Daniel Nicolau Saito

# Sistema Multiagente para Automação de Gerenciamento de Estoque de Produtos Alimentícios

Passo Fundo

2021

Daniel Nicolau Saito

# Sistema Multiagente para Automação de Gerenciamento de Estoque de Produtos Alimentícios

Monografia apresentada ao Curso de Ciência da Computação do Instituto Federal Sul-rio-grandense, Câmpus Passo Fundo, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SUL-RIO-GRANDENSE - CÂMPUS PASSO FUNDO

Orientador: Prof. Dr. João Mário Lopes Brezolin

Passo Fundo
2021

# Dedicatória

*Dedico este Texto ao inevitável futuro e suas ideias inovadoras*

# Agradecimentos

Primeiramente agradeço à vida pelas oportunidades que me trouxeram tão longe.

Segundamente agradeço à família que me apoiou de longe neste processo.

Por fim agradeço à todas as pessoas que contribuíram com este projeto, principalmente meus professores e amigos que indiretamente me inspiram todo dia.

# Epígrafe

"A inspiração vem para aqueles que buscam ver o mundo com outros olhos"

*Daniel Nicolau Saito*

# Resumo

O desperdício de alimentos, em contraste à insegurança alimentar presente no mundo, pode ser visto como um problema de distribuição. Este trabalho teve como objetivo produzir um sistema que pudesse ajudar no gerenciamento da logística de comércios do setor alimentício nas categorias varejo e atacado para a redução da quantidade de estoque parado em suas unidades. O desenvolvimento de tal sistema se deu com a utilização de agentes autônomos programados em Jason e Java interligados fracamente num sistema multiagente programado utilizando a infraestrutura de rede Jade para sua intercomunicação. A avaliação do sistema ocorreu junto a um funcionário de uma rede de atacado local e demonstrou a aplicabilidade em conceito com as ressalvas de uma mudança nas regras de negócio aplicadas.

**Palavras-chave**: Jade, Jason, logística, Java, supermercado, AgentSpeak

# Abstract

Food waste, in contrast to the food insecurity present in the world, can be seen as a distribution problem. This work aimed to develop a system that could help manage the logistics of businesses in the food sector in the retail and wholesale categories to reduce the amount of dead stock in their units. The development of such a system took place with the use of autonomous agents programmed in Jason and Java, weakly interconnected in a multi-agent system programmed using the Jade network infrastructure for its intercommunication. Its validation took place with an employee of a local wholesale network and demonstrated the applicability in concept with the reservations of a change in the applied business rules.

**Keywords**: Jade, Jason, Logistics, Java, supermarkets, AgentSpeak

# Lista de ilustrações

# Lista de abreviaturas e siglas

| | |
|---|---|
| ABRAS | Associação Brasileira de Supermercados |
| AMS | (*Agent Management System* ), Sistema de Gerenciamento de Agentes |
| BDI | (*Believes, Desires, Intentetions*), Crenças, Desejos, Intenções |
| DF | (*Directory Facilitator*), Facilitador de Diretórios |
| FAO | (*Food and Agriculture Organization of The United Nations*), Organização das Nações Unidas para Agricultura e Alimentação |
| FIPA | (*Foundation of Intelligent Physical Agents*), Fundação de Agentes Físicos Inteligentes |
| JADE | Java Agent Development |
| SMA | (*Multiagent System*), Sistema Multiagente |
| TCC | Trabalho de Conclusão de Curso |

# Sumário

# 1 Introdução

A automatização possibilitou ao ser humano produzir uma grande quantidade de alimentos, fármacos e outros bens de consumo em grandes quantidades. Isso acaba por ocasionar a produção de itens excedentes que acabam por ultrapassar a data de validade sem que haja um consumo apropriado, ocasionando desperdício. Essa situação, revela uma distribuição ineficiente de alimentos, tornando-se uma das causas dos problemas alimentícios numa região e também prejudicando o faturamento de redes de supermercado, que compram produtos do agricultor e não conseguem repassar uma quantidade suficiente ao consumidor final.

A comparação entre o aumento populacional e a produção alimentícia não segue o padrão malthusiano (MALTHUS, 1872), sendo que hoje há uma grande quantidade de comida disponível, em contraste a 8,21 milhões de pessoas que estão em estado de insegurança alimentar no mundo (FAO, 2018). Atualmente a produção de alimentos e fármacos é automatizada, mas a venda de tais produtos, como por exemplo a compra por parte de varejos e atacados não é tão eficiente, pois gera um desperdício no transporte e também na distribuição desses alimentos entre as filiais de uma empresa. Cerca de 1,3 bilhão de toneladas de alimento em todo o globo, segundo a ABRAS, é desperdiçada por ano, o que representa cerca de 30% da produção mundial (ABRAS, 2019).

Nascimento (2018), realizou entrevistas com 11 redes de supermercado existentes na região de Brasília, essas empresas entrevistadas denotam que há diversos problemas além da distribuição propriamente dita de alimentos, tal como o manuseio de produtos por parte dos funcionários e dos clientes, a refrigeração de produtos que necessitam, tais como carnes, o transporte, a data de validade, entre outros. A autora também assinala que todas as empresas entrevistadas em sua pesquisa possuem um sistema de cálculo de demanda para diminuir a perda de produtos. Entretanto, não se observa uma estratégia de diminuição de perdas considerando a realocação de produtos para alguma unidade que esteja com demanda maior. Ou seja, caso haja uma variação brusca de demanda numa determinada região, a probabilidade de faltar produtos pode ser maior. Considera-se, nesse sentido, o desenvolvimento de um sistema multiagente como possibilidade de reduzir ou minimizar o problema.

Para a redução de gastos e de excedentes desnecessários, é proposto um sistema multiagente em cada unidade de uma rede comercialização de produtos alimentícios no qual cada agente avalia a oferta e a demanda de produtos após um período de tempo e comunica-se com as demais filiais para escolher para quem irá exportar seu excedente e de quem importará o que falta, a fim de manter estoques mínimos. O sistema multiagente

apresentado nesse trabalho foi desenvolvido para amenizar o problema da distribuição de produtos para redes de supermercados e atacados.

## 1.1 OBJETIVOS

Desenvolver um sistema multiagente para gerenciar a redistribuição de produtos alimentícios entre unidades de uma empresa considerando a entrega de produtos da distribuidora para as filiais e entre filiais.

### 1.1.1 Objetivos Específicos

- Estudar a problemática da distribuição de estoque nas indústrias do ramo alimentício;
- Revisar bibliografia e avaliar propostas de sistemas correlatos;
- Avaliar tecnologias para desenvolvimento do sistema proposto;
- Criar uma interface de comunicação entre as unidades através de um *middleware* para compartilhamento de informações e oferta/demanda de produtos;
- Utilizar o sistema proposto para avaliar padrões de consumo local e corrigir a quantidade de demanda de cada produto através de simulações;
- Estabelecer interface entre sistema de estoque local e corrigir a quantidade de demanda de cada produto;
- Avaliar a eficácia do sistema proposto com a colaboração de usuários.

