UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

CIÊNCIA DA COMPUTAÇÃO

FELIPE FELIX DUCHEIKO

IMPLEMENTAÇÃO DE UM SISTEMA MULTIAGENTE COM MECANISMO DE NEGOCIAÇÃO DESCENTRALIZADO PARA UM ESTACIONAMENTO INTELIGENTE

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2019

FELIPE FELIX DUCHEIKO

IMPLEMENTAÇÃO DE UM SISTEMA MULTIAGENTE COM MECANISMO DE NEGOCIAÇÃO DESCENTRALIZADO PARA UM ESTACIONAMENTO INTELIGENTE

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação, do Departamento acadêmico de Informática, da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Gleifer Vaz Alves
Coorientador: Prof. Dr. André Pinz Borges

PONTA GROSSA

2019

TERMO DE APROVAÇÃO

IMPLEMENTAÇÃO DE UM SISTEMA MULTIAGENTE COM MECANISMO DE NEGOCIAÇÃO DESCENTRALIZADO PARA UM ESTACIONAMENTO INTELIGENTE

por

FELIPE FELIX DUCHEIKO

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 05 de junho de 2019 como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. Dr. Gleifer Vaz Alves
Orientador

_____
Prof. Dr. André Pinz Borges
Coorientador

_____
Prof. Dr. Paulo Leitão
Membro titular

_____
Prof. Dr. Augusto Foronda
Membro titular

_____
Prof. MSc. Geraldo Ranthum
Responsável pelo Trabalho de Conclusão de Curso

_____
Prof. MSc Saulo Jorge Beltrão de Queiroz
Coordenador do curso

AGRADECIMENTOS

Certamente estes parágrafos não irão atender a todas as pessoas que fizeram parte dessa importante fase de minha vida. Portanto, desde já peço desculpas àquelas que não estão presentes entre essas palavras, mas elas podem estar certas que fazem parte do meu pensamento e de minha gratidão.

Agradeço aos meus pais, Geraldo Miguel Ducheiko e Inês do Carmo Ducheiko, por serem exemplo de dedicação, honestidade e caráter. As minhas irmãs, Letícia Lais Ducheiko e Larissa Laísa Ducheiko, que me apoiaram e me incentivaram no decorrer de toda a graduação e também a minha sobrinha, Lavínia Ducheiko Machado, por sempre estar perto alegrando meus dias.

Aos meus amigos de sala Cintia Cararo e Felipe Soares, por todas as brincadeiras, trabalhos acadêmicos e incentivos no decorrer do curso e a todos os demais amigos que aqui não mencionei. Agradeço também ao Ricardo da Silva Rocha, pela paciência, companheirismo e incentivo, fundamentais para o desenvolvimento desse trabalho.

Agradeço ao meu orientador Prof. Dr. Gleifer Vaz Alves e ao meu coorientador Prof. Dr. André Pinz Borges, os quais não pouparam esforços para me auxiliar em toda a trajetória acadêmica. Aos amigos do laboratório LaCA (Laboratório de Computação Aplicada) pelas proveitosas discussões, ideias e troca de conhecimento.

Enfim, a todos os que por algum motivo contribuíram para a realização desta pesquisa.

“Science is not only compatible with spirituality; it is a profound source of spirituality.”

Carl Sagan

RESUMO

DUCHEIKO, Felipe Felix. **Implementação de um Sistema Multiagente com Mecanismo de Negociação Descentralizado para um Estacionamento Inteligente**. 2019. 93 f. Trabalho de Conclusão de Curso - Bacharelado em Ciência da Computação - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2019.

Encontrar uma vaga de estacionamento livre é um dos principais problemas enfrentado todos os dias pelos habitantes das cidades. Normalmente em horários de grande fluxo é gasto uma considerável parcela de tempo ou são percorridas longas distâncias até que seja encontrada uma vaga, ocasionando desperdício de combustível e gerando engarrafamentos. O projeto MAPS (*Multi-Agent Parking System*) é idealizado com objetivo de desenvolver um S*mart Parking* (estacionamento inteligente) utilizando técnicas de Sistema Multiagente. Este Sistema Multiagente para alocação de vagas de estacionamento foi desenvolvido utilizando o *framework* JaCaMo e possui um mecanismo de negociação centralizado onde existe a figura de um agente centralizador que controla a alocação das vagas. Isto impõe algumas limitações ao sistema, visto que caso este agente centralizador falhe o sistema como um todo é comprometido. Este trabalho propõe a implementação de um Sistema Multiagente com mecanismo de negociação descentralizado, visando oferecer uma alternativa de alocação de vagas para o MAPS, independente de um módulo central. O SMA apresentado neste trabalho é denominado MAPS-OPEN, pois é baseado em um SMA aberto, onde os agentes podem se comunicar e negociar livremente. O MAPS-OPEN utiliza um modelo de raciocínio e protocolo de negociação propostos pelos autores, onde os agentes podem assumir papeis distintos, conforme desejam estacionar ou deixar uma vaga. Para avaliar o modelo e protocolo aqui apresentados, o trabalho descreve um conjunto de experimentos de simulação executados com o SMA proposto, onde é possível verificar que o número de mensagens trocadas no sistema cresce conforme a quantidade de agentes e que com este crescimento do número de agente eles tendem a estacionar mais próximo do ponto onde desejavam estacionar.

