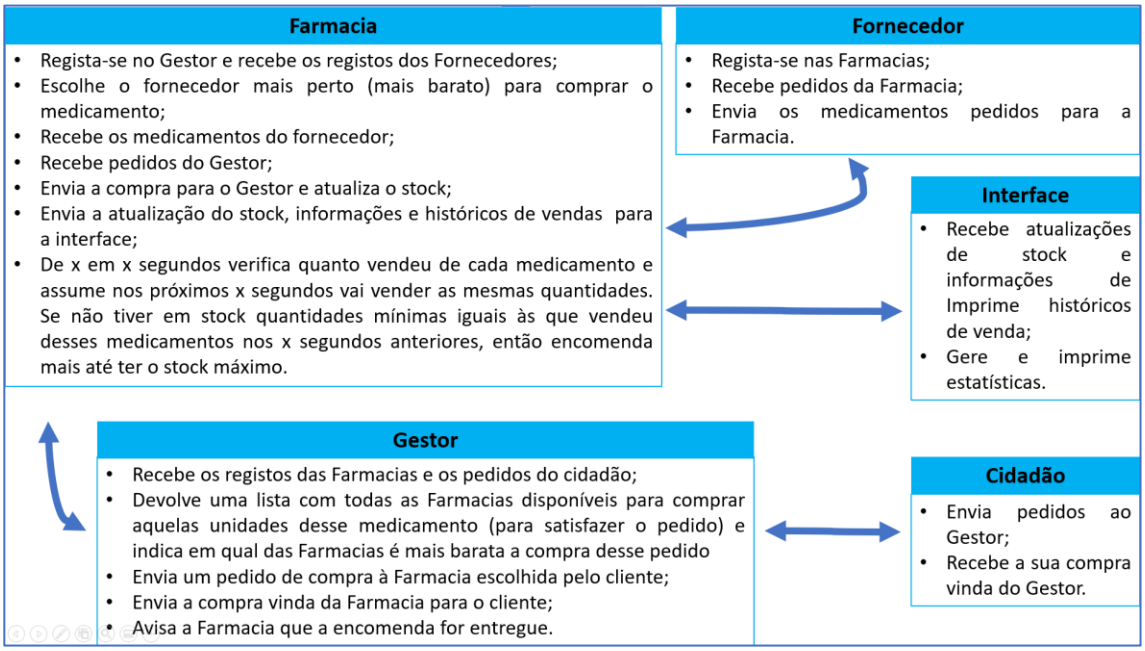



**Figura 1.** Esquema inicial com as relações entre agentes.

Através da análise da Figura 1, é possível proceder a uma análise detalhada das funções de cada agente e também apresentar algumas considerações que foram definidas para cada agente:

**Agente Cidadao**

Este agente simula as necessidades dos cidadãos. Assim, irá enviar o seu pedido de compra para o **Gestor**, juntamente com a sua localização, de modo que este lhe indique todas as farmácias onde ele pode efetuar a sua compra, assim como, a farmácia que possui o menor custo para o seu pedido. Por último, este recebe a sua compra vinda do **Gestor**.

A nível de considerações para este agente, considerou-se que cada **Cidadao** tem um padrão de compra fixo (por exemplo compra de 10 em 10 segundos) e compra n

unidades de um único medicamento, selecionado de forma aleatória a partir de uma lista de medicamentos.

**Agente Farmacia**

Inicialmente, este agente regista-se no **Gestor**, de modo que este saiba que essa **Farmacia** é uma farmácia válida e disponível para vender medicamentos. É de realçar que este agente tem posição fixa, sendo a sua posição conhecida pelo **Gestor**, aquando do registo da **Farmacia** no **Gestor**.

Adicionalmente, este agente é responsável pela escolha do **Fornecedor** a quem vai comprar os medicamentos, sendo que escolhe sempre o **Fornecedor** cujo custo da compra for o menor. O agente **Farmacia** recebe também os medicamentos que solicitou ao **Fornecedor**, bem como os pedidos de compra vindos do **Gestor**. Sempre que recebe uma solicitação do agente **Gestor**, este envia-lhe o respetivo pedido e atualiza o seu stock. Essa atualização de stock bem como informações e históricos de venda são, por sua vez, comunicados ao agente **Interface**.

Este agente é um agente proativo, uma vez que, de x em x segundos, verifica quanto vendeu de cada medicamento e assume nos próximos x segundos vai vender as mesmas quantidades. Se não tiver, em stock, quantidades mínimas desses medicamentos iguais às que nos x segundos anteriores, então encomenda mais até ter o stock máximo. Assim, este agente prevê as necessidades de compra dos **Cidadaos**, permitindo uma mitigação dos custos totais da **Farmacia**, ao evitar compras desnecessárias ao **Fornecedor**.

**Agente Gestor**

Este agente recebe os registos das **Farmacias**, assim como os pedidos de compra dos **Cidadaos**. Face aos pedidos dos **Cidadaos**, devolve uma lista com todas as **Farmacias** disponíveis para comprar as unidades do medicamento requisitado e indica também em qual das **Farmacias** fica mais barata a compra.

Após receber a seleção de **Farmacia** do **Cidadao**, o **Gestor** envia o pedido da respetiva compra para a **Farmacia** escolhida e assim que obtiver o pedido, reencaminha-o para o **Cidadao**. Por último, avisa a **Farmacia** que a encomenda foi entregue.

**Agente Fornecedor**

Este agente começa por se registar nas **Farmacias**, de modo que estas conheçam os medicamentos que este vende, assim como a sua localização. O agente **Fornecedor** recebe pedidos da **Farmacia** sempre que esta necessita de repor o seu stock e envia-lhe os medicamentos pedidos. É também de realçar que se considerou que todos os **Fornecedores** vão vender exatamente os mesmos medicamentos e que aceitam sempre os pedidos das **Farmacias**, pois, tem stock ilimitado e vendem todos os medicamentos que as **Farmacias** possam querer.

**Agente Interface**

Este é o agente com o qual o utilizador vai interagir, como forma de observar a gestão das **Farmacias**. O agente **Interface** recebe do agente **Farmacia** as suas atualizações de stock, informações e históricos de venda, imprimindo essas mesmas informações de cada farmácia.

Para além das considerações que já foram apresentadas, neste trabalho, considerou-se também que o custo dos medicamentos, quer os que as **Farmacias** compram aos **Fornecedores**, quer os que os **Cidadaos** pedem ao **Gestor** para comprar, estão sempre associados ao custo de transporte desses medicamentos. Assim, quanto maior for a distância entre o comprador/vendedor de medicamentos, maior será o custo do medicamento, sendo que por cada unidade de distância, o custo aumenta 1€.

Através do uso de metodologias de Agent UML, foram elaborados três diagramas – diagrama de classes, diagrama de colaboração/comunicação e diagrama de atividades – de forma a evidenciar, as caraterísticas e os tipos de agentes envolvidos, as caraterísticas fundamentais de interação entre agentes (processos de negociação) e ainda, a arquitetura e pipeline de execução do Sistema Multiagente.

## 2.2. Diagrama de Classes

A Figura 2 representa o diagrama de classes desenvolvido, sendo posteriormente fornecida uma descrição. De salientar que as operações representadas em cada classe estão apresentadas sob uma forma generalizada, podendo sofrer alterações mais tarde, aquando da execução do código.

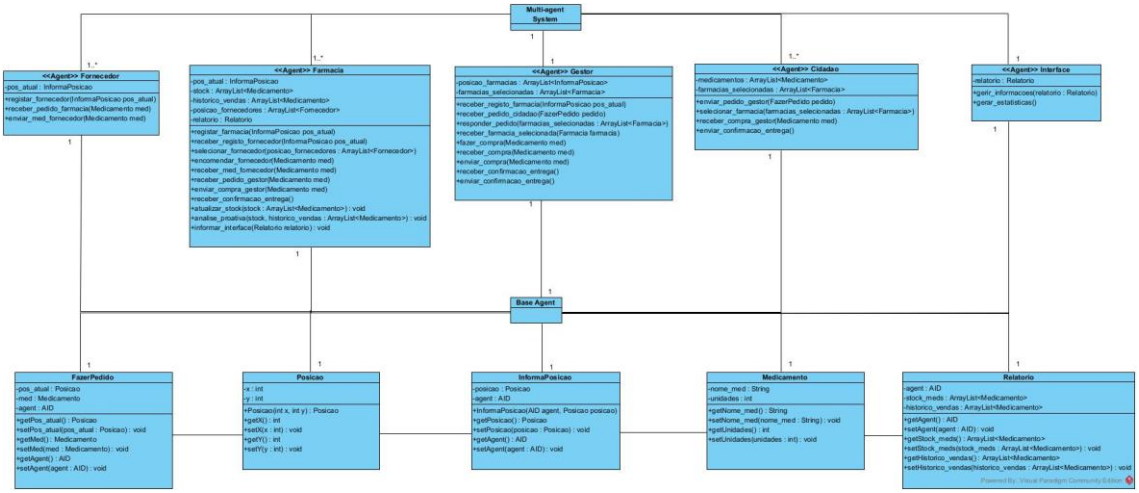



**Figura 2.** Diagrama de classes do SMA desenvolvido.

Através da análise da Figura 2, é possível verificar a existência de 5 classes que constituem um Base Agent e que serão utilizadas nos atributos e nas operações dos diversos agentes apresentados neste SMA. É de realçar que nessas 5 classes, as operações são sempre as mesmas, correspondendo aos gets e sets dos respetivos atributos e que existe apenas 1 classe de cada tipo.

Começando pela classe **Posicao**, esta contém 2 atributos do tipo inteiro, que são o *x*[3] e o *y* e que indicam as coordenadas x e y, respetivamente, do **Cidadao** que faz o pedido de compra. O objetivo desta classe é, portanto, fornecer a posição de um determinado agente.

Na classe **Medicamento** também existem 2 atributos. Um deles é a string que contém o nome do medicamento (*nome_med*) e o outro é um inteiro que contém as *unidades* do respetivo medicamento. Deste modo, esta classe dá a conhecer o medicamento que se pretende comprar/vender e em que quantidade.

Relativamente à classe **FazerPedido**, esta é utilizada pelos **Cidadaos** para o envio dos seus pedidos de compra ao **Gestor**. Contém 3 atributos, sendo eles a *pos_atual* que diz respeito a um objeto da classe **Posicao**, o *med* que é um objeto do tipo **Medicamento** e o *agent* que é um objeto que contém o AID do agente **Cidadao** que fez o pedido.

A classe **InformaPosicao** permite que um agente informe outro(s) agente(s) acerca da sua posição atual. Esta contém 2 atributos, sendo um deles a *posicao*, que é um objeto do tipo **Posicao** e o AID do *agent* que pretende informar o(s) outro(s).

[3] Os nomes dos atributos pertencentes aos agentes e às classes encontram-se em itálico, de modo a facilitar a leitura e compreensão deste documento.