## 1.2 ORGANIZAÇÃO DO TRABALHO

O presente trabalho é constituído por capítulos, o que possibilita uma melhor organização e compreensão dos temas abordados. 8 O Capítulo 2 apresenta o referencial teórico para a construção do sistema proposto, seguido pelo Capítulo 3 onde é escrito o desenvolvimento do sistema. Após a explicação quanto ao desenvolvimento, é apresentado no Capítulo 4 a avaliação feita juntamente à usuários da área, sendo seguida pelas considerações finais no Capítulo 5.

# 2 Referencial teórico

Este capítulo apresenta a problematização da distribuição de produtos perecíveis e apresenta os principais fundamentos acerca de sistema multiagente e as tecnologias que serão utilizadas para o desenvolvimento do projeto.

## 2.1 Distribuição de Estoque de produtos perecíveis

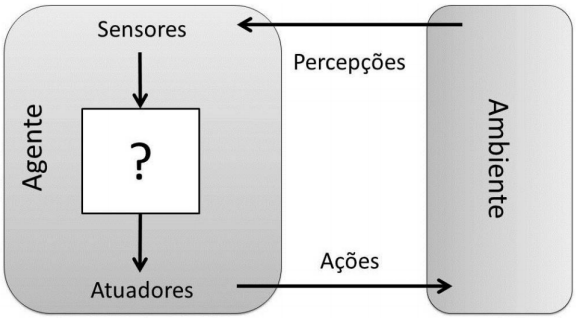
A distribuição de estoque de produtos alimentícios varia de acordo com a região e com a empresa analisada, sendo que em alguns casos há uma necessidade de um sistema de gerenciamento de estoque entre as filiais da rede, em outros casos já é utilizado um sistema complexo para gerenciamento de estoque local e global dos produtos oferecidos pelas lojas. Na pesquisa realizada para o desenvolvimento deste trabalho, anexada no Apêndice A, um funcionário de uma empresa que respondeu o questionário relatou que esta rede de atacados já utiliza um sistema completo de gerenciamento de estocagem e de distribuição de produtos entre as filiais (*crossdock*), além de gerir internamente o estoque e considerar uma margem de 5% já destinada à quebra, ou seja, com uma porcentagem baixa de desperdício.

Segundo Nascimento (2018), os principais fatores para o desperdício de alimentos nos mercados são a distribuição, o armazenamento indevido que acontece nas filiais e a data de validade dos produtos. Além disso, outros fatores também interferem nesse processo como o de manuseio inadequado por parte dos funcionários e dos clientes. Conforme a pesquisa realizada pela a autora, as empresas pesquisadas possuem um software para previsão de demandas, mas nenhum para realocação entre filiais. Entre as redes entrevistadas a estratégia de diminuição de perdas varia entre reaproveitamento dos alimentos em outros produtos ou seu uso para a alimentação dos próprios funcionários. Nesse caso, vale lembrar que os produtos ainda estão próprios para consumo, mas não conseguirão ser vendidos. Dessa forma, conclui-se que não há uma política de equalização de estoque entre filiais que também é dificultada pelas questões relativas ao transporte.

## 2.2 Sistemas Multiagentes (SMA)

Agentes racionais são definidos como entidades ativas, capazes de analisar e atuar sobre o ambiente em que se encontram inseridos. As ações imprimidas pelo agente são resultantes do processo de raciocínio orientado pelos objetivos com os quais ele se encontra comprometido. Os agentes são reconhecidos como entidades autônomas capazes de perceber o ambiente através de sensores conectados e interagir com este através de atuadores

Figura 1 – Esquema de um agente

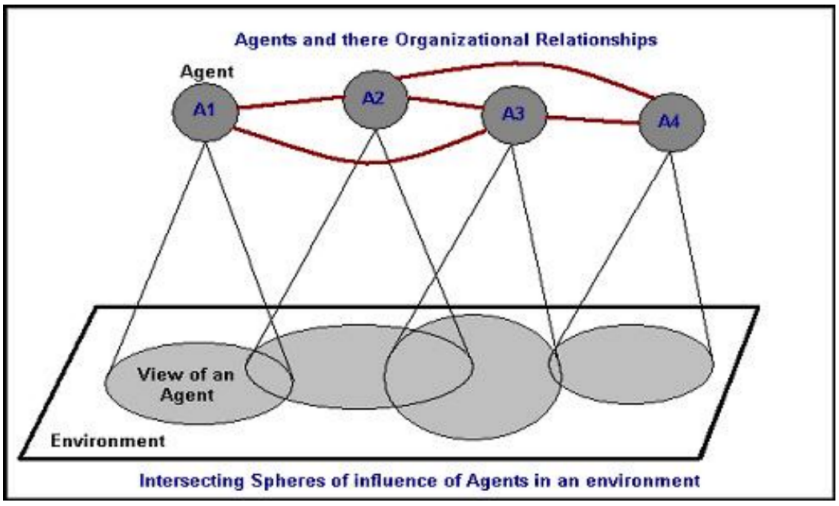



Fonte: (HUBNER JOMI F.; BORDINI, 2015)

(RUSSELL; NORVIG, 1995). A Figura 1 representa o esquema básico do processo de raciocínio implementado pelo agente: cada agente é composto por sensores e atuadores que percebem e atuam respectivamente, com o mundo a sua volta.

Um Sistema Multiagente (SMA) pode ser definido como uma extensão da tecnologia de agentes onde um grupo fracamente conectado de agentes autônomos age num ambiente para alcançar um objetivo comum, compartilhando ou não conhecimento uns com os outros (BALAJI; SRINIVASAN, 2010), como demonstra a Figura 2, na qual cada agente tem um campo de visão e um canal de comunicação com demais agentes para a troca de informações, para que cada agente possa atingir seus objetivos contando com a informação externa às suas crenças próprias.

Figura 2 – Esquema de um SMA




Fonte: (HUBNER JOMI F.; BORDINI, 2015)

Uma abordagem para a implementação de sistemas baseados em agentes é a arquitetura BDI(*Beliefs-Desires-Intentions*). Sua origem remonta ao trabalho desenvolvido por Bratman (1990, 1999), sobre a teoria do raciocínio prático (*Theory of Human Practical*

*Reasoning*). Segundo essa teoria, o processo de raciocínio prático é composto por duas etapas: a seleção dos estados que se busca atingir e a seleção de planos necessários para que eles sejam alcançados. A primeira etapa, conhecida como deliberação, está fundada nas crenças que o agente tem sobre o ambiente. A segunda descreve um processo conhecido como raciocínio orientado por objetivos (*means-ends reasoning*) no qual se determina uma sequência de ações que são necessárias para que determinado objetivo possa ser atingido.

As crenças definidas no sistema BDI são, segundo Greenstadt (2018), fatos representando o que o agente acredita sobre o mundo, ou seja, uma representação do estado do mundo. Estas crenças podem ser obtidas através de câmeras ou demais sensores ou ainda por inferências internas ou comunicação com outros agentes conectados. Esta crença não necessariamente precisa ser verdadeira, pois pode ser fruto de um defeito ou sujeira no sensor.

Ainda segundo Greenstadt (2018), desejos são objetivos, algum estado desejado que o agente queira alcançar. Podendo existir diversos desejos num mesmo agente, este deverá escolher quais seguir primeiro, resolvendo conflitos internos, ao realizar esta escolha ele estará decidido a realizar algum desejo, ou seja, terá a intenção de fazê-lo. Estas intenções não podem ser conflitantes entre si e devem ser consistentes.