Palavras-chave: Cidade Inteligente. Estacionamento Inteligente. Sistema Multiagente. Negociação Descentralizada. *Framework* JaCaMo.

ABSTRACT

DUCHEIKO, Felipe Felix. **Implementation of a Multi-agent System with a Decentralized Negotiation Mechanism in a Smart Parking**. 2019. 93 p. Work of Conclusion Course - Graduation in Computer Science - Federal Technology University - Paraná. Ponta Grossa, 2019.

Find a free parking spot is one of the major problems faced every day by the cities population. Usually, in times of great traffic flow to find a parking spot, a driver will waste time or will drive a long way until find a spot. Therefore, wasting fuel and generating traffic jams. In order to shed some light into this problem, the MAPS project (Multi-Agent Parking System) aims to develop a smart parking solution based on Multi-Agent techniques. This Multi-Agent system to allocate parking spots was developed using the JaCaMo framework and has a centralized negotiation mechanism, where a figure of a central agent controls the spot allocation, this imposes some limitations to the system because if this central agent fails the whole system is compromised. The work here presented shows the deployment of a Multi-Agent systems based on a decentralized negotiation mechanism, where the agents can freely negotiate the parking spots without the so-called central agent. The MAS described here is named MAPS-OPEN, since it is based on an open MAS, where agents can freely communicate and negotiate. The MAPS-OPEN defines a reasoning model and a negotiation protocol, where the JaCaMo agents can be assigned to different roles (buyer or seller) in order to deal with a parking spot. Moreover, we present some experiments in order to show how our MAS works in a simulated environment. Where it was possible to verify that the number of messages exchanged in the system grows according to the quantity of agents, and that with this growth of the number of agents make they tend to park closer to the point where they wanted to park.

Keywords: Smart City. Smart Parking. Multi-Agent System. Decentralized Negotiation. JaCaMo Framework.

LISTA DE ILUSTRAÇÕES

Figura 1 – Diagrama de *Venn* trabalhos relacionados 18
Figura 2 – Esquema de agente e ambiente 22
Figura 3 – Visão geral arquitetura BDI 26
Figura 4 – Estrutura de um SMA 28
Figura 5 – Visão geral *framework* JaCaMo 29
Figura 6 – Meta-modelo JaCaMo 30
Figura 7 – Trecho de código no Jason 32
Figura 8 – Trecho de código no *Cartago* 34
Figura 9 – Trecho de código no *Jason* 34
Figura 10 – Trecho de código no Moise 35
Figura 11 – Trecho de código no Moise 36
Figura 12 – Autômato de uma negociação 39
Figura 13 – Classificação de negociação 40
Figura 14 – Protocolo *Contract Net* 41
Figura 15 – Diagrama do Protocolo de Negociação 50
Figura 16 – Diagrama Visão Geral do SMA 53
Figura 17 – Negociação entre *Buyer* e *ParkingSpotController* 54
Figura 18 – Negociação entre *Seller* e *Buyer* 54
Figura 19 – Relacionamentos dos Papeis no *Moise* 56
Figura 20 – Esquema Especificação Moise 57
Figura 21 – Esquema de Negociação do SMA 67
Figura 22 – Aba *Home* da Interface Gráfica 68
Figura 23 – Aba *Parking Spots* da Interface Gráfica 69
Figura 24 – Modelo de Estacionamento Utilizado 70
Figura 25 – Simulação do SMA com o SUMO 71