A última classe presente no diagrama, que não corresponde a um agente, é a classe **Relatorio**, cujo objetivo é fornecer um relatório com os stocks e com o histórico de vendas à **Interface**. Esta contém 3 atributos, que são o AID do agent do tipo **Farmacia** que vai enviar as informações, a quantidade de medicamentos que essa **Farmacia** tem em stock (*stock_meds*) e o histórico de vendas dessa **Farmacia** (*historico_vendas*), sendo estes dois últimos atributos do tipo ArrayList de **Medicamentos**.

Relativamente aos agentes, existem 5 agentes, todos ligados ao Sistema Multiagente, sendo eles: **Fornecedor**, **Farmacia**, **Gestor**, **Cidadao** e **Interface**.

Começando pela classe **Fornecedor** (tipo *Agent*), esta é constituída pelo **atributo** *pos_atual* que é do tipo **InformaPosicao**, e pelas **operações** *registar_fornecedor*[4], *receber_pedido_farmacia* e *enviar_med_fornecedor*, sendo que o número de **Fornecedores** varia desde 1 até infinito. A operação *registar_fornecedor* consiste no registo dos **Fornecedores** nas várias **Farmacias**. Isto porque, como os **Fornecedores** têm uma localização fixa, então, assim que são criados, informam as **Farmacias** das suas localizações, de modo a que estas possam calcular as distâncias aos **Fornecedores**, consequentemente, saber o custo associado à aquisição de medicamentos e dessa forma saber se vão comprar lá os medicamentos ou se tem outro **Fornecedor** com melhores preços. Relativamente à operação *receber_pedido_farmacia*, esta permite ao **Fornecedor** receber os pedidos de aquisição de **Medicamentos** vindos das **Farmacias** e a operação *enviar_med_fornecedor* é utilizada para que os **Fornecedores** enviem às **Farmacias** os seus pedidos.

Passando agora para a classe **Farmacia** (tipo *Agent*), em que podem existir de 1 a infinitas **Farmacias**, esta contém 5 atributos, em que 2 desses atributos são ArrayList de **Medicamentos** (*stock* e *historico_vendas*), um é a *pos_atual* do tipo **InformaPosicao**, outro é um ArrayList de **Fornecedores** e o último é um objeto do tipo **Relatorio** (*relatorio*). Ao nível das operações, existem 11 operações, sendo uma delas o *registar_farmacia*, em que, da mesma forma que os **Fornecedores** se registam nas **Farmacias**, as **Farmacias** também se registam no **Gestor**, de forma a que este saiba que essas **Farmacias** existem e estão disponíveis para vender os seus medicamentos. A 2º operação é a *receber_registo_fornecedor*, em que as **Farmacias** recebem os registos de cada um dos **Fornecedores**. Mediante os **Fornecedores** presentes no *posicao_fornecedores*, a **Farmacia** vai selecionar o **Fornecedor** mais barato para a

[4] Os nomes das operações pertencentes aos agentes encontram-se em itálico, de modo a facilitar a leitura e compreensão deste documento.

aquisição de medicamentos em falta no stock através da operação *selecionar_fornecedor*, proceder à encomenda dos respetivos medicamentos (*encomendar_fornecedor*) e posteriormente recebê-los (*receber_med_fornecedor*). Através da operação *receber_pedido_gestor*, a **Farmacia** vai receber um pedido feito por um **Cidadao** e que o **Gestor** lhe encaminhou e assim que o pedido estiver pronto reencaminha-o para o **Gestor** (*enviar_compra_gestor*). Para finalizar o processo de compra, a **Farmacia** recebe a confirmação de que o **Cidadao** recebeu o seu pedido (*receber_confirmacao_entrega*). A **Farmacia** possui também uma operação para atualizar o seu stock sempre que adquire ou vende medicamentos (*atualizar_stock*) e uma operação para fazer a análise proativa do stock, prevendo os medicamentos que serão vendidos e a respetiva quantidade (*analise_proativa*). Por último, a **Farmacia** informa a **Interface** da atualização de stock e dos históricos de vendas através do **Relatorio** (*informar_interface*).

Relativamente à classe **Gestor** (tipo *Agent*), verifica-se que existe apenas um único **Gestor** que é um agente centralizado, funcionando como um intermediador entre os pedidos dos **Cidadaos** e a compra dos produtos na **Farmacia**. Nesta classe existem 2 atributos que são um ArrayList de objetos do tipo **InformaPosicao** (*posicao_farmacias*) e um ArrayList de **Farmacias** que contém as *farmacias_selecionadas*. A 1ª operação presente nesta classe é a *receber_registo_farmacia* que permite ao **Gestor** receber os registos de cada nova **Farmacia** e guardá-los na **posicao_farmacias**. Assim, sempre que o **Gestor** receber um pedido de um **Cidadao** (*receber_pedido_cidadao*), este poderá selecionar as **Farmacias** que vendem os **Medicamentos**[5] solicitados pelo **Cidadao** e indicar-lhe qual a mais barata (*responder_pedido*). Após o **Cidadao** selecionar a **Farmacia** em que pretende comprar e informar o **Gestor** da sua escolha (*receber_farmacia_selecionada*), o **Gestor** vai tratar de comprar o pedido à **Farmacia** selecionada (*fazer_compra*), recebê-la (*receber_compra*) e enviar a compra ao **Cidadao** (*enviar_compra*). Por último, o **Gestor** recebe a confirmação do **Cidadao** de que este recebeu a entrega (*receber_confirmacao_entrega*) e transmite essa mensagem à **Farmacia** (*enviar_confirmacao_entrega*).

A classe **Cidadao** (tipo *Agent*), é constituída pelos atributos *medicamentos* que é um ArrayList de objetos do tipo **Medicamento** e *farmacias_selecionadas* que é um ArrayList de **Farmacias**. Esta classe é composta pelas operações *enviar_pedido_gestor*, *selecionar_farmacia*, *receber_compra_gestor* e *enviar_confirmacao_entrega*. A

[5] O uso da palavra Medicamentos não indica que o Cidadao pode adquirir medicamentos diferentes, mas sim, que pode adquirir várias unidades de um mesmo medicamento.

operação *enviar_pedido_gestor* permite ao **Cidadao** enviar o seu pedido de compra ao **Gestor**. Por sua vez, a operação *selecionar_farmacia* é utilizada para que o **Cidadao** escolha, de entre as **Farmacias** apresentadas pelo **Gestor**, aquela onde pretende efetuar a sua compra. De seguida, através da operação *receber_compra_gestor,* o **Cidadao** recebe a compra que pediu ao **Gestor** e ainda lhe envia uma mensagem a confirmar a receção da compra (*enviar_confirmacao_entrega*). É ainda de realçar que podem existir de 1 a infinitos **Cidadaos**.

A classe **Interface** (tipo *Agent*) é constituída por um único agente e por um atributo *relatorio* que é um objeto do tipo **Relatorio**. Esta classe contém também as **operações** *gerir_informacoes* e *gerar_estatisticas*. Estas duas operações permitem ao agente **Interface** receber de cada **Farmacia**, as suas atualizações de stock e informações e históricos de venda e, com isso, imprimir o histórico e stock de cada uma.

## 2.3. Diagrama de Colaboração/Comunicação

De modo a representar o modo de funcionamento do sistema farmacêutico e tendo em vista uma visualização mais simplificada do mesmo, optou-se pela realização de dois diagramas de colaboração/comunicação. Deste modo, é possível representar individualmente cada caso específico que possa ocorrer durante o funcionamento do sistema, o que facilita tanto a conceitualização, como a visualização do processo a decorrer.

Na Figura 3 está representado o 1º diagrama de colaboração/comunicação que diz respeito ao processo de encomenda de **Medicamentos** ao **Fornecedor** por parte da **Farmacia**, o registo desta no **Gestor** e as comunicações com a **Interface**.

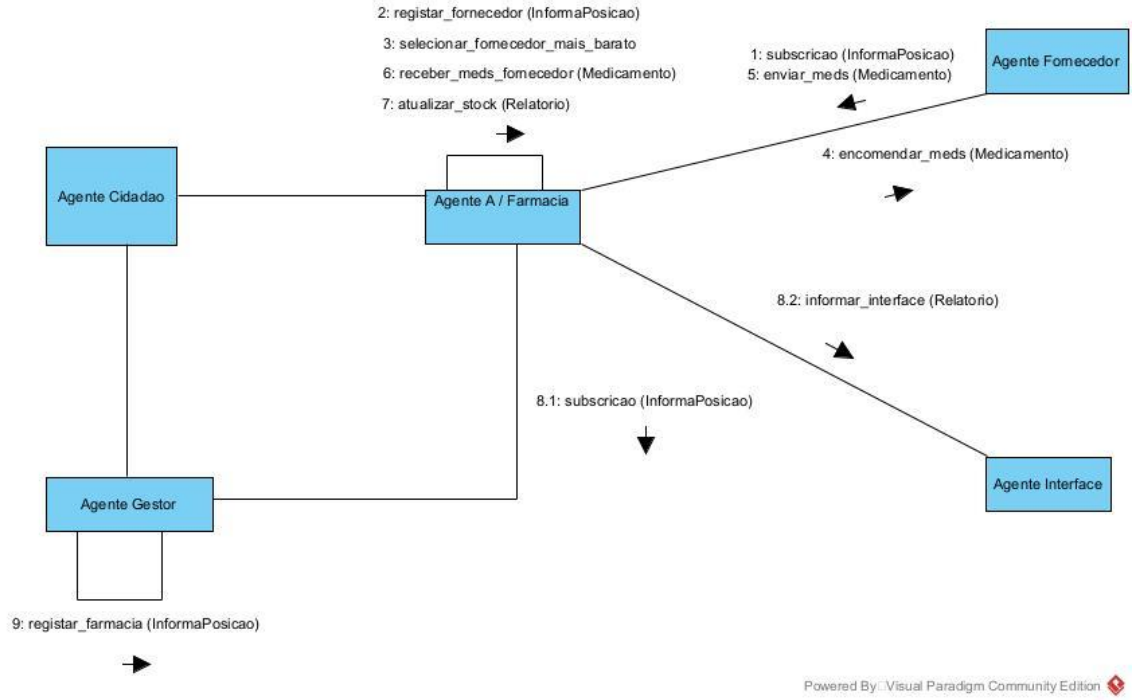



**Figura 3.** Diagrama de colaboração/comunicação relativo ao processo de encomenda de Medicamentos ao Fornecedor por parte da Farmacia, o registo desta no Gestor e as comunicações com a Interface.