O sistema BDI tem como objetivo a programação de agentes voltados a solucionar um problema e não de encontrar uma solução ótima, visto que o tempo de processamento para se encontrar tal solução pode ser o suficiente para que o mundo ao redor deste agente tenha se alterado desde de quando seu processamento começou, tornando seu resultado ultrapassado, o forçando a realizar novos cálculos.

## 2.3 Tecnologias utilizadas no desenvolvimento do sistema proposto

Nesta seção são detalhadas as tecnologias que foram utilizadas na implementação do SMA proposto neste trabalho.

### 2.3.1 Jade

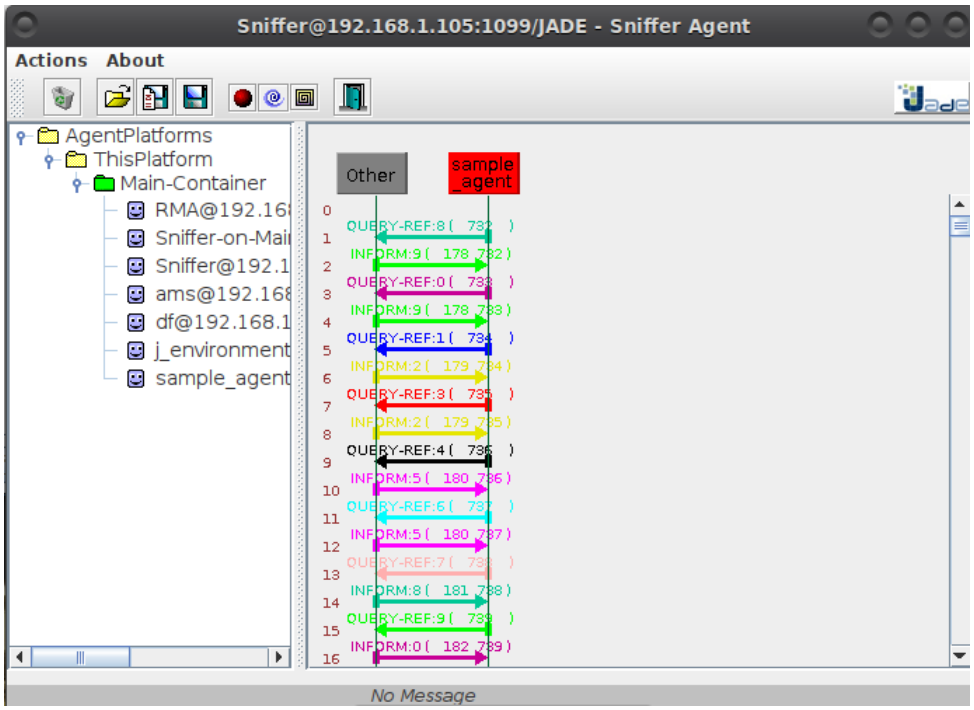
A Ferramenta Jade é um *framework* que, através da linguagem de programação Java, simplifica a implementação de SMA implementando um *middleware* que atende às especificações FIPA (*Foundation of Intelligent Physical Agents*) Estas especificações definem princípios para o desenvolvimento de agentes e sua intercomunicação, de forma a criar um consenso entre diferentes instituições, iniciando um processo de padronização (BELLIFEMINE et al., 2005).

A implementação através do Jade possibilita o envio flexível e eficiente de mensagens ACL (*Agent Communication Language*) e o controle remoto de agentes conectados à rede

através de um AMS (*Agent Management System*) e de um DF (*Directory Facilitator*), que são agentes internos do Jade para gerir de maneria mais eficiente a infraestrutura. A gestão da plataforma ocorre de forma a criar containers onde residem um ou mais agentes cada, possibilitando ao sistema ser escalonável em tempo de execução, fornecendo uma base já existente para agentes que serão executados em containers já existentes e a mudança de um agente para outro container. Os agentes AMS e DF servem para, entre outras funções, manter uma lista de agentes executados e *hosts* conectados à central onde o ambiente Jade foi executado, ficando disponível para a consulta e gerenciando deadlocks na comunicação.

Segundo a documentação do Jade (2021), o *framework* requer que haja sempre um container instanciado dentro de cada máquina. Podendo existir apenas um por *host*, esses containers são responsáveis por servir de sustentação para um ou mais agentes, como demonstrado na Figura 3.

Figura 3 – Interface do Jade




Fonte: Própria

Um SMA em Jade requer que haja pelo menos 2 agentes em sua matriz, ou seja, no contêiner principal e primordial da rede. Estes agentes, que já vêm por padrão programados, o AMS e o DF. Além destes fundamentais agentes que já são inicializados ao começo de uma nova rede Jade, também pode ser habilitado um agente *Sniffer* para vasculhar a rede em busca de agentes em *hosts* remotos. Caso não seja habilitado, o sistema apenas conseguirá tratar informações centralizadas num único *host*.

### 2.3.2 Jason

O *framework* Jason (*Java-based interpreter for an extended version of AgentSpeak*) é uma extensão da linguagem de programação abstrata chamada *AgentSpeak* orientada a agentes, que utiliza a arquitetura BDI. O *framework* possui um motor de inferência baseado em Prolog, o que permite a criação de fatos e regras utilizando a sintaxe dessa linguagem de programação.

Figura 4 – Exemplo de arquivo .mas2j

```
MAS jasonProject {

    infrastructure: Jade(main_container_host("TCCAgent1"))


    environment: RedeAgentes

    agents:
        agent1 beliefBaseClass jason.bb.JDBCPersistentBB(
        "org.postgresql.Driver", // driver for postgresql
        "jdbc:postgresql://localhost/%s", //URL connection
        "agent1dbuser", //user
        "pato", //password
        "[a(1,tablea,columns(runs(integer))),p(5,produtos,columns(id(serial
}
```

Fonte: Própria

O *framework* Jason pode ter duas infraestruturas embutidas: uma centralizada, para projetos exclusivamente existentes numa única máquina, ou utilizando o Jade, para o desenvolvimento de SMA distribuído. Tal infraestrutura exemplificada na Figura 4, onde o arquivo possui extensão *.mas2j* (SMA para java) e contém as principais declarações de um projeto em Jason, tal como a lista de agentes e um ambiente (*environment*) que tem uma funcionalidade de um sensor para todos os agentes de um determinado container.

No caso da utilização de uma infraestrutura Jade, é requerido dizer qual o endereço de IP/DNS onde o container principal da rede de agentes se localiza, tal como é feito no exemplo, determinando o container principal como sendo no *host* **TCCAgent1**, criado no projeto.

Cada agente listado pode estender uma classe específica, como por exemplo a classe de agente persistente, ou seja, que realiza um acesso ao banco de dados para manter permanente suas crenças importantes, tal como a própria lista de estoque que vai se alterando.

A integração com o banco precisa que sejam inseridos alguns parâmetros na declaração do agente, tal como as configurações de autenticação, o *Driver* do banco e o esquema de tabelas que serão criadas ou acessadas caso já existam, com os determinados campos especificados, além de como o Jason vai se referir a tais tabelas, já que para este,

Figura 5 – Exemplo de código Jason

```
/* Initial beliefs and rules */
lista_prod_names(["pao","leite"]).
lista_prod_reqs([]).
/* Initial goals */

!start.

/* Plans */

+!start
<-  .print("agent1 - Hello World");
    ?lista_prod_names(X);
    .print("lista nomes: ",X);
    loc.localinterface("Agent1");
```

Fonte: Própria

as tabelas nada mais são que crenças a serem lidas, editadas, removidas ou adicionadas. Um exemplo dessa interação pode ser visto na Figura 17.