LISTA DE GRÁFICOS

Gráfico 1 – Número de Mensagens Médias Enviadas e Tempo Médio para Estacionar em Relação ao Número de *Drivers* 74
Gráfico 2 – Distância Média entre o Ponto de Desejo e o Ponto Estacionado e Valor Médio de Venda das Vagas em Relação ao Número de *Drivers* 75
Gráfico 3 – Valor Médio de Venda das Vagas e Distância Média entre o Ponto de Desejo e o Ponto Estacionado em Relação a Distância Máxima Aceitável 77
Gráfico 4 – Número de Mensagens Médias Enviadas e Tempo Médio para Estacionar em Relação ao Valor Máximo disposto a pagar 79
Gráfico 5 – Número de Mensagens Médias Enviadas e Tempo Médio para Estacionar em Relação ao Valor de $\boldsymbol{\delta}$ 80

LISTA DE TABELAS

Tabela 1 – Configurações dos Cenários do Experimento 1 73
Tabela 2 – Configurações dos Cenários do Experimento 2 76
Tabela 3 – Configurações dos Cenários da Experimento 3 78
Tabela 4 – Configurações dos Cenários do Experimento 4 80

LISTA DE CÓDIGOS

Código 1 – Especificação Estrutural no *Moise* 55
Código 2 – Especificação Funcional no *Moise* 57
Código 3 – Implementação plano *arriveParking* em *Jason* 59
Código 4 – Implementação plano StartNegotiation em *Jason* 60
Código 5 – Implementação plano OfferSpot em *Jason* 61
Código 6 – Implementação plano analyzeOffer em *Jason* 63
Código 7 – Implementação plano ultimatum em *Jason* 64
Código 8 – Implementação plano leaveParking em *Jason* 65
Código 9 – Especificação Normativa no *Moise* 66

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

ABGS	*Agent Based Guiding System*
BDI	*Belief-Desire-Intention*
BRF	*Belief Revision Function*
Cartago	*Common ARTifact infrastructure for AGents Open environments*
GPAS	Grupo de Pesquisa em Agentes de *Software*
GPS	Sistema de Posicionamento Global
IA	Inteligência Artificial
IAD	Inteligência Artificial Distribuída
JaCaMo	*Jason* + *Cartago* + *Moise*
Jason	*A Java-based interpreter for an extended version of AgentSpeak*
MAPS	*Multi-Agent Parking System*
MCP	*Monotonic Concession Protocol*
NAMA	*Navigating Aircrafts Multi-Agent*
SMA	Sistema Multiagente
TraCI	*Traffic Control Interface*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO 13
1.1 OBJETIVOS 15
1.1.1 Objetivo Geral 15
1.1.2 Objetivos Específicos 15
1.2 DELIMITAÇÕES DO TRABALHO 16
1.3 JUSTIFICATIVA 16
1.4 ORGANIZAÇÃO DO TRABALHO 17
2 TRABALHOS RELACIONADOS 18
3 AGENTES E SISTEMAS MULTIAGENTES 21
3.1 AGENTES 21
3.1.1 Autonomia 23
3.1.2 Proatividade 23
3.1.3 Reatividade 23
3.1.4 Habilidade Social 24
3.2 CLASSIFICAÇÕES DE AGENTES 24
3.3 ARQUITETURAS INTERNAS DE AGENTES 24
3.3.1 Arquitetura BDI 25
3.4 SISTEMAS MULTIAGENTES 27
3.5 FERRAMENTAS 29
3.5.1 Framework *JaCaMo* 29
3.5.1.1 *Jason* 31
3.5.1.2 *Cartago* 33
3.5.1.3 *Moise* 34
3.5.2 Prometheus 37
3.5.3 SUMO 37
3.6 NEGOCIAÇÃO ENTRE AGENTES 38
3.6.1 Protocolos de negociação 40
3.6.1.1 Protocolo Contract Net 41
3.6.1.2 Protocolo monotônico de concessão 42
3.6.1.3 Faratin 43
4 DESENVOLVIMENTO DO SISTEMA MULTIAGENTE 45
4.1 MODELO DE RACIOCÍNIO 46
4.2 PROTOCOLO DE NEGOCIAÇÃO 50
4.3 IMPLEMENTAÇÃO DO SISTEMA MULTIAGENTE 51
4.4 INTERFACE GRÁFICA 68
4.5 CONEXÃO COM O SUMO 69
5 RESULTADOS 72
5.1 CENÁRIOS DE TESTES 72

5.2 EXPERIMENTO 1 73
5.3 EXPERIMENTO 2 76
5.4 EXPERIMENTO 3 77
5.5 EXPERIMENTO 4 79
6 CONCLUSÃO 82
6.1 TRABALHOS FUTUROS 83
6.2 PUBLICAÇÕES 84
REFERÊNCIAS 85
APÊNDICE A - Código especificação organizacional do sistema 89

1 INTRODUÇÃO

O conceito de *Smart City* (Cidade Inteligente) surgiu durante a última década, com a fusão de várias ideias de como tecnologias de informação e da comunicação podem melhorar o funcionamento das cidades, tendo o intuito de melhorar a eficiência e a competitividade das cidades, criando novas maneiras para solucionar problemas. A essência do conceito é integrar as tecnologias que até agora têm sido desenvolvidas separadamente, mas que tem ligações claras em seu funcionamento e podem ser desenvolvidas de forma integrada (BATTY et al., 2012).