Como se pode visualizar na Figura 3, este diagrama contém 10 passos, sendo que dois deles (passo 8.1 e 8.2) são realizados em paralelo. De seguida, segue-se uma explicação para cada passo:

1. O agente **Fornecedor** vai subscrever-se na **Farmacia**, de modo a que esta saiba que esse **Fornecedor** existe e que está disponível para a venda de **Medicamentos**;
2. De seguida, a **Farmacia** recebe o registo do **Fornecedor**, passando a saber onde este está localizado;
3. Conhecendo todos os **Fornecedores**, a **Farmacia** vai selecionar o mais barato para lhe encomendar os **Medicamentos**, inicialmente porque está sem stock, e depois, sempre que tiver baixas de stock;
4. Após ter escolhido o **Fornecedor**, a **Farmacia** vai encomendar-lhe os **Medicamentos** que precisar;
5. Por sua vez, o **Fornecedor** vai enviar para a **Farmacia** os **Medicamentos** que esta pediu;
6. A **Farmacia** vai receber os **Medicamentos** que solicitou ao **Fornecedor**;
7. Depois de receber os **Medicamentos**, a **Farmacia** vai atualizar o seu stock;

8. Assim que tenha finalizado a atualização de stock, a **Farmacia** passa a ser uma **Farmacia** válida e disponível para a compra de **Medicamentos**, pelo que se vai registar no **Gestor** (8.1). Simultaneamente a este processo, a **Farmacia** vai comunicar com a **Interface**, indicando-lhe o seu stock atual (8.2);
9. Por último, o **Gestor** vai receber o registo da **Farmacia**, podendo a partir de agora, encaminhar-lhe pedidos de **Cidadaos**.

O 2º diagrama de colaboração/comunicação, representado na Figura 4, é referente ao processo de compra por parte do **Cidadao**.

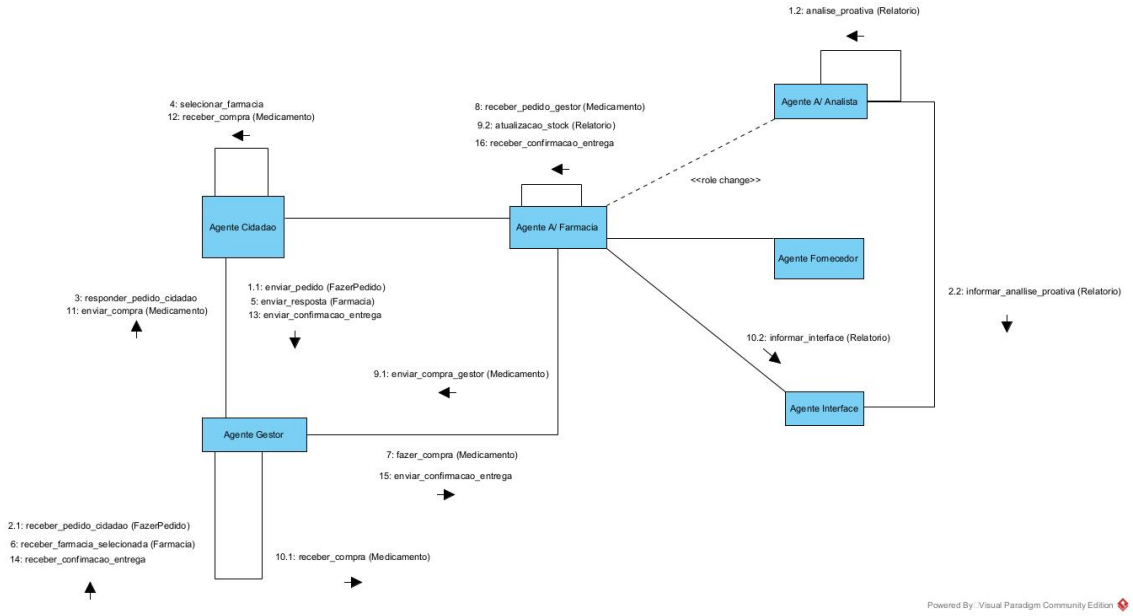



**Figura 4.** Diagrama de colaboração/comunicação relativo ao processo de compra por parte do Cidadao.

Como se pode visualizar na Figura 4, este diagrama contém 20 passos, sendo que oito deles (passos 1.1 e 1.2; 2.1 e 2.2; 9.1 e 9.2; 10.1 e 10.2) são realizados em paralelo. Verifica-se também que o agente A tem 2 funções, atuando como **Farmacia**, permitindo as compras do **Cidadao**, ou como **Analista**, de forma a proceder à análise proativa. De seguida, segue-se uma explicação para cada passo:

1. O **Cidadao** envia o seu pedido de compra ao **Gestor** (1.1), ao mesmo tempo que é inicializada a análise proativa por parte do agente **A /Analista** (1.2);
2. O **Gestor** recebe o pedido enviado pelo **Cidadao** (2.1), ao mesmo tempo que o **Analista** informa a **Interface** da análise proativa realizada (2.2);

3. O **Gestor** responde ao pedido do **Cidadao** e envia-lhe uma lista com todas as **Farmacias** em que o **Cidadao** pode comprar, especificando aquela em que a compra fica mais barata;
4. De seguida, o **Cidadao** recebe a lista enviada pelo **Gestor** e seleciona a **Farmacia** em que quer comprar;
5. O **Cidadao** comunica ao **Gestor** qual a **Farmacia** que escolheu;
6. O **Gestor**, por sua vez, recebe a escolha do **Cidadao**;
7. Sabendo a escolha do **Cidadao**, o **Gestor** pode iniciar o processo de compra, em que vai contactar com a **Farmacia** selecionada e enviar-lhe o pedido do **Cidadao**;
8. A **Farmacia** vai receber o pedido vindo do **Gestor**;
9. Tendo a encomenda pronta, a **Farmacia** vai enviar a compra ao **Gestor** (9.1), ao mesmo tempo que procede à atualização do stock (9.2);
10. O **Gestor** vai então receber a compra que a **Farmacia** lhe enviou (10.1) e simultaneamente, a **Farmacia** vai informar a **Interface** acerca da nova atualização de stock (10.2);
11. Posteriormente, o **Gestor** vai enviar a compra que recebeu da **Farmacia**, para o **Cidadao** que a pediu;
12. O **Cidadao** vai receber a compra;
13. Uma vez que o **Cidadao** já recebeu o seu pedido, vai enviar uma mensagem ao **Gestor**, a confirmar a receção da encomenda;
14. Por sua vez, o **Gestor** vai receber a mensagem com a confirmação de entrega da compra;
15. Sabendo que o **Cidadao** recebeu a compra, o **Gestor** vai transmitir essa mensagem à **Farmacia,** uma vez, que esta não sabe se os produtos que disponibilizou chegaram ao destino;
16. Por último, se tudo correr bem, a **Farmacia** recebe a confirmação de que o **Cidadao** recebeu o seu pedido.

## 2.4. Diagrama de Atividades

À semelhança dos diagramas de colaboração/comunicação, optou-se pela realização de dois diagramas de atividades. No 1º diagrama, apresentado na Figura 5, estão representadas as atividades envolvidas na encomenda de **Medicamentos** ao **Fornecedor** por parte da **Farmacia**, o registo desta no **Gestor** e as comunicações com a **Interface**.

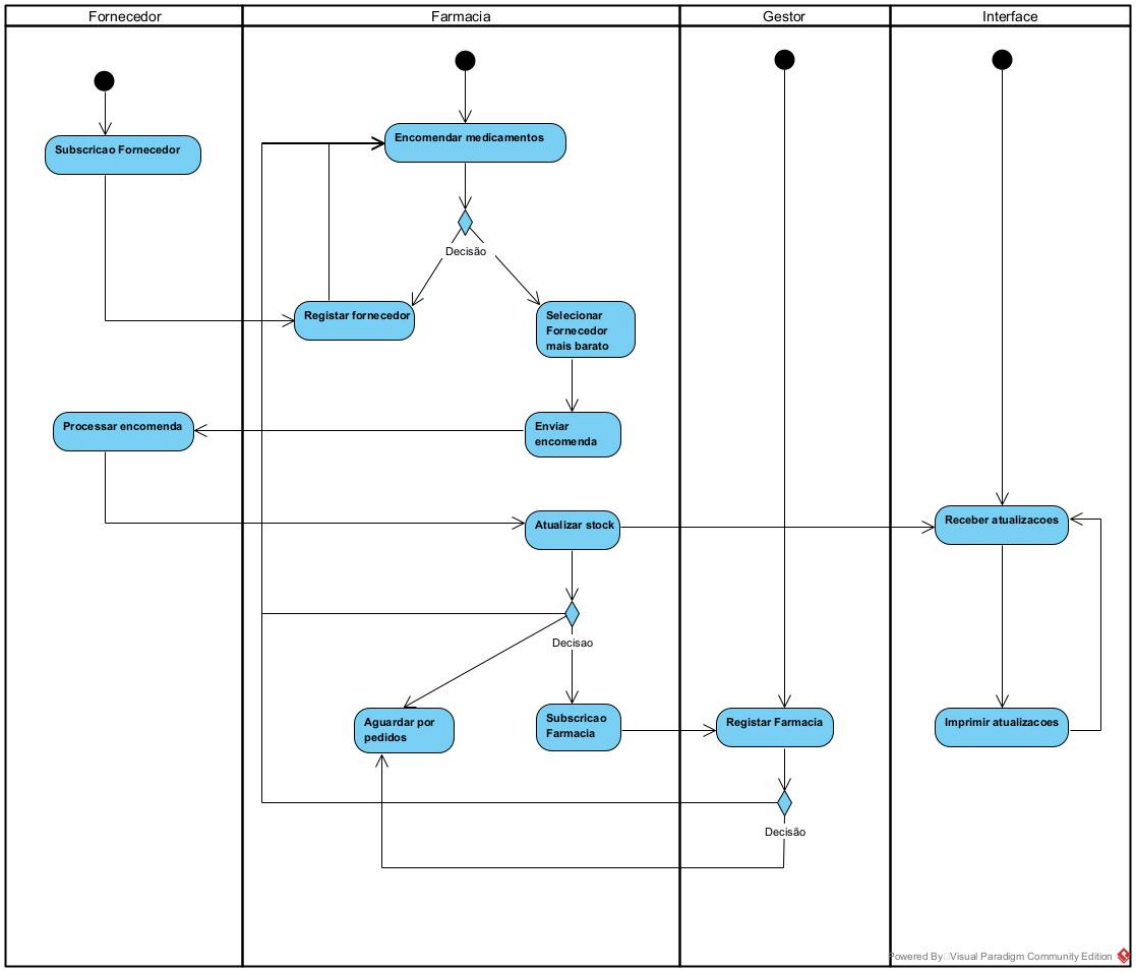



**Figura 5.** Diagrama de atividades relativo ao processo de encomenda de Medicamentos ao Fornecedor por parte da Farmacia, o registo desta no Gestor e as comunicações com a Interface.

Pela análise da Figura 5, verifica-se que a 1ª atividade do agente **Farmacia**, após a sua inicialização, é a encomenda de **Medicamentos** de um determinado **Fornecedor**. Para tal poder ocorrer, tem de existir **Fornecedores** registados na **Farmacia**, logo, se não houverem **Fornecedores** registados, estes vão ter de se registar para poderem estar disponíveis para a venda de **Medicamentos**. Caso os **Fornecedores** já estejam registados, então, a **Farmacia** pode selecionar o **Fornecedor** mais barato e proceder à compra. Por

sua vez, o **Fornecedor** vai processar a encomenda da **Farmacia** e assim que esta a receba vai atualizar o seu stock.