Figura 6 – Exemplo de Ação Interna

```
public class localinterface extends DefaultInternalAction implements PropertyChangeListener{

    private static JFormattedTextField latTextField;
    private static JFormattedTextField longTextField;
    public Double latitude, longitude;

    @Override
    public Object execute(TransitionSystem ts, Unifier un, Term[] args) throws Exception {

        final TransitionSystem transSystem = ts;

        String title = ( (StringTerm)args[0]).getString();
        localinterface win = new localinterface();

        JFrame frame = new JFrame(title);
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
```

Fonte: Própria

Uma linguagem ser orientada à agentes é a construção em código de entidades que possuem objetivos próprios, pré-programados ou não, que vão ao máximo tentar alcançar estes objetivos e definir prioridades com base nas suas crenças e na possibilidade de execução de seus planos.

O código da Figura 5 tem um exemplo das funcionalidades do Jason e de como a linguagem aborda o conceito de BDI, mostrando de início duas crenças que são duas listas, uma com dois elementos e outra vazia. Seguidas por um desejo *!start* que possui um plano definido *+!start* onde é mostrado uma mensagem de boas vindas e é perguntado o estado da crença *lista_prod_names* cujo valor é retornado na variável **X** e mostrado no console em seguida. Por último é executada uma ação interna, funcionalidade na qual o Jason executa uma classe java que estende uma *DefaultInternalAction* (Ação Interna

padrão). A ação interna chamada pelo agente está demonstrada na Figura 6

Uma característica do Jason é que ele possui suporte a banco de dados. Isso é tratado com a declaração do agente como um tipo *beliefBaseClass* que utiliza o JDBC do Java, com alguns parâmetros como o driver do PostgreSQL, o pool de conexões, o usuário e a senha, como mostra a Figura 4. A criação e interação com as tabelas se dá com a tradução destas em crenças acessíveis à linguagem.

## 2.4 Trabalhos relacionados

Para estabelecer um melhor embasamento para a pesquisa, foram pesquisados trabalhos com temas relacionados à presente proposta. Nessa seção, são apresentadas semelhanças e diferenças desses trabalhos em relação a proposta apresentada nesse trabalho.

O trabalho desenvolvido por Lao e Long (2002) propõe um sistema multiagente para modelar o complexo problema de otimização chamado *Problema de Roteamento de Inventário (IRP)*. No trabalho proposto em Singapura, há apenas um fornecedor no sistema que distribui um único produto para diversos clientes, utilizando a política de reabastecimento chamada "inventário gerenciado por vendedores". No sistema por eles proposto, o fornecedor possui uma quantidade infinita de estoque do produto e cada cliente mantém um pequeno estoque local desse produto que é consumido numa taxa constante. Este fornecedor constantemente monitora o inventário de cada cliente e centraliza as decisões de reabastecimento. Seu trabalho representa clientes, rotas e entregas como agentes distintos que cooperam e competem por recursos.

Kim, Kwon e Kwak (2010) desenvolveram na Coréia do Sul um trabalho com a proposta de criar um algoritmo distribuído utilizando um modelo multiagente para realizar o controle de uma cadeia de abastecimento, o modelo proposto por eles é baseado em níveis de reabastecimento, onde existem agentes de varejo, que representam o nível de serviço ao consumidor, e agentes de reabastecimento, que atendem à requisições feitas de níveis mais baixos quantas vezes quanto a demanda variável tornar necessária. Um exemplo seria: os agentes de varejo são nível 0, que enviam sua requisição para agentes de reabastecimento que estão no nível um, este respondendo à requisição e encaminhando um reabastecimento para o nível 2 de agentes, assim por diante.

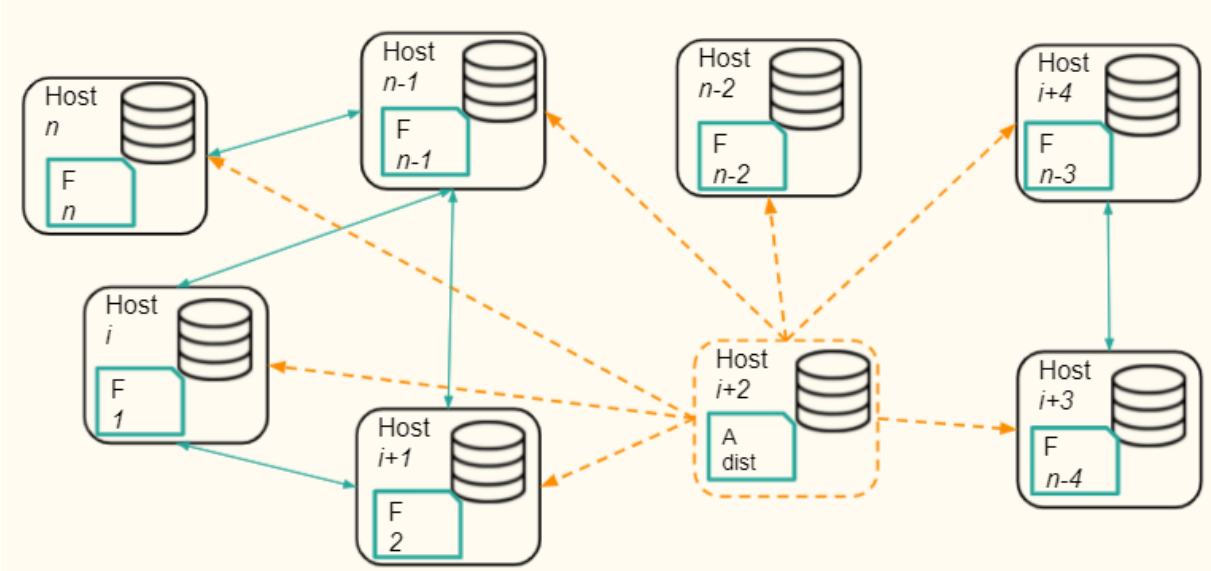
Ambos os trabalhos utilizam-se de um sistema multiagente para distribuir até certo ponto o gerenciamento de produtos. Porém, enquanto (LAO; LEONG, 2002) se concentraram em um problema para ditar as rotas que deveriam ser usadas com um único fornecedor, centralizando as decisões de reabastecimento, o artigo produzido por (KIM; KWON; KWAK, 2010) foca no abastecimento em múltiplos níveis de estoque, considerando ainda uma demanda volúvel.

No sistema proposto nesse trabalho, diferentemente, não se utiliza uma arquitetura multi nível para lidar com o gerenciamento, possuindo apenas os agentes de reabastecimento e os agentes de varejo, ainda levando em conta que os agentes de varejo, diferentemente dos apresentados pelos artigos relacionados, serão imbuídos de comunicar-se entre si para a oferta de produtos, sem que cada agente compartilhe a própria demanda.