Dentre os inúmeros desafios a serem enfrentados pelas cidades destacam-se os de mobilidade urbana. Estima-se que em Nova York cerca de 40% dos congestionamentos são ocasionados por motoristas buscando vagas de estacionamento (KOSTER; KOCH; BAZZAN, 2013). Quando se percebe que a demanda de vagas de estacionamentos não está sendo satisfeita a solução adotada normalmente é um aumento quantitativo do número de vagas. Porém, a utilização das mesmas vagas de modo mais inteligente pode amenizar ou até mesmo solucionar o problema.

Smart Parkings (Estacionamentos Inteligentes) podem auxiliar na otimização da atribuição das vagas de estacionamento, segundo Nocera, Napoli e Rossi (2014) *Smart Parkings* são sistemas compostos por dispositivos de *hardware*, capazes de detectar o nível de ocupação do estacionamento e *softwares* integrados, para gerir a atribuição desses espaços. Tais sistemas são concebidos para auxiliar os motoristas na localização de vagas disponíveis, colaborando com a solução de problemas relacionados à mobilidade urbana.

Várias abordagens de pesquisa em *Smart Parking* estão sendo desenvolvidas. Segundo Revathi e Dhulipala (2012) estas abordagens podem ser categorizadas conforme as tecnologias empregadas em sua implementação, algumas das categorias são: *Automated Parking* (foco em mecanismos computadorizados), *E-Parking* (utiliza Internet ou SMS) e *Agent Based Guiding System* (ABGS). Esta última é caracterizada pela utilização de Sistemas Multiagentes na implementação de *Smart Parkings*.

A utilização de técnicas de Sistemas Multiagentes são bem aplicadas na resolução de problemas com ambientes distribuídos e complexos, como é o caso dos *Smart Parkings*. Os agentes que compõem estes SMAs possuem características de

resolução de problemas valiosas em cenários de *Smart Parkings*. Algumas dessas características incluem autonomia, atividade, proatividade, adaptabilidade (RIZVI; ZEHRA; OLARIU, 2018).

Sistemas Multiagentes são definidos como sistemas compostos de vários elementos computacionais que realizam interações ente si, de modo a atingirem seus objetivos, sendo tais elementos conhecidos como agentes (WOOLDRIDGE, 2009). Esses agentes possuem duas características importantes: primeiramente são capazes de ações autônomas e em segundo lugar têm a capacidade de interagir uns com os outros pela interação análoga às interações sociais humanas.

Por meio das habilidades sociais os agentes devem ser capazes de negociar uns com os outros, afim de solucionarem os problemas de forma distribuída, como ocorre em sociedades. Segundo Alonso *et al.* (2008), negociação é capacidade do agente de assumir compromissos, resolver conflitos e chegar a acordos com outros agentes com o intuito de assegurar a realização de seus objetivos.

Wooldridge (2009) define três configurações para se desenvolver um mecanismo de negociação: i) *one-to-one*: onde um agente negocia somente com um outro agente, neste caso os agentes possuem preferências simétricas; ii) *many-to-one* (centralizado): nesta configuração, um único agente negocia com vários outros agentes, como acontece em um leilão; e iii) *many-to-many* (descentralizado): neste caso, muitos agentes negociam com muitos outros agentes simultaneamente, como acontece em um mercado, outra classificação para negociação mais abrangente será abordada no decorrer deste trabalho.

Com o objetivo de aplicar métodos e técnicas de Sistema Multiagente, na criação de soluções para alocação de vagas e gerenciamento de um *Smart Parking*, foi concebido o projeto MAPS (*Multi-Agent Parking System*). Este projeto é desenvolvido no GPAS (Grupo de Pesquisa em Agentes de *Software* - UTFPR - PG) e seus primeiros resultados são apresentados no trabalho de Castro (2015), onde por meio do *framework* JaCaMo (BOISSIER *et al.*, 2011) foi implementado um SMA para alocação de vagas de estacionamento. O *framework* JaCaMo é composto de três níveis: i) *Jason,* programação de agentes; ii) *Cartago,* programação do ambiente; e iii) *Moise,* especificação das regras sociais do sistema.