Após a atualização do stock, a **Farmacia** vai informar a **Interface** acerca das atualizações, para que esta as possa imprimir. De modo a introduzir recursividade no diagrama, foi colocada uma ligação do "Imprimir atualizacoes" para o "Receber atualizacoes". Além disso, o agente **Farmacia** vai ter de tomar outra decisão. Se esse agente ainda não estiver registado no **Gestor**, então terá de se registar, caso já esteja registado, poderá aguardar por mais pedidos do **Gestor** ou então, proceder à encomenda de mais **Medicamentos** se tiver o stock em baixo. Estando a **Farmacia** registada, esta terá de decidir se precisa de encomendar mais **Medicamentos** ou se simplesmente vai aguardar por mais pedidos.

O 2º diagrama de atividades, representado na Figura 6, é referente ao processo de compra por parte do **Cidadao**.

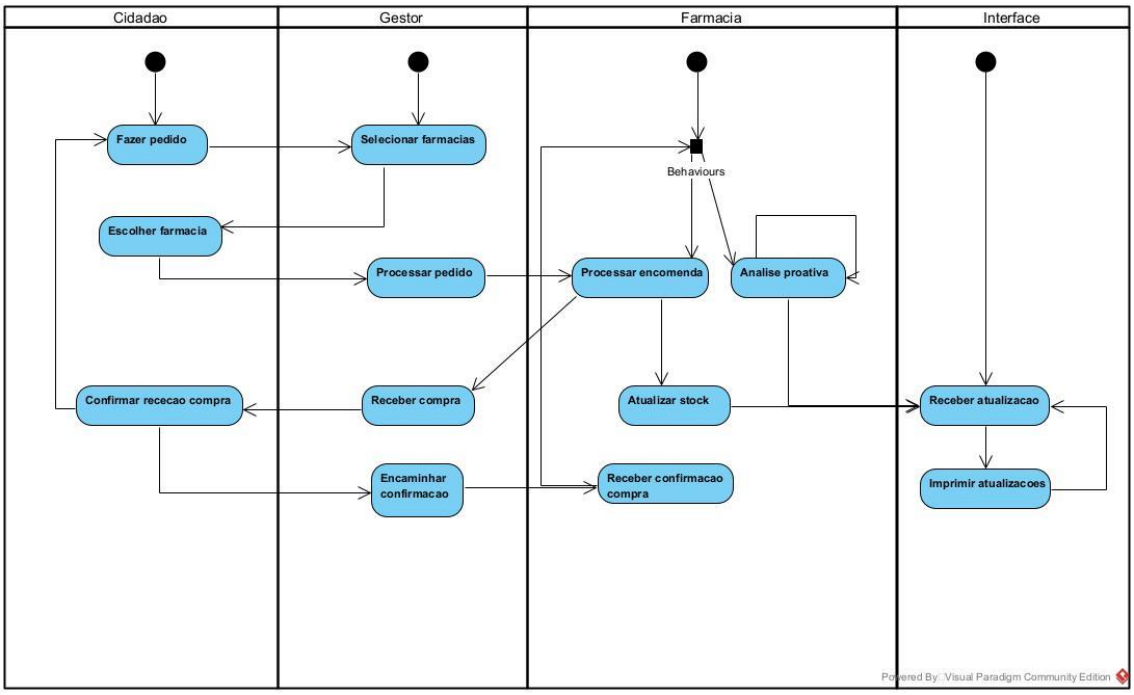



**Figura 6.** Diagrama de atividades relativo ao processo de compra por parte do Cidadao.

Pela análise da Figura 6, verifica-se que a 1ª atividade do agente **Cidadao**, logo após a sua inicialização, é fazer um pedido. Este pedido é enviado ao **Gestor**, que irá selecionar as **Farmacias** onde é possível comprar os **Medicamentos** pedidos pelo **Cidadao**, indicando qual dessas **Farmacias** tem o melhor preço. De seguida, o **Cidadao** vai escolher, de entre as **Farmacias** apresentadas pelo **Gestor**, aquela em que quer comprar. Posteriormente, o **Gestor** vai processar o pedido, encaminhando-o para a

**Farmacia** selecionada. Por sua vez, a **Farmacia** vai processar a encomenda e atualizar o seu stock, comunicando-o à **Interface** para que esta o possa imprimir. Além de atualizar o stock, a **Farmacia** vai também enviar a compra para o **Gestor**, que a vai enviar ao **Cidadao**. Quando a receber, o **Cidadao** vai confirmar a receção ao **Gestor**, para que este comunique essa mensagem à **Farmacia**.

É ainda de realçar, que além de processar encomendas, a **Farmacia** tem ainda outro behaviour, que corresponde à análise proativa e que permite fazer previsões de vendas. Sempre que a **Farmacia** proceder a uma análise proativa, comunica-a à **Interface**, para que possa ser apresentada por esta.

Por último, após comunicar a receção do pedido, o **Cidadao** pode voltar a fazer outros pedidos, da mesma forma que a **Farmacia**, depois de receber a confirmação, também pode voltar a executar um dos seus behaviours.

# 3. Implementação

Nesta subsecção consta uma descrição os vários agentes constituintes deste sistema. Os agentes envolvidos são: **Cidadao, Farmacia, Interface, Fornecedor** e **Gestor.** Para além destes agentes, foram criadas cinco classes: **FazerPedido, InformaPosicao, Medicamento, Posicao** e **Relatorio**. Para visualização da evolução do stock das farmácias, dos históricos de vendas mensal e total e dos lucros mensal e total criou-se também as classes **BarChart_Stock, BarChart_Historico_Mensal, BarChart_Historico_Total, BarChart_Lucro_Mensal** e **BarChart_Lucro_Total.** Por último, a classe **MainContainer** é obviamente o **MainContainer** da Plataforma em causa. Para o desenvolvimento dessas classes e agentes teve-se em conta todo o planeamento acima apresentado.

## 3.1. Classe Posicao

Esta classe foi criada para que os agentes pudessem ter uma determinada posição na matriz que representa o espaço total onde se os agentes se podem localizar. Os atributos são:

- int *X;*
- int *Y.*

Relativamente aos métodos, esta classe possui os *getters* e *setters* para cada um dos atributos anteriormente referidos.

## 3.2. Classe InformaPosicao

Esta classe foi criada para que um determinado agente pudesse informar um outro agente da sua posição. Para isso, possui os seguintes atributos:

- AID *agent;*
- **Posicao** *posicao.*

O atributo *posicao* corresponde a um objeto to tipo **Posicao**. Relativamente aos métodos, esta classe possui os *getters* e *setters* para cada um dos atributos anteriormente referidos.

## 3.3. Classe Medicamento

Esta classe constitui os dados relativos aos medicamentos, sendo utilizada pelos vários agentes para pedirem/receberem medicamentos. Os seus atributos são as caraterísticas do medicamento, ou seja:

- String *nome;*
- int *unidades;*
- double *preco***.**

Relativamente aos métodos, esta classe possui os *getters* e *setters* para cada um dos atributos anteriormente referidos.

## 3.4. Classe FazerPedido

Esta classe constitui os dados necessários para que um **Cidadao** possa fazer um pedido de compra ao **Gestor**. Os seus atributos são:

- AID *agent;*
- **Posicao** *pos;*
- **Medicamento** *med***.**

O atributo *pos* corresponde a um objeto to tipo **Posicao** e o atributo *med* corresponde a um objeto do tipo **Medicamento**. Relativamente aos métodos, esta classe possui os *getters* e *setters* para cada um dos atributos anteriormente referidos.

## 3.5. Classe Relatorio

A classe **Relatorio** foi concebida para que a farmácia pudesse enviar um relatório das informações e atualizações à **Interface**, cujo conteúdo seriam os seguintes atributos:

- O AID do agente **Farmacia** (AID *agent*);
- O ArrayList do stock de medicamentos (ArrayList<Medicamento> *stock*);
- O HashMap do histórico de vendas mensal (HashMap<Medicamento, Integer> *historico_vendas_mensal*);
- O HashMap do histórico de vendas total (HashMap<Medicamento, Integer> *historico_vendas_total*);
- O HashMap do lucro de vendas mensal (HashMap<Medicamento, Double> *lucro_vendas_mensal*);
- E ainda o HashMap do lucro de vendas total (HashMap<Medicamento, Double> *lucro_vendas_total*).

Relativamente aos métodos, esta classe possui os *getters* e *setters* para cada um dos atributos anteriormente referidos, bem como um construtor que tem como argumento todos os atributos.

## 3.6. Agente Farmacia

O agente **Farmacia** é responsável por informar o **Gestor** acerca dos seus **Medicamentos** em stock e sempre que for escolhido pelo **Gestor** (em nome do **Cidadao**) para vender **Medicamentos**, tem obrigatoriamente de os vender. Esta atitude por parte da **Farmacia** visa imitar a atitude dos vendedores de uma farmácia real, que nunca recusam a venda de **Medicamentos** a clientes, pois, visam aumentar os seus lucros. Este agente é ainda responsável por atualizar os stocks, históricos de vendas e de lucros e por enviar os **Relatorios** à **Interface**. Os atributos deste agente são:

- Objeto do tipo **Posicao** com a posição do agente **Farmacia** (**Posicao** *posicao_farmacia*);
- Objeto do tipo **InformaPosicao** com os dados do agente **Farmacia** (**InformaPosicao** *dados_farmacia*);
- O ArrayList do stock de medicamentos (ArrayList<Medicamento> *stock*);
- O HashMap do histórico de vendas mensal (HashMap<Medicamento, Integer> *historico_vendas_mensal*);
- O HashMap do histórico de vendas total (HashMap<Medicamento, Integer> *historico_vendas_total*);
- O HashMap do lucro de vendas mensal (HashMap<Medicamento, Double> *lucro_vendas_mensal*);
- O HashMap do lucro de vendas total (HashMap<Medicamento, Double> *lucro_vendas_total*);
- O ArrayList de objetos do tipo **InformaPosicao** com os dados dos agentes **Fornecedor** (**InformaPosicao** *pos_fornecedores*);
- O AID do agente **Fornecedor,** cuja distância à **Farmacia** é a menor (AID *agent*);
- O número de **Medicamentos** que cada **Cidadao** pede no seu pedido (int *num_meds*);
- O nome do **Medicamento** que cada **Cidadao** pede no seu pedido (String *nome_med*);
- O HashMap que contém o nome do **Medicamento** que foi comprado e respetiva quantidade (HashMap<String, Integer> *medicamentos*).

No *setup()* cada **Farmacia** é inicializada. Primeiramente são inicializadas as variáveis globais existentes no agente. De seguida são escolhidas 2 coordenadas *X* e *Y*, respetivamente, podendo ser qualquer número inteiro de 0 a 100 que correspondem à **Posicao** da **Farmacia**. Posteriormente, são criados 13 **Medicamentos**, dos quais a **Farmacia** escolhe aleatoriamente 10 para constituírem o seu stock. Logo depois são inicializados os históricos de vendas e de lucros, colocando as unidades vendidas e os lucros, respetivamente, a 0. Por último, a **Farmacia** regista-se nas Páginas Amarelas e são apresentados os behaviours da **Farmacia**.