# 3 Desenvolvimento do aplicativo proposto

Neste capítulo são abordados os aspectos do desenvolvimento do aplicativo proposto. A Figura 7 retrata um exemplo de sistema onde há uma quantidade $n$ de *hosts* sendo que cada um possui um agente e um banco de dados onde suas crenças são armazenadas. Os agentes nomeados como $F$ são pertencentes a filiais, enquanto o nomeado como $A$ pertence à distribuidora. Este último é representado em linhas tracejadas, bem como suas conexões de entrega de produtos, no caso abrangendo a totalidade da rede. As em linhas contínuas, enquanto isso, representam os *hosts* e suas comunicações limitadas por distâncias hipotéticas, mostrando suas possibilidades de entrega e requisições de produtos.

Figura 7 – Diagrama do sistema multiagente




Fonte: própria

O sistema multiagente foi implementado utilizando o *framework* Jason que possui a infraestrutura Jade para a comunicação entre vários agentes em diversos *hosts*. A comunicação dos agentes ocorre através de mensagens no padrão FIPA e terá como objetivo compreender a posição relativa dos demais agentes e posteriormente avaliar requisições feitas por outros agentes em prol da manutenção da relação entre estoque e demanda.

As características que direcionam o processo de negociação entre os agentes são:

- Saber a própria posição;
- Ter conhecimento das posições dos demais agentes;

- Saber a distância entre si e outros agentes para fins de decisão;
- Armazenar como crenças num banco de dados os agentes num determinado raio hábil para a negociação de produtos;
- Escolher dentre as requisições dos agentes o que seu excedente melhor consegue suprir.

Figura 8 – Plano para cálculo de local

```
+!start
<-  .print("agent1 - Hello World");
    ?lista_prod_names(X);
    .print("lista nomes: ",X);
    loc.localinterface("Agent1");
    ?show_all_products.


+!local(X,Y)
<-  .print("X: ",X, "\n Y: ",Y);
    +posicao(self,X,Y).
```

Fonte: Própria

Quando um novo agente que se conecta no SMA primeiramente adquire a crença de sua posição. A descoberta de sua posição é feita através da interação com o ambiente por meio de uma ação interna implementada na linguagem de programação Java. O plano que dispara essa ação é apresentado na Figura 6, que será chamada no Jason. As ações internas são formas de se executar classes java executando nelas alguma programação mais ostensiva ou extensiva, tal como a criação de uma interface ou o cálculo de distância utilizando Pitágoras, ambas utilizadas no projeto. No caso do projeto, tal classe se chama **localinterface** dentro do pacote java chamado **loc**. Esta chamada da ação executa uma tela de interface com o usuário desenvolvida utilizando a API Swing no formato visto na Figura 8, com o nome do pacote e o nome da classe.

Figura 9 – Criação do plano +!Local

```
JButton confirm = new JButton("Confirm");
confirm.addActionListener( new ActionListener(){
    public void actionPerformed(ActionEvent e){
        System.out.println("latitude inside ActionListener: " + ((Number)latTextField.getValue()).doubleValue());
        transSystem.getC()
        .addAchvGoal(
            Literal.parseLiteral("local("
            + ((Number)latTextField.getValue()).doubleValue() +","
            + ((Number)longTextField.getValue()).doubleValue() +")"), null);
    }
});
```

Fonte: Própria

Nesta ação interna ocorrerá a inserção das coordenadas **X** e **Y** correspondentes às coordenadas geográficas da loja em questão. Então, ao se submeter o valor, a ação interna

executará um plano de nome **+!local(X,Y)**, como demonstrado na Figura 9, contendo os parâmetros digitados pelo usuário e armazenando-os numa crença chamada **posicao** em seguida relatando tal posição para todos os agentes conectados na rede, como mostrado na Figura 8.

O envio em *broadcast* da posição do agente é recebida pelos demais na rede e processado de forma a calcular a distância relativa a este novo agente e se ainda não presente, adicionem-no à própria base de dados como uma crença persistente. Após adicionar ou atualizar a posição do novo agente, é enviada como retorno a posição para que este novo *host* possa também saber a posição de cada um dos demais agentes e adicioná-los à sua base de dados. Este processo pode ser analisado na Figura 10.

Figura 10 – plano RemotePosition

```
+remotePosition(Xremote,Yremote)[source(N)]
: N \==self & not( d(Nome,Dist,Distibui) & Nome=N)
<- .print("remote position of ",N,": X=",Xremote," Y=",Yremote,"\n");
    ?posicao(self,X,Y);
    dist.calculate(X,Y,Xremote,Yremote,Distancia);
    if(Distancia <= 20){
        +d(N,Distancia,true);
    };
    .send(N,tell, remotePosition(X,Y)).

+remotePosition(Xremote,Yremote)[source(N)]
: N \==self & (d(Nome,DistOld,Distibui) & Nome=N)
<-  .print(N," já está no banco");
    ?posicao(self,MyX,MyY);
    dist.calculate(MyX,MyY,Xremote,Yremote,NovaDistancia);
    if(NovaDistancia <=20 & NovaDistancia\==DistOld ){
        -+d(Nome,NovaDistancia,Distibui);
    }elif( NovaDistancia >20){
        -d(Nome,NovaDistancia,Distibui);
    };
    .send(N,tell,remotePosition(MyX,MyY));
    ?lista_prod_names(ListaNomes);
    ?lista_prod_reqs(ListaValores);
    .send(N,tell,receiveList(ListaNomes,ListaValores));
    !media(MediaMinha,ListaValores);
    +minhaMediaReqs(MediaMinha).
```

Fonte: Própria

A avaliação da distância é importante pois, dado o conjunto $S_i$(i=1,2,3,...,n) de agentes, não haverá n-1 conexões para cada agente i ∈ S, mas sim apenas conexões entre agentes próximos, reduzindo o tráfego desnecessário de informações e otimizando a procura por uma melhor oferta de produtos, caso haja mais de uma unidade na área. Este cálculo é realizado através de outra ação interna executada a partir do plano **+remotePosition**, pertencente ao pacote **dist**. A ação recebe as coordenadas locais e as coordenadas remotas, retornando na variável **NovaDistancia** a distancia entre si e o agente remoto, o cálculo é feito em java e mostrado na Figura 11

Os agentes do sistema estão programados para alcançar sua balança entre estoque

Figura 11 – Ação interna que calcula distância

```java
public class calculate extends DefaultInternalAction {

    @Override
    public Object execute(TransitionSystem ts, Unifier un, Term[] args) throws Exception {
        ts.getAg().getLogger().info("executing internal action 'dist.calculate'");
        try{
            Double myX = ((NumberTerm)args[0]).solve();
            Double myY = ((NumberTerm)args[1]).solve();
            Double remoteX = ((NumberTerm)args[2]).solve();
            Double remoteY = ((NumberTerm)args[3]).solve();

            double partialPitagorean = Math.pow(Math.abs(myX-remoteX),2)+Math.pow(Math.abs(myY-remoteY),2);

            Double dist = Math.sqrt(partialPitagorean);
            System.out.println("resultado do cálculo:");
            System.out.println(dist);

            return un.unifies(args[4], new NumberTermImpl(dist) );

        }catch(Exception e){
            ts.getAg().getLogger().warning("Error in internal action 'dist.calculate' :"+e);
        }

        return false;
    }
}
```

Fonte: Própria

e demanda. Ou seja, deixar o valor de estoque e demanda como iguais ou o mais próximo disso. Para executar essa manutenção, o agente possui um plano que envia a todos os demais agentes no seu raio de atuação duas listas com informações sobre os produtos, sendo que uma das listas tem os nomes dos produtos que se pode oferecer ou que se precisa receber e a outra lista possui a diferença entre o estoque e a demanda desses produtos da primeira lista.