O SMA desenvolvido no projeto MAPS possui um mecanismo de negociação centralizado, onde todos os agentes motoristas negociam exclusivamente com o agente administrador. Esta abordagem possui vantagens, como o fato de os agentes

motoristas trocarem mensagem apenas com o agente administrador, diminuindo assim a troca de mensagens no sistema e consequentemente o custo computacional (CÂMARA et al., 2014). Mas também possui desvantagens, visto que o agente administrador pode falhar, assim comprometendo todo o SMA, pois os motoristas irão solicitar vagas e não obterão respostas do agente administrador.

O trabalho aqui proposto objetiva complementar o Projeto MAPS por meio da implementação de um mecanismo de negociação descentralizada, oferecendo uma alternativa de negociação ao SMA do MAPS e consequentemente uma nova configuração de alocação de vagas, diferente da alocação com abordagem centralizada.

Na abordagem proposta neste trabalho o SMA é aberto, onde os agentes podem entrar e sair do sistema durante a execução do SMA. Os *drivers* podem assumir papeis e negociar livremente uns com os outros, sem a existência de um agente centralizador que gerencie a alocação das vagas.

1.1 OBJETIVOS

Nesta seção serão abordados os objetivos geral e específicos a serem atingidos pelo presente trabalho

1.1.1 Objetivo Geral

O objetivo geral deste trabalho é implementar um Sistema Multiagente com mecanismo de negociação descentralizado entre agentes no contexto de um estacionamento inteligente.

1.1.2 Objetivos Específicos

Para atingir o objetivo geral deste trabalho é necessário atingir os seguintes objetivos específicos:

- Definir um modelo de raciocínio;
- Criar um protocolo de negociação descentralizado;
- Implementar os agentes que negociarão as vagas, em *Jason*;

- Implementar os artefatos do ambiente, em *Cartago*;
- Implementar o modelo organizacional do Sistema Multiagente, utilizando o *Moise*;
- Executar testes e simulações do Sistema com o mecanismo de negociação descentralizado;
- Analisar os resultados obtidos por meio dos testes e simulações.

1.2 DELIMITAÇÕES DO TRABALHO

Esta seção descreve as delimitações e escopo do presente trabalho, como pode ser observado a seguir:

- Cada agente *driver* pode assumir apenas dois papeis: *buyer* (quando está à procura de uma vaga) e *seller* (quando está saindo de uma vaga);
- Os *drivers* e as vagas de estacionamento são localizados no sistema utilizando o conceito de coordenadas cartesianas. Coordenas cartesianas identificam e localizam um ponto num plano ou espaço bidimensional, por meio de um par ordenado;
- Todos os agentes obedecem rigorosamente o protocolo de negociação e modelo de raciocínio propostos;
- Todos os testes e simulações apresentados neste trabalho apenas utilizam o mecanismo de negociação descentralizado;
- Os agentes das simulações são gerados aleatoriamente.

1.3 JUSTIFICATIVA

O projeto MAPS visa oferecer possíveis alternativas para questões de mobilidade em cidades inteligentes por meio de soluções para gerenciamento e organização de *Smart Parkings*. Os primeiros resultados do projeto foram obtidos no trabalho de Castro, Alves e Borges (2016), onde é apresentado o desenvolvimento de um Sistema Multiagente para a alocação de vagas de estacionamento com um mecanismo de negociação centralizado. Neste mecanismo um agente administrador negocia vagas de estacionamentos com vários agentes motoristas.

Segundo Shehory (1998), SMAs que dependem de um módulo central possuem desvantagens visto que a organização da estrutura é rígida e não permite uma organização dinâmica dos agentes, para que assim se adaptem às necessidades de uma tarefa específica. Geralmente, esta dependência do módulo central implica que o agente centralizador pode controlar parcialmente ou totalmente os outros agentes do sistema. Este fator entra em contraste com uma das principais características de agentes, a autonomia. Outro ponto negativo em sistemas com mecanismo de negociação centralizado é a alta dependência do sistema em relação ao módulo central. Caso esse módulo falhe certamente todo o sistema é afetado.