Para este agente, foram definidos quatro behaviours que correspondem ao behaviour *Registar_farmacia*[6], behaviour *InformarInterface*, behaviour *Receber_msg* e behaviour *RestockProativo*.

Começando pelo *Registar_farmacia*, neste behaviour, todas as **Farmacias** registam-se no **Gestor**, cujo type do Service Description é "central". Para o registo, cada **Farmacia** envia uma mensagem ACL, cuja performative é SUBSCRIBE, para o **Gestor** e cujo conteúdo é o objeto *dados_farmacia*, um objeto do tipo **InformaPosicao**. Como cada **Farmacia** se regista apenas uma única vez, então este é um OneShotBehaviour.

O behaviour *InformarInterface* também é um OneShotBehaviour, pois corresponde ao **Relatorio** inicial enviado para a **Interface**, logo depois da inicialização da **Farmacia**. Para que a Interface tenha acesso ao **Relatorio**, é enviada uma mensagem ACL cuja performative é INFORM e cujo conteúdo é um objeto do tipo **Relatorio**.

O behaviour *Receber_msg* é um CyclicBehaviour, uma vez que a **Farmacia** está sempre à espera de receber mensagens vindos dos diferentes agentes. A ação que o agente **Farmacia** vai executar depende do tipo de performative da mensagem recebida. Este behaviour está preparado para receber quatro tipos de performative: SUBSCRIBE, REQUEST, PROPOSE e CONFIRM.

Quando recebe uma mensagem de um **Fornecedor** do tipo SUBSCRIBE, cujo conteúdo é um objeto do tipo **InformaPosicao**, a **Farmacia** regista esse **Fornecedor** no ArrayList *pos_fornecedores*. De seguida, para cada uma das posições desse Array, o agente **Farmacia** determina qual das distâncias entre o respetivo **Fornecedor** e a **Farmacia** é a menor, obtendo-se assim o AID do **Fornecedor** mais próximo (*closestFor*). Isto é necessário, porque, mais uma vez, tentando aproximar os comportamentos da **Farmacia** aos dos vendedores de uma **Farmacia** real, a o agente **Farmacia** vai escolher sempre comprar no **Fornecedor** mais barato, que neste caso, é o **Fornecedor** mais próximo, pois desprezaram-se os custos associados ao medicamento em si.

A **Farmacia** recebe também mensagens ACL do **Gestor** do tipo REQUEST, em que este pretende averiguar se a **Farmacia** tem o **Medicamento** que o **Cidadao** quer comprar, na quantidade necessária. Para isso, o conteúdo da mensagem recebida é um objeto do tipo **Medicamento**. Sempre que recebe essa mensagem, a **Farmacia** vai averiguar se tem o **Medicamento** na quantidade pedida em stock. Se tiver, envia uma mensagem ACL, cuja performative é INFORM., ao **Gestor** a informar que tem o

[6] Os behaviours encontram-se realçados a itálico.

**Medicamento**, respondendo-lhe com o **Medicamento** e respetivo preço. Se tiver o **Medicamento** em stock, mas não em quantidade suficiente, então a **Farmacia** envia uma mensagem ACL do tipo REFUSE ao **Gestor**, cujo conteúdo é o respetivo **Medicamento**. Simultaneamente, envia uma mensagem ACL do tipo REQUEST_WHEN ao **Fornecedor**, cujo conteúdo são as unidades de **Medicamento** necessárias para que a Farmacia possa ter em stock a quantidade solicitada. Caso a **Farmacia** não tenha o **Medicamento** no seu stock, então, simplesmente envia uma mensagem ACL do tipo REFUSE ao **Gestor** e cujo conteúdo é o respetivo **Medicamento**.

Posteriormente, se a **Farmacia** for escolhida pelo **Cidadao** para efetuar a compra, então, o **Gestor** envia-lhe uma mensagem do tipo PROPOSE, cujo conteúdo é um objeto do tipo **Medicamento**. Nesta mensagem, o Gestor vai propor a compra e a **Farmacia** obviamente aceita a proposta, enviando uma mensagem do tipo ACCEPT_PROPOSE ao **Gestor** e cujo conteúdo é o **Medicamento** que o **Cidadao** pretende comprar. Logo de seguida, a **Farmacia** procede às atualizações do stock, históricos de vendas e de lucros, reduzindo no stock as unidades que vendeu, aumentando nos históricos de vendas mensal e total a quantidade de **Medicamentos** vendidos nesse pedido e aumentando também o respetivo lucro mensal e total. É de realçar que foi necessária a utilização do HashMap *medicamentos*, pois, uma mesma **Farmacia** pode vender vários **Medicamentos** e por isso, a utilização da variável *num_meds* tornava-se insuficiente, pois, iria acumular a soma das quantidades de todos os **Medicamentos** adquiridos, sem distinguir de que tipo eram.

Por último, a **Farmacia** recebe uma mensagem ACL do tipo CONFIRM do **Gestor**, a confirmar que o **Cidadao** recebeu o pedido.

Sendo o agente **Farmacia** um agente proativo, isto é, tem a capacidade de avaliar o stock de **Medicamentos** e, com base na análise do histórico de vendas mensal, prever as necessidades de compra dos **Cidadaos**, encomendando, ou não, mais produtos ao **Fornecedor**. Esta análise proativa é realizada através do TickerBehaviour *RestockProativo* que atua a cada 1000ms. Assim, ao fim de cada mês, caso o stock de um produto seja inferior ao seu histórico de vendas no último mês, é realizada uma encomenda ao **Fornecedor**, recorrendo a uma mensagem ACL com performative REQUEST. A encomenda consiste na diferença entre a quantidade dos produtos que foram vendidos no último mês e a quantidade existente em stock. Assim, é assegurado um stock mínimo para o mês seguinte que corresponde ao histórico de vendas do mês anterior. De seguida, é enviado um **Relatorio** das atualizações à **Interface** através de uma mensagem ACL do tipo INFORM. Finalmente, é feito um reset às variáveis mensais.

## 3.7. Agente Gestor

O agente **Gestor** atua como o intermediário entre as **Farmacias** e os **Cidadaos**, transmitindo informações entre ambos. O seu objetivo principal é facilitar o processo de compra dos **Cidadaos**, em que estes apenas lhe dizem qual o **Medicamento** que pretende comprar e o **Gestor** envia-lhes uma lista com todas as **Farmacias** em que o seu pedido se encontra disponível e o respetivo preço. Os atributos deste agente são:

- Objeto do tipo **Posicao** com a posição do agente **Gestor** (**Posicao** *posicao_gestor*);
- Objeto do tipo **Posicao** com a posição do agente **Cidadao** que fez um determinado pedido (**Posicao** *posicao_cidadao*);
- Objeto do tipo **FazerPedido** com os dados do pedido de um agente **Cidadao** (**FazerPedido** *pedido_cidadao*);
- O HashMap dos pedidos dos **Cidadaos**, cuja chave é o AID do **Cidadao** e o valor é o respetivo pedido (HashMap<AID, **FazerPedido**> *pedidos_cidadaos*);
- O ArrayList de objetos do tipo **InformaPosicao** com os dados dos agentes **Farmacia** (**InformaPosicao** *pos_farmacias*);
- O HashMap das **Farmacias** disponíveis para a compra do pedido do **Cidadao** e respetivo preço dos **Medicamentos** (HashMap<AID, Double> *farmacias_disponiveis*);
- O HashMap das **Farmacias** selecionadas para a compra do pedido do **Cidadao** e respetivo preço total do pedido (HashMap<AID, Double> *farmacias_selecionadas*);
- O AID do agente **Farmacia,** que foi escolhido pelo **Cidadao** para lhe vender os **Medicamentos** solicitados (AID *farmacia_escolhida*);
- O número de **Farmacias** indisponíveis para a venda do pedido (int *farmacias_nao_selecionadas*);
- O HashMap do par **Cidadao** – **Farmacia** associado a cada pedido (HashMap<AID, AID> *cidadao_farmacia*).

No *setup()* o **Gestor** é inicializado. Primeiramente são inicializadas as variáveis globais existentes no agente. De seguida são escolhidas 2 coordenadas *X* e *Y*, respetivamente, podendo ser qualquer número inteiro de 0 a 100, que correspondem à **Posicao** do **Gestor**. Posteriormente, o **Gestor** regista-se nas Páginas Amarelas e por último, é apresentado o seu behaviour *Receber_msg*.

O behaviour *Receber_msg* é um CyclicBehaviour, uma vez que o **Gestor** está sempre à espera de receber mensagens vindos dos diferentes agentes. A ação que o agente **Gestor** vai executar depende do tipo de performative da mensagem recebida. Este behaviour está preparado para receber sete tipos de performative: SUBSCRIBE, REQUEST, REFUSE, INFORM, AGREE, ACCEPT_PROPOSE e CONFIRM.

Quando recebe uma mensagem de uma **Farmacia** do tipo SUBSCRIBE, cujo conteúdo é um objeto do tipo **InformaPosicao**, o **Gestor** regista essa **Farmacia** no ArrayList *pos_farmacias*, passando a ter acesso à sua posição e AID.

O **Gestor** recebe também mensagens ACL do **Cidadao** do tipo REQUEST, em que o **Cidadao** pretende que o **Gestor** lhe informe sobre quais as **Farmacias** que têm disponível o pedido do **Cidadao**. Para isso, o **Cidadao** envia a mensagem cujo conteúdo é um objeto do tipo **FazerPedido**. Ao receber este tipo de mensagens, o **Gestor** envia uma mensagem ACL, com performative REQUEST, a cada uma das **Farmacias**, cujo AID se encontra no ArrayList *pos_farmacias*, de forma a averiguar quais as **Farmacias** têm o pedido. É de realçar que cada pedido é adicionado ao HashMap *pedidos_cidadaos*, para ser possível aceder às informações relativas aos pedidos atuais do **Gestor.**

Posteriormente, o **Gestor** vai receber 2 tipos de performatives de mensagens ACL das **Farmacias**. Se uma determinada **Farmacia** não tiver o pedido, então o **Gestor** vai receber uma mensagem do tipo REFUSE e a variável farmacias_nao_selecionadas aumenta 1 unidade. Se, por outro lado, uma **Farmacia** tiver o pedido, então o **Gestor** vai receber uma mensagem do tipo INFORM e essa **Farmacia** é adicionada ao HashMap *farmacias_disponiveis.* Assim que o número de **Farmacias** disponíveis e de **Farmacias** não selecionadas for igual, o **Gestor** calcula o preço total do pedido que corresponde a uma soma do custo de transportar o pedido da **Farmacia** ao **Gestor** (0.03 UM por cada unidade de distância) mais o custo de transportar o pedido do **Gestor** ao **Cidadao** (0.02 UM por cada unidade de distância) mais o custo do **Medicamento** em si, multiplicado pelas unidades pedidas. Cada um destes custos, assim como o AID da respetiva **Farmacia** associada, são adicionados ao HashMap *farmacias_selecionadas*, que depois é enviado

numa mensagem ACL do tipo PROPOSE ao **Cidadao**, para que este possa escolher em que **Farmacia** deseja comprar.