A fim de testar a comunicação e alteração de produtos entre as filiais, há, no plano *+remotePosition* o trecho com a função *.send(N,tell,receiveList(ListaNomes,ListaValores)*, apresentado na Figura 10, que **diz**(*tell*) para o agente remoto N executar o plano *+receiveList*, passando como parâmetros a Lista de nomes de produtos e a lista da diferença entre estoque e demanda de cada produto. Estas listas são recuperadas das crenças do agente.

Figura 12 – Plano calculador da média local e remota recebida

```
+receiveList(NomesProdutos,ValoresProdutos)[source(Root)]
<-  +balanca(Root,0);
    !show_income(NomesProdutos,ValoresProdutos,Root);
    !media(MediaRemote,ValoresProdutos); // media dos remoto
    ?my_requestesd_products(MyListOfNames,MyListOfRequests);
    !media(MediaLocal,MyListOfRequests);
    MediaReqs = MediaRemote - MediaLocal;
    .print("media das requisicoes: ",MediaReqs);
    if(MediaReqs < 200 & MediaReqs > -200){
        .print("Ask if he is a good exchange for the other one");
        .send(Root,achieve,avaliarMedias(MediaReqs));
    }.
```

Fonte: Própria

Estes parâmetros são então lidos no plano *+receiveList* que será executado no

agente remoto. O plano apresentado na Figura 12 é que realiza o processo de buscar no banco de dados através do plano *!show_income*. Este por sua vez verifica a existência do produto no próprio banco de dados para excluir aqueles que estão na base remota mas não na sua base de dados local.

Este plano é mostrado na Figura 13, sendo que a intenção *p(_,A,_,Estoque,Demanda)* é a crença que representa a tabela "produtos" do banco de dados local, e neste plano é utilizada como pesquisa para posteriormente adicionar estas informações em uma lista, e esta numa crença.

Figura 13 – Plano para buscar no banco de dados os itens para cálculo de média

```
+!show_income([],[],Root).
+!show_income([A|T],[H|Y],Root) :  p(_,A,_,Estoque,Demanda)
  <-    ?my_requestesd_products(ListaNomes,ListaRequests);
        !insert_list(A,ListaNomes,FinalListaNomes);
        !insert_list(Estoque-Demanda,ListaRequests,FinalListaRequests);
        -+my_requestesd_products(FinalListaNomes,FinalListaRequests);

        !show_income(T,Y,Root).
+!show_income([_|T],[_|Y],Root)
  <- !show_income(T,Y,Root).
```

Fonte: Própria

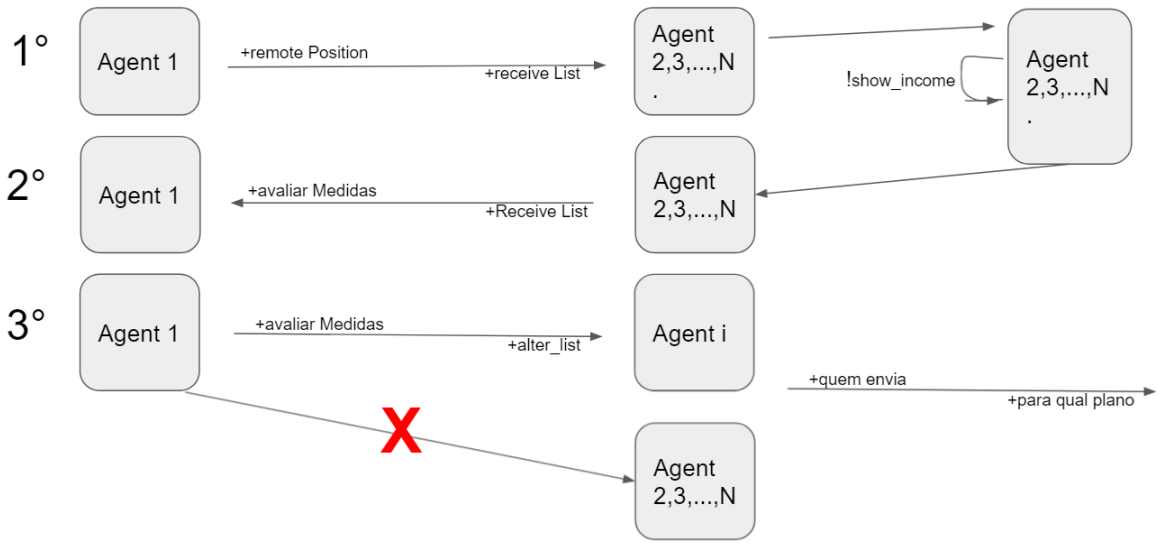
Após a execução do plano *!show_income*, o plano *+receiveList()* calcula a média dos valores da requisição feita pelo agente remoto. As listas de produtos e médias são então lidas para calcular um limiar que define um interesse noutro agente. No exemplo da Figura 12, isso foi manualmente colocado como 200, de forma que se a requisição possuir média inferior ao módulo de 200, será enviada sua média para avaliação, demonstrando um interesse.

Todo esse processo é representado na Figura 14. Onde resumidamente mostra que um agente arbitrário "um" inicia a comunicação enviando sua lista e recebe a média de todos os comunicantes, em seguida escolhe para qual agente vai concluir a transação e diz-lhe para alterar seus produtos, após também executar a própria intenção *alter_list*, com objetivo de alterar também seu próprio estoque.

Como visto na Figura 14, o plano chamado *+avaliarMedias* tem sua execução solicitada e portanto realiza a tarefa de buscar a lista das requisições remotas, salvas em crenças a cada vez que este plano é executado e liberado a partir que um número satisfatório de agentes responda à requisição, no caso configurado manualmente para 2 agentes responderem, como revelado na Figura 15 com o condicional *if*, onde se é lida as crenças para envio das informações pela rede.

Em todo o caso é calculado qual seria a escolha da média de requisição mais próxima do necessário através da função interna *calculatemod*, onde se é retornado qual o agente remoto que possui a maior vantagem de negócio. A decisão, após a chegada de um número

Figura 14 – Esquema do envio do fluxo das mensagens para negociação




Fonte: Própria

Figura 15 – Plano para avaliar as médias remotas e selecionar a mais vantajosa

```
+!avaliarMedias(MediaReqs)[source(N)]
<-  ?listaReqs(ListaSource,ListaMedia);
    !insert_list(N,ListaSource,FinalListaSource);
    !insert_list(MediaReqs,ListaMedia,FinalListaMedia);
    -+listaReqs(FinalListaSource,FinalListaMedia);
    !show_req_list(FinalListaSource,FinalListaMedia);
    ?minhaMediaReqs(OwnMedia);
    mod.calculatemod(OwnMedia,FinalListaSource,FinalListaMedia,Answer);
    .print("answer: ",Answer);
    ?respostas(NumberOfResps);
    -+respostas(NumberOfResps+1);
    ?respostas(NumeroAtualDeRequisicoes);
    if(NumeroAtualDeRequisicoes >=2){
        ?lista_prod_names(MyProductNames);
        ?lista_prod_reqs(MyProductReqs);
        .send(Answer,achieve,alter_list(MyProductNames,MyProductReqs));
        !alter_list(MyProductNames,MyProductReqs);
    }.
```

Fonte: Própria

suficiente de requisições, é enviada através da intenção *.send(Answer,achieve,alter_list)* para o agente escolhido e depois o plano *alter_list* local é executado para fazer tal alteração também no banco de dados local.