A implementação do mecanismo de negociação descentralizado, proposto no presente trabalho, será uma evolução para o Projeto MAPS, visto que o SMA do projeto possuirá uma alternativa para realizar negociações e alocar vagas, sem a figura do agente administrador. Com esta nova abordagem de negociação espera-se que o SMA do projeto MAPS deixe de ser suscetível a falhas de um agente administrador, visto que os agentes motoristas negociarão diretamente uns com os outros, sem o intermédio de um agente centralizador.

O SMA proposto trabalha com a alocação de recursos, as vagas de estacionamento, para agentes, os *drivers.* Segundo An, Gatti e Lesser (2016), mecanismos descentralizados são eficientes quando agentes transferem recursos que não são tão valiosos para agentes que valorizam mais estes recursos. Este é o caso do projeto MAPS, pois quando um agente deixa de usar uma vaga este recurso passa a ter um valor baixo para este agente, mas possui um valor alto para um agente que está procurando uma vaga.

1.4 ORGANIZAÇÃO DO TRABALHO

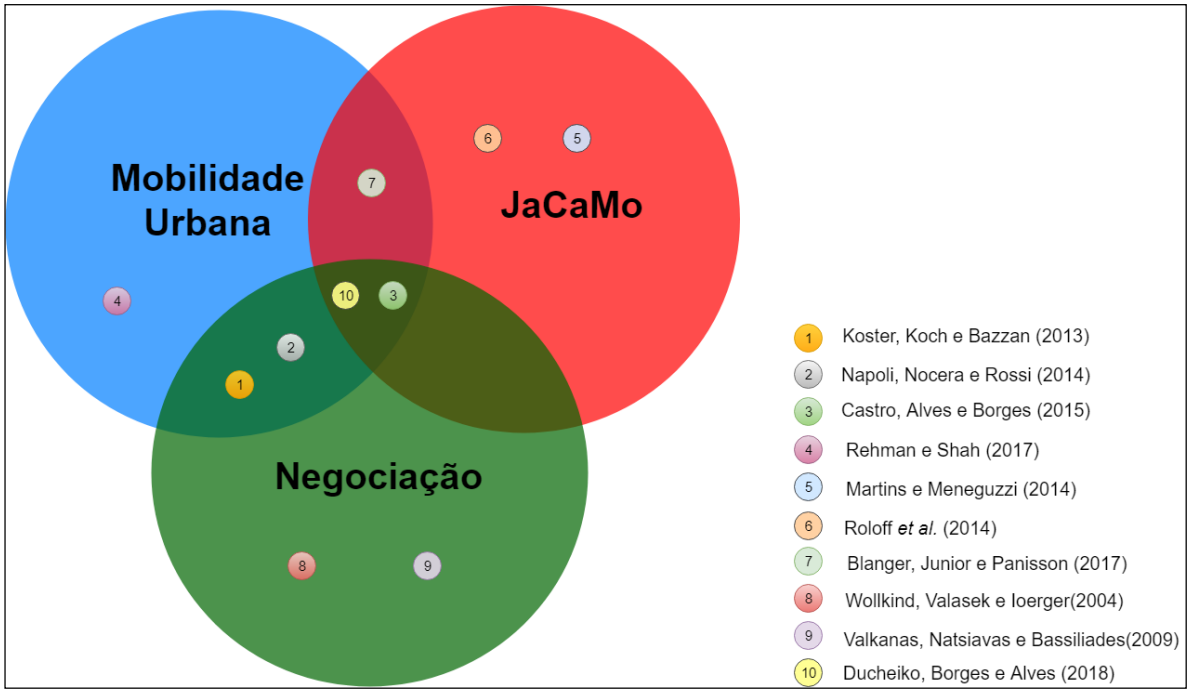
Este trabalho está estruturado em seis capítulos. No capítulo dois são apresentados os trabalhos relacionados. O capítulo três apresenta as definições sobre agentes e SMAs, bem como as ferramentas utilizadas para o desenvolvimento deste trabalho, como é o caso do *framework* JaCaMo, onde suas principais características são apresentadas. Na sequência, ainda no capítulo três é abordada a negociação entre agentes, onde três protocolos de negociação são apresentados. O capítulo quatro apresenta o desenvolvimento do Sistema Multiagente desenvolvido neste

trabalho. O capítulo cinco aborda os experimentos e resultados obtidos. Por fim, no capítulo seis é apresentada a conclusão do trabalho.

2 TRABALHOS RELACIONADOS

Existem vários trabalhos sendo desenvolvidos nas áreas de SMAs e mobilidade urbana. Esta seção apresenta alguns trabalhos relacionados bem como suas principais características e contribuições. A Figura 1 mostra um diagrama de Venn que relaciona as principais áreas da presente pesquisa (mobilidade urbana, negociação e *framework* JaCaMo) com seus respectivos trabalhos relacionados apresentados neste capítulo. No centro do diagrama de Venn é possível notar o presente trabalho (10).