De seguida, o **Gestor** recebe uma mensagem ACL do **Cidadao**, do tipo AGREE, a informar-lhe em qual das **Farmacias** deseja efetuar a compra. Para isso, o conteúdo da mensagem enviada pelo **Cidadao** é o AID da **Farmacia** escolhida. Assim que tem essa informação, o **Gestor** envia uma mensagem ACL do tipo PROPOSE à **Farmacia** escolhida e cujo conteúdo é o **Medicamento** solicitado pelo **Cidadao**.

O **Gestor** recebe também mensagens ACL, cuja performative é ACCEPT_PROPOSAL, da **Farmacia** escolhida pelo **Cidadao**, e cujo conteúdo é o **Medicamento** solicitado.

Por último, o **Gestor** recebe uma mensagem do **Cidadao**, cuja performative é CONFIRM a confirmar a receção do pedido e logo de seguida, envia uma mensagem ACL também com performative CONFIRM, para confirmar à **Farmacia** que o pedido chegou corretamente ao **Cidadao**.

## 3.8. Agente Cidadao

O agente **Cidadao** atua como um cliente das **Farmacias**, que não contacta diretamente com estas. Em vez disso, este envia os seus pedidos ao **Gestor**, que trata de agilizar todo o processo de compra. Os atributos deste agente são:

- Objeto do tipo **Posicao** com a posição do agente **Cidadao** que fez um determinado pedido (**Posicao** *posicao_cidadao*);
- Um ArrayList de Strings que contém os nomes de todos os **Medicamentos** que cada **Cidadao** pode escolher adquirir (ArrayList<String> *nome_meds*);
- O nome do **Medicamento** escolhido aleatoriamente para o **Cidadao** comprar periodicamente (String *random_medicamento*);
- O número de unidades do **Medicamento** escolhido aleatoriamente para o **Cidadao** comprar periodicamente (int *uni_meds*).

No *setup()* o **Cidadao** é inicializado. Primeiramente são inicializadas as variáveis globais existentes no agente. De seguida são escolhidas 2 coordenadas *X* e *Y*, respetivamente, podendo ser qualquer número inteiro de 0 a 100, que correspondem à

**Posicao** do **Cidadao**. Posteriormente, o **Cidadao** escolhe aleatoriamente um nome de um Medicamento existente no ArrayList *nome_meds*, assim como uma quantidade aleatória entre 1 a 6 unidades, sendo esse valor o *uni_meds*. Por último, são apresentados os seus behaviours *Enviar_pedido_gestor* e *Receber_msg*.

No behaviour *Enviar_pedido*_gestor, o **Cidadao** envia pedidos, sempre iguais ao Gestor a cada 800m ms. O pedido consiste num **Medicamento** cujo nome é o *random_medicamento* e cujas unidades são *uni_meds*, ambos definidos no *setup()* do agente. É de realçar que apesar da classe **Medicamento** apresentar também preço, o **Cidadao** envia sempre pedidos cujo preço é zero, não porque esse seja o preço real do **Medicamento**, mas porque, num contexto real, um **Cidadao**, para efetuar uma compra não tem obrigatoriamente de saber o preço daquilo que quer comprar, muito menos especificar o preço da sua compra. Após formando o pedido, este é enviado numa mensagem ACL ao **Gestor**, cuja performative é REQUEST, de forma a averiguar quais as **Farmacias** que têm esse pedido disponível para venda e o respetivo preço total do pedido.

O behaviour *Receber_msg* é um CyclicBehaviour, uma vez que o **Cidadao** está sempre à espera de receber mensagens do **Gestor**. A ação que o agente **Cidadao** vai executar depende do tipo de performative da mensagem recebida. Este behaviour está preparado para receber dois tipos de performative: PROPOSE e INFORM.

O **Cidadao** recebe mensagens ACL do tipo PROPOSE, cujo conteúdo é um HashMap que contém como chaves os AIDs das **Farmacias** que têm o pedido do **Cidadao** e como valores os respetivos preços do pedido. Se apenas existir uma **Farmacia** nesse HashMap, então, obviamente, o **Cidadao** efetua a compra nessa **Farmacia**, caso contrário, o **Cidadao** seleciona uma **Farmacia** para comprar. Inicialmente, o **Cidadao** determina qual a **Farmacia** em que o custo total é o menor e adiciona esta **Farmacia** a um ArrayList *lista_farmacias_random*. De seguida, coloca as restantes **Farmacias** num outro ArrayList *lista_farmacias* e desta escolhe aleatoriamente uma **Farmacia**, adicionando a escolhida à *lista_farmacias_random*. Posteriormente, escolhe 1 das 2 **Farmacias** existentes na *lista_farmacias_random* para efetuar a compra. Com isto, pretende-se por um lado dar liberdade ao **Cidadao** de escolher uma **Farmacia** que não seja a mais barata, e por outro lado, fazer com que haja 50% de probabilidade de escolher a mais barata. O motivo pelo qual isto foi efetuado, foi para assemelhar o comportamento do agente **Cidadao** ao de um **Cidadao** real, que estaria provavelmente ficaria indeciso

entre comprar na sua **Farmacia** de confiança (que não seria a mais barata) ou comprar na **Farmacia** mais barata. Após escolhida a **Farmacia**, o AID desta é enviada numa mensagem ACL ao **Gestor** e cujo performative é AGREE.

Por último, o **Cidadao** recebe também mensagens do **Gestor**, cuja performative é INFORM e cujo conteúdo é o seu pedido. Assim que tenha o seu pedido, envia uma mensagem ACL com performative CONFIRM ao **Gestor**, a confirmar a receção do pedido.

## 3.9. Agente Fornecedor

O agente **Fornecedor** fornece Medicamentos às Farmacias sempre que estas solicitam. Este atributo possui apenas um único atributo:

- Objeto do tipo **InformaPosicao** com os dados relativos ao agente **Fornecedor** (**InformaPosicao** *pos_atual*).

No *setup()* o **Fornecedor** é inicializado. Primeiramente são inicializadas as variáveis globais existentes no agente. Por último, são apresentados os seus behaviours *Registar_fornecedor* e *Receber_e_responder_farmacia*.

No behaviour *Registar_fornecedor*, todos os **Fornecedores** registam-se em todas as **Farmacias**, cujo type do Service Description é "farmacia". Para o registo, cada **Fornecedor** envia uma mensagem ACL, cuja performative é SUBSCRIBE para cada **Farmacia** e cujo conteúdo é o objeto *pos_atual*. Como cada **Fornecedor** se regista apenas uma única vez, então este é um OneShotBehaviour.

O behaviour *Receber_e_responder_farmacia* é um CyclicBehaviour, uma vez que o **Fornecedor** está sempre à espera de receber mensagens da **Farmacia**. A ação que o agente **Fornecedor** vai executar depende do tipo de performative da mensagem recebida. Este behaviour está preparado para receber dois tipos de performative: REQUEST e REQUEST_WHEN.

Sempre que o **Fornecedor** recebe uma mensagem ACL cuja performative é REQUEST_WHEN, esta mensagem é proveniente de uma **Farmacia** que precisa de aumentar o stock de um determinado **Medicamento**, uma vez que esta **Farmacia** recebeu um pedido do **Gestor** e não tinha quantidade suficiente do **Medicamento** para atender ao pedido. Recebendo esta mensagem, cujo conteúdo é o **Medicamento** em falta, e sendo os

**Fornecedores** entidades que tem stock ilimitado, então, a **Farmacia** restabelece automaticamente o seu stock.

O **Fornecedor** recebe também mensagens ACL cuja performative é REQUEST, sendo estas mensagens provenientes de **Farmacias** que precisam de fazer o restock proativo de alguns **Medicamentos**. Deste modo, o conteúdo da mensagem é um ArrayList de **Medicamentos** que contém os **Medicamentos** a repor na **Farmacia**. Mais uma vez, o stock é reposto automaticamente na **Farmacia**.

## 3.10. Agente Interface

O agente **Interface** tem como objetivo receber **Relatorios** de cada **Farmacia** existente com atualizações de stock de **Medicamentos** e múltiplas estatísticas relativas ao seu funcionamento. Este agente é caracterizado por um atributo e um método.

O atributo é:

- *estatistica*, ou seja, um HashMap composto pelo nome local de uma **Farmacia** e o **Relatorio** que está a enviar (HashMap<String, **Relatorio**> *estatistica*).

O método é apenas um *setter*:

- *setEstatistica(*HashMap<String, **Relatorio**> *estatistica)*: permite definir o HashMap estatística que contém a **Farmacia** e o seu respetivo **Relatorio**.

Posto isto, a **Interface** apresenta dois behaviours, o behaviour *Receber* e o behaviour *MostrarEstatisticas*.

O primeiro behaviour, *Receber*, consiste num CyclicBehaviour, visto que a **Interface** deve estar sempre disponível para receber os **Relatorios** das **Farmacias**. Estes são recebidos através de mensagens ACL com a *performative* INFORM. Desta forma, recorrendo ao conteúdo dos **Relatorios**, a **Interface** realiza e imprime as estatísticas. Para tal, é necessário colocar no HashMap *estatistica* tanto a **Farmacia** que enviou o **Relatorio**, como o **Relatorio** recebido.

O segundo behaviour, *MostrarEstatisticas*, corresponde a um TickerBehaviour que se repete a cada 1300ms, apresentando as estatísticas relativas ao mês anterior, que corresponde a um período de 1000ms. Este comportamento permite que a **Interface** mostre as estatísticas definidas que consistem em:

- **Farmacia** com maior número de vendas no último mês;
- **Farmacia** com maior lucro no último mês;
- Histórico de vendas mensal por **Farmacia**;
- Histórico de vendas total por **Farmacia**;
- Histórico de lucro mensal por **Farmacia**;
- Histórico de lucro total por **Farmacia**;
- Os três produtos mais vendidos por cada **Farmacia**;
- Stock das **Farmacias**.

Para visualizar, ou seja, imprimir a **Farmacia** com maior número de vendas, é necessário percorrer os **Relatorios** das **Farmacias** e fazer a contagem de cada **Medicamento** somando os valores obtidos. Posto isto, comparam-se os valores de **Medicamentos** vendidos por cada **Farmacia** e imprime-se a **Farmacia** que vendeu a maior quantidade de **Medicamentos**, bem como a respetiva quantidade. O mesmo raciocínio é aplicado para imprimir a **Farmacia** com maior lucro mensal, com a diferença que em vez de se somarem as quantidades, somam-se os lucros.