Figura 16 – Ação interna que visa o cálculo do valor mais próximo da média local

```
ts.getAg().getLogger().info("\n\nsize of Array: "+remoteNamesString.size()+"\n\n");
if(remoteNamesString.size()<=1){
    return un.unifies(args[3], new LiteralImpl(remoteNamesString.get(0)));
}

for(Integer i=0;i< remoteNamesString.size();i++){
    hashMapTest.put(remoteMediasDouble.get(i),remoteNamesString.get(i));
    ts.getAg().getLogger().info("\n\n value of remote Medias: agent:"+remoteNamesStr
}
Double above = hashMapTest.ceilingKey(myMedia);
Double below = hashMapTest.floorKey(myMedia);
ts.getAg().getLogger().info("\n\n\nvariables of CalculateMod: above:"+ above + "\n b
Double bestValue =remoteMediasDouble.get(0);
String bestSource = remoteNamesString.get(0);
if(below != null && above != null){
    bestValue = myMedia - below > above - myMedia ? above : below;
    bestSource = hashMapTest.get(myMedia - below > above - myMedia ? above : below);
}else if(above != null){
    bestValue = above;
    bestSource = hashMapTest.get(above);
}else if(below != null){
    bestValue = below;
    bestSource = hashMapTest.get(below);
}
return un.unifies(args[3], new LiteralImpl(bestSource));
```

Fonte: Própria

Figura 17 – Plano para alterar o banco de dados local

```
+!alter_list([],[]).
+!alter_list([H|T],[N|L]) : p(Id,Nome,Preco,Estoque,Demanda) & H=Nome
<-  .print("\n\nProduto:",H,", Request:",N);
    NewValue = Estoque - N;
    -+p(Id,Nome,Preco,NewValue,Demanda);
    !alter_list(T,L).
+!alter_list([H|T],[N|L]) : not( p(Id,Nome,Preco,Estoque,Demanda) & H=Nome)
<- !alter_list(T,L).
+!alter_list([_|T],[_,L])
<-  !alter_list(T,L).
```

Fonte: Própria

# 4 Avaliação do Sistema

Nesta seção é detalhado o funcionamento do sistema desenvolvido para minimizar os problemas de distribuição de produtos alimentícios em mercados e a avaliação feita em conjunto com um funcionário de uma rede de atacado de médio porte.

Foi necessária a construção de um ambiente simulado para realizar a demonstração do funcionamento do sistema. Este ambiente foi composto por 3 máquinas virtuais conectadas em uma rede interna. Cada máquina virtual possui 2.5GB de memória e um agente diferente, cada um representando uma filial presente na infraestrutura de comunicação. Em cada máquina virtual também se encontra um banco de dados escrito em PostgreSQL para simular uma interação com os produtos armazenados em um sistema pré-existente, sendo que para os agentes, as informações armazenadas representam crenças persistentes que ele consegue recuperar sempre que necessário.

Nos testes os agentes foram nomeados como agentes um, dois e três, sua enumeração sendo para facilitar referências à elas, estes agentes estão presentes nas máquinas "*Servidor1*", "*Servidor2*" e "*Servidor3*" respectivamente.

Ao serem inicializados, os agentes solicitam que sejam inseridas as coordenadas de da sua própria localização geográfica e em seguida calculam a sua distância em relação aos demais agentes ativos no sistema. O resultado desse calculo é adicionado no processo de raciocínio na forma de uma crença persistente, ou seja, essa informação é armazenada localmente em uma tabela do Banco de dados PostgreSQL, Optou-se por essa estratégia para otimizar o processo de descoberta, uma vez, não se faz necessário calcular novamente as distâncias dos agentes que já foram encontrados no sistema. As distâncias dos agentes são armazenadas em uma tabela nos bancos de dados, para levar em consideração nas requisições apenas os agentes dentro de 20km, utilizado como um valor para os testes. A tabela no postgreSQL utilizada para armazenar essas crenças é composta pelas colunas de nome, distância, e se é ou não uma distribuidora de cada unidade que se conectou com um determinado agente. Além desta tabela, a persistência de produtos é feita com o armazenamento de nome, preço, quantidade de estoque e demanda.

Inicialmente foram inseridos produtos na tabela correspondente de cada agente, sendo que estes produtos são os mesmos para todos com a particularidade de que seus estoques e demandas possuem valores distintos para que o cálculo da negociação possa ocorrer.

A execução dos agentes se dá na ordem de o primeiro ser o que possui junto de si a infraestrutura do ambiente Jade, em seguida qualquer um dos dois pode ser executado, em testes realizados a ordem de execução entre os agentes **dois** e **três** foi alterada e o

comportamento do sistema se manteve estável.

Ao se executar o sistema, o agente executa a ação interna para inserção das coordenadas, como há 3 agentes rodando em máquinas virtuais num mesmo computador físico, mesmo o processamento dessa parte se torna um pouco lento, mas em uma situação fora do que se encontraria na realidade. Conforme os agentes vão sendo ligados e suas posições configuradas eles já automaticamente vão executando os demais planos e comunicações sem a interferência humana ou algum atraso, isso acaba gerando um erro caso se tente ligar os três agentes para depois inserir suas posições, pois nos passos seguintes há a busca pelas posições dos outros agentes conectados, neste caso ele encontra outro agente mas não consegue ler sua crença de posição.

Com este cuidado na ordem de execução dos agentes, a procura por produtos sobrando ou faltando nos agentes conhecidos ocorreu de forma correta e até aparentemente mais rápido que o esperado, considerando o atraso de processamento inerente da execução simultânea de múltiplas máquinas virtuais com 2.5GB de memória cada. O *host* onde estas máquinas estão hospedadas possui um processador AMD Ryzen 3200G com 4MB e 16GB de RAM.

Um vídeo foi gravado para a demonstração do funcionamento e disponibilizado através da plataforma do YouTube no endereço <https://youtu.be/Us4IhgnoG2I>. No teste gravado primeiramente cada agente que foi instanciado consegue mostrar ao usuário uma interface para inserção de suas coordenadas e, assim que o último agente é executado, o sistema automaticamente determina quais os agentes que estarão envolvidos na negociação com base nas médias calculadas, no exemplo demonstrado no vídeo o agente **um** realiza uma negociação com o agente **dois** no valor de 50 unidades de cada um dos produtos, já que esta quantidade satisfaria ambos, ou seja, o estoque excedente não ficou parado pois foi redirecionado do primeiro agente para suprir a falta de produtos no agente **dois**, enquanto o agente **três** que não foi escolhido para participar da negociação não teve seu estoque alterado.

Somente quando o agente **dois** é executado, há uma escolha pois ela só acontece quando há dois ou mais agentes concorrentes para a negociação, a escolha então acarreta na mudança dos valores no banco de dados dos dois agentes selecionados para a comunicação, demonstrada novamente no vídeo, Estas comunicações ocorreram de forma esperada com a ressalva do cuidado necessário para a ordem de execução dos agentes já mencionada.