Figura 1 – Diagrama de *Venn* trabalhos relacionados



Fonte: Autoria Própria.

Koster, Koch e Bazzan (2014) apresentam o *wePark,* que é uma aplicação que possibilita aos motoristas reportar e procurar vagas de estacionamentos disponíveis por meio de negociação de agentes. Esta ferramenta utiliza a colaboração coletiva de seus usuários para coletar dados comportamentais dos próprios usuários. Com esta coleta de dados os autores propõem maneiras de incentivar a adoção do

sistema por meio da utilização de métodos alternativos como reciprocidade, reputação, altruísmo.

Napoli, Nocera e Rossi (2014) apresentam um sistema para alocação de vagas baseado em um mecanismo centralizado de negociação entre agentes de *software*, neste sistema os motoristas negociam as vagas, em termos de distância e custo, diretamente com um agente administrador, que coordena a alocação de vagas de todo o sistema. Os autores também apresentam um protocolo de negociação específico para o sistema desenvolvido baseado em rodadas, propostas e contrapropostas.

Castro, Alves e Borges (2016) apresentam um SMA que utiliza o *framework* JaCaMo para alocação de vagas de estacionamento com um mecanismo de negociação centralizado. O SMA proposto pelos autores possui dois tipos de agentes: os motoristas que podem solicitar ou deixar uma vaga e um agente administrador que gerencia a alocação das vagas, negociando diretamente com cada motorista. A alocação de vagas é realizada conforme o grau de confiança de cada motorista. Este é um valor numérico que varia conforme a utilização do estacionamento pelo motorista, quanto mais vezes o motorista utilizar o estacionamento maior será seu grau de confiança, desta forma incentivando a utilização do sistema. É importante ressaltar que mesmo o trabalho destes autores tendo as mesmas áreas da presente pesquisa eles possuem singularidades e não concitem no mesmo trabalho. A principal diferença é que o trabalho aqui apresentado propõe um mecanismo de negociação descentralizado, diferentemente do proposto pelos autores.

Rehman e Shah (2017) apresentam um *Smart Parking* baseado no Sistema de Posicionamento Global (GPS) e com uma arquitetura cliente servidor. Ao final os autores apresentam um comparativo sobre o *Smart Parking* desenvolvido e um sistema de estacionamento normal, chegando a conclusão que o *Smart Parking* colabora significativamente para reduzir o tempo de procura por vagas de estacionamento, o custo das vagas e as emissões de CO2 de na atmosfera.

Martins e Meneguzzi (2014) apresentam um SMA que utiliza o *framework* JaCaMo para implementar uma *Smart Home* (Casa Inteligente), com o intuito de minimizar o consumo de energia e maximizar o conforto dos seus moradores. O SMA utiliza o nível da organização social (*Moise)* do JaCaMo para implementar os papeis em que os dispositivos eletrônicos podem operar, desde papeis econômicos de

energia até papeis de grande demanda de energia, contribuindo para evitar picos de utilização energética.

Roloff *et al.* (2014) apresentam um SMA para produção de placas de circuito impresso também utilizando o *framework* JaCaMo. Os autores aplicaram a metodologia *Prometheus* para modelarem o desenvolvimento do SMA, a qual permitiu mapear as sinergias de todos os níveis do *Framework* JaCaMo e especificar uma documentação clara do sistema.

Blanger, Junior e Panisson (2017) apresentam um SMA para gerenciar a alocação de motoristas de táxis utilizando o *framework* JaCaMo, o sistema é composto de três tipos de agentes: os clientes, que requisitam os motoristas, os motoristas, que atendem os pedidos dos clientes e um agente administrador, responsável por alocar os motoristas disponíveis para as requisições dos clientes. Segundo os autores com a implementação deste SMA o tempo de espera dos clientes foi consideravelmente reduzido.

Wollkind, Valasek e Ioerger (2004) apresentam um SMA que faz uso de técnicas de negociação entre agentes para solucionar conflitos de tráfego aéreo, este sistema possui um mecanismo de negociação descentralizado. A negociação se inicia quando um agente Aeronave identifica um possível conflito aéreo e envia uma mensagem para todos os agentes envolvidos neste possível conflito, a partir disto eles negociam a melhor solução para o problema. Para tanto é utilizado o *Monotonic Concession Protocol* (MCP), o qual será abordado de modo mais aprofundado na seção sobre negociação entre agentes.