Relativamente ao histórico de vendas mensal por **Farmacia**, apenas foi necessário percorrer as **Farmacias** que enviaram relatórios e verificar o HashMap *historico_vendas_mensal*, imprimindo o mesmo para que se possa visualizar a informação. O mesmo acontece para a estatística histórico de vendas total por **Farmacia**, imprimindo-se o *historico_vendas_total*, para a estatística lucro de vendas mensal por **Farmacia**, imprimindo-se o *lucro_vendas_mensal* e para a estatística lucro de vendas total por **Farmacia**, imprimindo-se o *lucro _vendas_total*. De forma a visualizar as flutuações destas estatísticas ao fim de cada mês, para cada **Farmacia** imprimiu-se a respetiva estatística e foi exibido um gráfico de barras com a informação da estatística de cada **Farmacia**. Para isso, recorreu-se ao *JFreeChart* na classe **BarChart_Stock,** em que através da estatística presente em cada **Relatorio** enviado por cada **Farmacia**, adiciona-se o valor da estatística, o nome do **Medicamento** e a respetiva **Farmacia** ao gráfico.

Para obter os três **Medicamentos** mais vendidos por cada **Farmacia**, foi necessário observar o *historico_vendas_mensal* de cada uma e verificar quais os três **Medicamentos** que apresentavam as quantidades de venda mais elevadas, imprimindo os **Medicamentos** e as quantidades obtidas.

Por último, escolheram-se duas maneiras de visualizar as flutuações de stock ao fim de cada mês. Em primeiro lugar, para cada **Farmacia** imprimiu-se o stock de

**Medicamentos**. Em segundo lugar, é exibido um gráfico de barras com a informação do stock de cada **Farmacia**. Para isso, recorreu-se ao *JFreeChart* na classe **BarChart_Stock**, em que através do stock de **Medicamentos** de cada **Relatorio** enviado por cada **Farmacia**, adiciona-se a quantidade do **Medicamento**, o nome do **Medicamento** e a respetiva **Farmacia** ao gráfico.

## 3.11. Classes BarChart

A classe **BarChart_Stock** permite a visualização dos stocks de **Medicamentos** das várias **Farmacias** inicializadas no sistema através de gráficos de barras. Assim, estes gráficos apresentam o **Medicamento** e o número de unidades em stock para cada **Farmacia**. O mesmo raciocínio e forma de construção foi aplicado para construir as classes: **BarChart_Historico_Mensal, BarChart_Historico_Total, BarChart_Lucro_Mensal** e **BarChart_Lucro_Total**

Estes gráficos são bastante interessantes, pois permitem visualizar de forma mais eficiente as alterações nas diferentes estatísticas que ocorrem nas **Farmacias** ao fim de cada mês.

## 3.12. MainContainer

O **MainContainer** permite a inicialização de todos os agentes do sistema, sendo constituído por uma main que permite colocar os agentes em execução.

# 4. Demonstrações

Nesta secção serão demonstrados os outputs da consola, bem como alguns resultados estatísticos que o sistema imprime.

Em primeiro lugar, são inicializados o **Gestor** e a **Interface** (Figura 7).

```
Starting Gestor.
******************************

Starting Interface.
******************************
```

**Figura 7.** Inicialização do Gestor e da Interface.

De seguida, foram inicializadas duas **Farmacias, Fornecedores** e **Cidadaos** (Figuras 8 a 10). Além de inicializar os agentes, os **Fornecedores** registaram-se nas **Farmacias** e as **Farmacias** no **Gestor**. O número de agentes selecionados foi apenas um número simbólico utilizado para teste. Contudo, o sistema desenvolvido neste trabalho foi também testado com 999999999 **Farmacias**, Fornecedores e **Cidadaos** (uma vez que esse é o maior número int que existe) e verificou-se que o sistema continuava igualmente a funcionar, embora fosse necessário no **MainContainer** aumentar a thread para os agentes terem mais tempo de se registarem. Além disso, torna-se mais difícil a leitura dos resultados da consola.

É ainda de realçar que a apresentação dos 5 gráficos desenvolvidos desacelera a impressão de estatísticas consideravelmente, de tal modo que o **Relatorio** Mensal aparecia em partes separadas na consola dificultando a leitura. Deste modo, optou-se por apenas apresentar o gráfico relativo aos stocks. Contudo, basta comentar uns e descomentar outros, para se ter acesso aos gráficos que se pretender.

```
Starting Farmacia0:
        - Stock: [Medicamento [nome=brufen, unidades=10, preco=1.7], Medicamento [nome=benuron, unidades=10, pre
        - Coordenadas: x = 52 y = 10
------------------------------------------------------------------------------------------------------------
Starting Farmacia1:
        - Stock: [Medicamento [nome=benuron, unidades=10, preco=2.0], Medicamento [nome=aspirina, unidades=10, p
        - Coordenadas: x = 32 y = 58
------------------------------------------------------------------------------------------------------------
Gestor: Farmacia0 registada!
Gestor: Farmacia1 registada!
************************************************************************************************************
```

**Figura 8.** Inicialização das Farmacias.

```
Starting Fornecedor.
**********************************************
Starting Fornecedor.
Farmacia0: Fornecedor1 fornecedor registado!
Farmacia1: Fornecedor1 fornecedor registado!
Farmacia1: Fornecedor0 fornecedor registado!
Farmacia0: Fornecedor0 fornecedor registado!
```

**Figura 9.** Inicialização dos Fornecedores.

```
Starting Cidadao1.
        -Coordenadas: x = 40, y = 94.
--------------------------------------
Starting Cidadao0.
        -Coordenadas: x = 83, y = 88.
--------------------------------------
```

**Figura 10.** Inicialização dos Cidadaos.

Após aparecer o 1º **Relatorio** Mensal, com todas as variáveis a 0 (Figura 11), pois ainda nada foi vendido, surge o 1º pedido dos **Cidadaos** 1 e 0, que pretendem comprar 4 unidades de voltaren e 5 unidades de xanax. O **Gestor** recebe os pedidos dos **Cidadaos** e comunica-os a cada uma das **Farmacias**. Verificou-se que ambas as **Farmacias** têm ambos os pedidos dos **Cidadaos**. Neste exemplo, os 2 **Cidadaos** decidiram efetuar a compra na **Farmacia** 0. Esta envia esses pedidos ao **Gestor**, que os transmite ao respetivo **Cidadao**. Estes por sua vez confirmam ao **Gestor** que receberam o pedido e o **Gestor** confirma essa informação à **Farmacia** 0. Todo este processo está representado na Figura 12.

```
**********************************************************************************************************************************************************
Interface: RELATÓRIO MENSAL.

FARMÁCIA COM MAIOR NÚMERO DE VENDAS NO ÚLTIMO MÊS
Farmacia0=0 unidades

FARMÁCIA COM MAIOR LUCRO NO ÚLTIMO MÊS
Farmacia0=0.0 unidades monetárias

HISTÓRICO DE VENDAS MENSAL POR FARMÁCIA
Farmacia0: {Medicamento [nome=brufen, unidades=25, preco=1.7]=0, Medicamento [nome=fenistil, unidades=25, preco=4.5]=0, Medicamento [n
Farmacia1: {Medicamento [nome=rennie, unidades=25, preco=18.5]=0, Medicamento [nome=strepfen, unidades=25, preco=10.2]=0, Medicamento

HISTÓRICO DE VENDAS TOTAL POR FARMÁCIA
Farmacia0: {Medicamento [nome=brufen, unidades=25, preco=1.7]=0, Medicamento [nome=fenistil, unidades=25, preco=4.5]=0, Medicamento [n
Farmacia1: {Medicamento [nome=rennie, unidades=25, preco=18.5]=0, Medicamento [nome=strepfen, unidades=25, preco=10.2]=0, Medicamento

HISTÓRICO DE LUCRO MENSAL POR FARMÁCIA
Farmacia0: {Medicamento [nome=brufen, unidades=25, preco=1.7]=0.0, Medicamento [nome=fenistil, unidades=25, preco=4.5]=0.0, Medicament
Farmacia1: {Medicamento [nome=rennie, unidades=25, preco=18.5]=0.0, Medicamento [nome=strepfen, unidades=25, preco=10.2]=0.0, Medicame

HISTÓRICO DE LUCRO TOTAL POR FARMÁCIA
Farmacia0: {Medicamento [nome=brufen, unidades=25, preco=1.7]=0.0, Medicamento [nome=fenistil, unidades=25, preco=4.5]=0.0, Medicament
Farmacia1: {Medicamento [nome=rennie, unidades=25, preco=18.5]=0.0, Medicamento [nome=strepfen, unidades=25, preco=10.2]=0.0, Medicame

TOP 3 DE PRODUTOS MAIS VENDIDOS NO ÚLTIMO MÊS POR FARMACIA
Farmacia0: {}
Farmacia1: {}

STOCK DAS FARMÁCIAS
Farmacia0: [Medicamento [nome=brufen, unidades=25, preco=1.7], Medicamento [nome=benuron, unidades=25, preco=2.0], Medicamento [nome=v
Farmacia1: [Medicamento [nome=benuron, unidades=25, preco=2.0], Medicamento [nome=aspirina, unidades=25, preco=2.5], Medicamento [nome
```