Após a fase de testes o sistema foi apresentado para um funcionário de uma empresa de atacado e varejo e sua avaliação revelou aplicável a ideia do sistema com as condições de que o cálculo de escolha refletisse melhor a realidade. Tais alterações no código seriam para atender uma perda de 5% que acontece no transporte, custos do transporte e do produto no cálculo da escolha do agente para se realizar a negociação. Além da sugestão de não se efetuar trocas quando não convir ao agente.

Com tais alterações, inclusa também a implementação de uma informação no banco de dados que seria o custo do produto, o sistema seria aplicável no mercado e sua incorporação em outras plataformas existentes seria factível.

# 5 CONSIDERAÇÕES FINAIS

Neste trabalho foi abordada uma possível estratégia para minimizar o estoque local e manter um equilíbrio entre os excedentes e déficits de filiais de uma rede de supermercado. Este objetivo foi concedido como uma tentativa de diminuir o desperdício na parte logística de supermercados.

As tecnologias utilizadas para a abordagem foram o Jason e o Jade, construídos em cima da linguagem de programação Java sendo que o Jason é uma linguagem a parte escrita em AgentSpeak, semelhante ao Prolog. A infraestrutura para comunicação entre agentes hospedados remotamente, construída em Jade, demonstrou-se bem performática além de der tratado bem muitas mensagens em tráfego simultaneamente.

O ambiente de testes foi criado contendo três máquinas virtuais, sendo que cada agente reside em uma destas máquinas. Os resultados oriundos dos testes foram satisfatórios em vista da negociação de produtos bem-sucedida. Além dos testes foi realizada uma avaliação em conjunto com um funcionário de uma rede de atacados da região que realizou apontamentos no que diz respeito à melhor escolha das regras de negócio para a tomada de decisão de um agente. Trabalhos futuros podem desenvolver melhor o sistema e aplicar tais alterações com o objetivo de aumentar a performance e a verossimilhança tornando-o passível de ser incorporado aos sistemas existentes.

# Referências

ABRAS. *Brasil desperdiça 23M toneladas*. 2019. Disponível em: <https://www.abras.com.br/clipping/geral/69338/brasil-desperdica-23-6-milhoes-de-toneladas-de-alimentos-por-ano>. Acesso em: 06 Nov. 2020. Citado na página 10.

BALAJI, P.; SRINIVASAN, D. An introduction to multi-agent systems. In: *Innovations in multi-agent systems and applications-1*. [S.l.]: Springer, 2010. p. 1–27. Citado na página 13.

BELLIFEMINE, F. et al. Jade—a java agent development framework. In: *Multi-agent programming*. [S.l.]: Springer, 2005. p. 125–147. Citado na página 14.

FAO. *Graziano da Silva fala de metodologias contra perdas e desperdícios em Fórum na UniCamp*. 2018. Disponível em: <http://www.fao.org/brasil/noticias/detail-events/pt/c/1157134/>. Acesso em: 26 Nov. 2020. Citado na página 10.

HUBNER JOMI F.; BORDINI, R. H. 2015. Disponível em: <http://jason.sourceforge.net/mini-tutorial/hello-bdi/>. Acesso em: 26 Nov. 2020. Citado na página 13.

KIM, C. O.; KWON, I.-H.; KWAK, C. Multi-agent based distributed inventory control model. *Expert Systems with Applications*, Elsevier, v. 37, n. 7, p. 5186–5191, 2010. Citado na página 18.

LAO, Y.; LEONG, H. W. A multi-agent based approach to the inventory routing problem. In: SPRINGER. *Pacific Rim International Conference on Artificial Intelligence*. [S.l.], 2002. p. 345–354. Citado na página 18.

MALTHUS, T. R. *An Essay on the Principle of Population.*. [S.l.: s.n.], 1872. Citado na página 10.

NASCIMENTO, C. R. d. Desperdício de alimentos em supermercados: causas, estratégias e consequências. 2018. Citado 2 vezes nas páginas 10 e 12.

RUSSELL, S.; NORVIG, P. *Prentice Hall Series in Artificial Intelligence*. [S.l.]: Prentice Hall Englewood Cliffs, NJ:, 1995. 32–52 p. Citado na página 13.

# APÊNDICE A – Questionário

Seja bem-vindo!

Sou Daniel Nicolau Saito e estou fazendo o Trabalho de conclusão de curso de Ciência da computação no Instituto Federal Sul Rio-grandense, orientado pelo professor João Mário Lopes Brezolin.

Venho por meio deste formulário coletar informações a respeito do processo de distribuição de produtos perecíveis, obstante no que tange a taxa de desperdício e os mecanismos de sistema que acionam uma importação por parte das filiais de produtos

1) **Você é funcionário de uma rede de atacado ou varejo?**

R: Atacado

2) **Quais são as situações que implicam numa atualização de estoque nas filiais? E nas distribuidoras?**

R: Algumas situações que implicam para uma excelência em estoque é a falta de gestão com o produto e sistemas, hoje em empresas grandes trabalha com sistema avançado exemplo SAP que já lhe da os dias de venda do produto o giro, portanto tudo via sistema. somente necessitando de pessoas para observar em sua entrada a qualidade e temperatura e fazer o peps do produto para o não vencimento em câmera e loja.

3) **Em quais categorias são divididos os produtos de alimentos à venda na rede?**

R: As categorias são divididas em curvas: A / B / C . Portanto vai de acordo com o giro do produto é qualificado em curva. Também temos os itens qualificados como de Risco ou seja as Organizações ficam em observação constante e fazem inventario ou seja um rotativo de inventario com itens de risco exemplo mussarela, picanha, contra file, file mignon, produtos que tem valores agregados altos e com grande risco de furtos.

4) **Quantos produtos alimentícios de cada categoria no total são oferecidos pela rede? (que ficam em estoque frequentemente)**

R: Hoje temos mais de 15.000 itens cadastrados, Somente o setor de perecível mais

de 6.000 itens cadastrados. Lembrando que cada item tem seu código de barra que funciona igual a um CPF ele não pode ser cancelado. Portanto muitos itens a empresa não trabalha mais, ai fica em um giro de produto de mais ou menos 12.000 itens.

5) **Considerando as categorias correspondentes, quão frequente uma filial, faz a importação de produtos alimentícios? Essa importação é feita de outras filiais ou da distribuidora próxima?**

R: Bom temos as transferências entre filiais as crossdock, recebimento direto em cada loja e temos também o recebimento em deposito onde transfere para as filiais e fornecedores outras redes.

6) **Há um período fixo para solicitação de reposição de produtos nas filiais? Essa requisição é feita por categoria de produtos?**

R: Não, hoje é tudo automático o sistema já sabe o tanto que esta faltando para a reposição ou promoção. independente da categoria ou curva do item.

7) **Há uma taxa de desperdício contabilizada por mês (ou outro período de tempo)?**

R: Sim

8)**Qual a porcentagem de desperdício de produtos que ocorre nessas filiais ou distribuidoras?**

R: 5% da venda já é destinada a quebra do produto. Em nota fiscal ou negociação por espaço loja. ou boleto. também tem a compensação em açougue que é o rendimento da carnes para sua perca.