**Figura 11.** Inicialização do Relatorio Mensal.

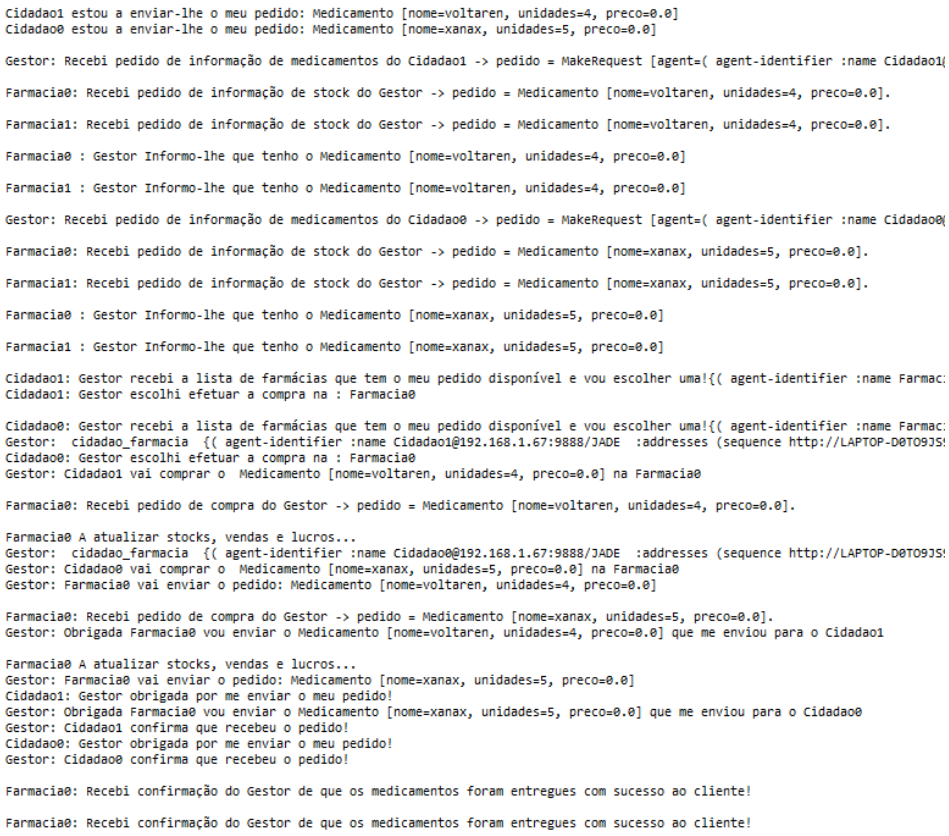



**Figura 12.** Processo de pedido, aquisição e receção de pedidos

Três ms após o fim de cada mês, a **Interface** imprime o **Relatorio** mensal (Figura 13). Como se pode verificar, no mês passado, a **Farmacia** que vendeu mais Medicamentos foi a **Farmacia** 0, que além de ter vendido os **Medicamentos** apresentados na Figura 12, vendeu também mais 4 unidades de voltarem ao **Cidadao** 1 (não está representado em Figuras). Verifica-se também que antes da impressão do **Relatorio**, ocorreu um restock proativo por parte das **Farmacias**, pois este ocorre sempre 1 vez por mês.

```
Farmacia1 A proceder ao restock proativo

Farmacia0 A proceder ao restock proativo

Fornecedor0: Recebi pedido de restock. A reestabelecer o stock na Farmacia0 para os [Medicamento [nome=voltaren, unidades=6, preco=5.6]]

********************************************************************************************************************************************
Interface: RELATÓRIO MENSAL.

FARMÁCIA COM MAIOR NÚMERO DE VENDAS NO ÚLTIMO MÊS
Farmacia0=13 unidades

FARMÁCIA COM MAIOR LUCRO NO ÚLTIMO MÊS
Farmacia0=59.8 unidades monetárias

HISTÓRICO DE VENDAS MENSAL POR FARMÁCIA
Farmacia0: {Medicamento [nome=aspirina, unidades=10, preco=2.5]=0, Medicamento [nome=voltaren, unidades=8, preco=5.6]=8, Medicamento [no
Farmacia1: {Medicamento [nome=benuron, unidades=10, preco=2.0]=0, Medicamento [nome=fucidine, unidades=10, preco=20.5]=0, Medicamento [n

HISTÓRICO DE VENDAS TOTAL POR FARMÁCIA
Farmacia0: {Medicamento [nome=aspirina, unidades=10, preco=2.5]=0, Medicamento [nome=voltaren, unidades=8, preco=5.6]=8, Medicamento [no
Farmacia1: {Medicamento [nome=benuron, unidades=10, preco=2.0]=0, Medicamento [nome=fucidine, unidades=10, preco=20.5]=0, Medicamento [n

HISTÓRICO DE LUCRO MENSAL POR FARMÁCIA
Farmacia0: {Medicamento [nome=aspirina, unidades=10, preco=2.5]=0.0, Medicamento [nome=voltaren, unidades=8, preco=5.6]=44.8, Medicament
Farmacia1: {Medicamento [nome=benuron, unidades=10, preco=2.0]=0.0, Medicamento [nome=fucidine, unidades=10, preco=20.5]=0.0, Medicament

HISTÓRICO DE LUCRO TOTAL POR FARMÁCIA
Farmacia0: {Medicamento [nome=aspirina, unidades=10, preco=2.5]=0.0, Medicamento [nome=voltaren, unidades=8, preco=5.6]=44.8, Medicament
Farmacia1: {Medicamento [nome=benuron, unidades=10, preco=2.0]=0.0, Medicamento [nome=fucidine, unidades=10, preco=20.5]=0.0, Medicament

TOP 3 DE PRODUTOS MAIS VENDIDOS NO ÚLTIMO MÊS POR FARMACIA
Farmacia0: {Medicamento [nome=voltaren, unidades=8, preco=5.6]=8, Medicamento [nome=xanax, unidades=5, preco=3.0]=5}
Farmacia1: {Medicamento [nome=xanax, unidades=5, preco=3.0]=5}

STOCK DAS FARMÁCIAS
Farmacia0: [Medicamento [nome=brufen, unidades=10, preco=1.7], Medicamento [nome=benuron, unidades=10, preco=2.0], Medicamento [nome=asp
Farmacia1: [Medicamento [nome=benuron, unidades=10, preco=2.0], Medicamento [nome=aspirina, unidades=10, preco=2.5], Medicamento [nome=x

********************************************************************************************************************************************
```

**Figura 13.** Restock proativo nas Farmacias e apresentação do Relatorio Mensal por parte da Interface.

Por fim, apresenta-se, a título de exemplo, um gráfico de barras do stock de cada **Farmacia** (Figura 14).

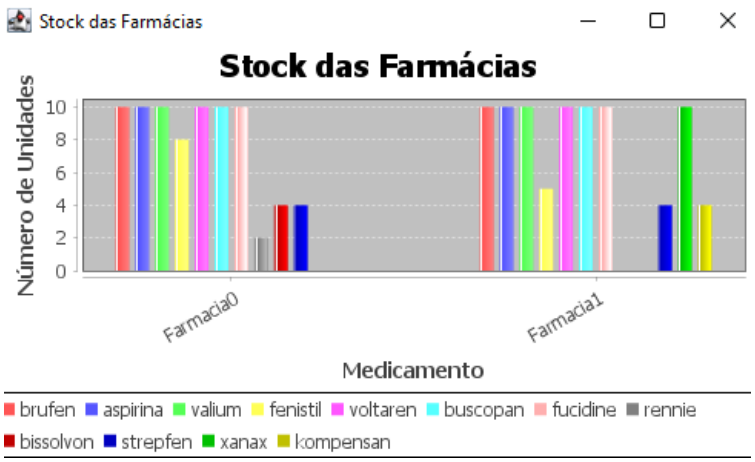



**Figura 14.** Gráfico Ilustrativo do Stock das Farmacias.

# 5. Conclusão

Através deste trabalho, foi desenvolvido um Sistema Multiagente que simula um serviço farmacêutico entre cidadãos e farmácias, tendo também intervindo um gestor, um fornecedor e uma interface que apresenta os resultados.

Desta forma, o sistema desenvolvido tentou recriar a realidade, onde os cinco tipos de agentes criados comunicam entre si empregando comportamentos como CyclicBehaviour, presente em todos os agentes, OneShotBehaviour presente no agente fornecedor e no agente farmácia e TickerBehaviour presente no agente cidadão, no agente farmácia e no agente Interface.

Verificou-se que o Sistema Multiagente desenvolvido corresponde à arquitetura projetada na primeira fase do trabalho prático: cada cidadão envia vários pedidos com periodicidade, sendo estes compostos por vários medicamentos com quantidades aleatórias. Por sua vez, o gestor gere com sucesso a compra e venda de produtos farmacêuticos, averiguando as farmácias disponíveis e efetuando a compra na farmácia selecionada pelo cidadão. As farmácias além de terem a capacidade de antecipar as necessidades dos cidadãos, são ainda responsáveis por enviar ao gestor as informações e medicamentos por ele pedidos. Os fornecedores permitem as reposições de stocks sempre que necessário e a interface imprime as estatísticas do mês passado.

Para concluir, em relação à arquitetura inicial, o único ponto que não foi totalmente atingido foi o facto de que em vez do gestor a informar o cidadão de todas as farmácias disponíveis para a venda do seu pedido, indicando a farmácia com o melhor preço para o seu pedido, este apenas envia para o cidadão uma lista com todas as farmácias disponíveis para a venda e respetivo preço. Esta alteração foi necessária pois só é possível enviar um objeto em cada mensagem ACL e além disso, num contexto real, quando se utiliza por exemplo UberEats, esta aplicação devolve uma lista com os locais onde se encontra o pedido disponível e são os utilizadores que tem de procurar, caso queiram, qual o local mais barato. Deste modo, esta alteração assemelha-se mais a um caso real, do que a inicialmente pensada. É ainda de realçar que em relação aos diagramas iniciais, verificou-se que foi necessário utilizar mais atributos do que os previamente definidos.

# Referências

[1] Stanford Edu. http://jmc.stanford.edu/artificial-intelligence/what-is-ai/index.html. Acedido a 29 nov 2021.

[2] Gerhard Weiß, "Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence", MIT Press, EUA, 1999.

[3] Pattie Maes, "Situated Agents Can Have Goals", Designing Autonomous Agents, Maes (editor), MIT Press.

[4] César Analide, Paulo Novais, "Agentes Inteligentes", Universidade do Minho, 2006.

[5] Labrou, Y. and Finin, T. (1997). Semantics and Conversations for an Agent Communication Language. In Huns, M. and Singh, M., editors, Readings in Agents. Morgan Kaufmann Publishers.

[6] Neches, R., Fikes, R., Gruber, T., Patil, R., Senator, T., and Swartout, R. W. (1991). Enabling Tecnology for Knowledge Sharing. AI Magazine, 12(3).

[7] Gruber, T. (1993). A translation approach to portable ontologies. Knowledge Acquisition, 5(2):199220.

[8] Bellifemine, F. Poggi, A. and Rimassa. G. (1999). JADE: a FIPA-compliant Agent Framework. In Proceedings of the International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM99), pages 97 108, London, UK.

[9] Caire Giovanni, "JADE Tutorial JADE Programming For Beginners", Telecom Italia S.p.A., 2009.

[10] Unified Modeling Language (UML) | Sequence Diagrams - GeeksforGeeks. (n.d.). https://www.geeksforgeeks.org/unified-modeling-language-umlsequence-diagrams/. Acedido a 01 dez 2021.

[11] What is Unified Modeling Language | Lucidchart. (n.d.). https://www.lucidchart.com/pages/what-is-UML-unified-modeling-language. Acedido a 01 dez 2021.

[12] Brazier, Frances M.T., Catholijn M. Jonkers, and Jan Treur, ed., Principles of Compositional Multi-Agent System Development Chapman and Hall, 1998.

[13] Herlea, Daniela E., Catholijun M. Jonker, Jan Treur, and Niek J.E. Wijngaards, ed., Specification of Behavioural Requirements within Compositional Multi-Agent System Design Springer, Valencia, Spain, 1999.

[14] Jonker, Catholijn M., and Jan Treur, ed., Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness Springer, 1997.
[15] Kinny, David, and Michael Georgeff, "Modelling and Design of Multi-Agent Systems," Intelligent Agents III: Proceedings of the Third International workshop on Agent Theories, Architectures, and Languages (ATAL'96), ed., Springer, Heidelberg, 1996.
[16] Kinny, David, Michael Georgeff, and Anand Rao, "A Methodology and Modelling Technique for Systems of BDI Agents," Agents Breaking Away. 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96)., Walter VandeVelde and John W. Perram ed., Springer, Berlin, 1996, pp. 56-71.
[17] Wooldridge, Michael, Nicholas R. Jennings, and David Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design," International Journal of Autonomous Agents and Multi-Agent Systems, 3:Forthcoming, 2000.
[18] Unified Modeling Language (UML) | An Introduction - GeeksforGeeks. (n.d.). https://www.geeksforgeeks.org/unified-modeling-language-umlintroduction/. Acedido a 01 dez 2021.