Valkanas, Natsiavas e Bassiliades (2009) apresentam um trabalho similar ao de Wollkind, Valasek e Ioerger (2004), onde é apresenta o *framework Navigating Aircrafts Multi-Agent* (NAMA), uma plataforma baseada em SMA que permite a implementação e testes de estratégias personalizadas de voo livre, prevendo possíveis colisões entre as aeronaves. O NAMA também utiliza um mecanismo de negociação descentralizado, onde as ações de cada aeronave do sistema são simuladas por um agente Piloto Automático. Assim, não existe um agente centralizador e sim vários agentes Pilotos Automáticos, que simulam as várias aeronaves do sistema.

3 AGENTES E SISTEMAS MULTIAGENTES

A Inteligência Artificial Distribuída (IAD) é um campo de pesquisa que surgiu no final da década de 1970 e engloba áreas como Inteligência Artificial (IA), Ciência da Computação, Ciência de Organização e Gestão, Sociologia, Economia e Filosofia (WOOLDRIDGE, 2009). Segundo Wooldridge (1999) a IAD é o estudo, construção e a aplicação de Sistemas Multiagentes, que são sistemas compostos de vários agentes inteligentes e interativos os quais perseguem um conjunto de objetivos em comum.

Este capítulo aborda alguns conceitos fundamentais para compreensão do presente trabalho, como o conceito de agentes, o conceito de Sistemas Multiagentes e uma breve introdução ao conceito de negociação, o qual será abordado novamente de maneira mais aprofundada no capítulo três.

3.1 AGENTES

Existem várias definições para agentes, uma delas é a de Ferber (1999), onde afirma que agentes são entidades físicas, como robôs, aeronaves e humanos, ou virtuais, como *softwares,* que:

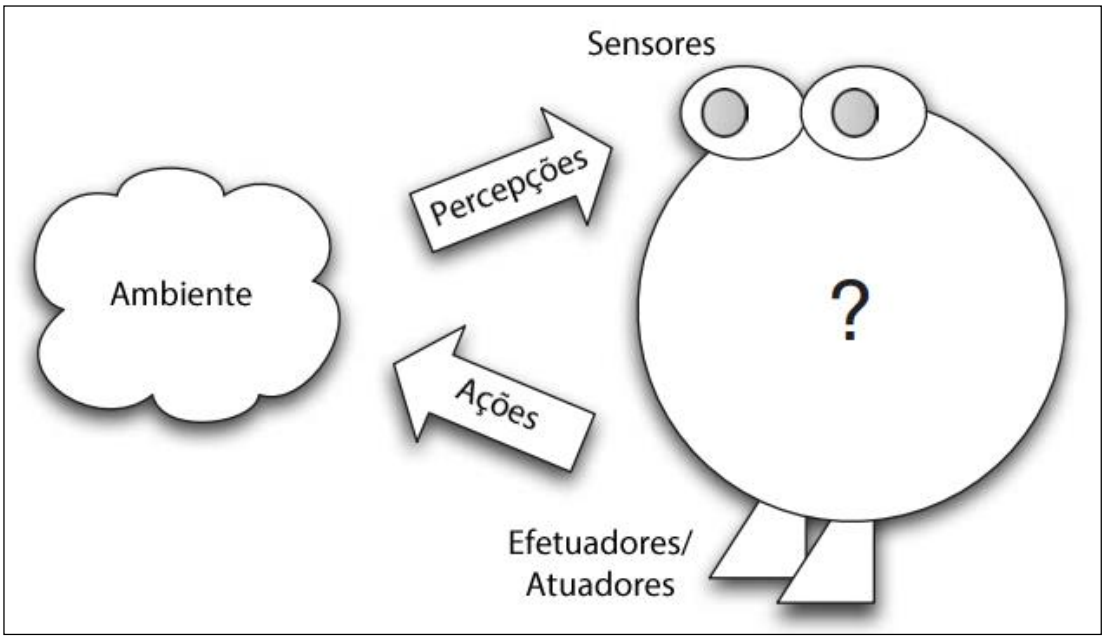
- São capazes de agir em um ambiente;
- Podem comunicar-se diretamente com outros agentes;
- São impulsionados por um conjunto de tendências (objetivos individuais ou funções de satisfação/sobrevivência) que buscam otimizar;
- Possuem apenas uma representação parcial do ambiente em que estão envoltos;
- Possuem habilidades e podem oferecer serviços;
- Podem ser capazes de reproduzir-se;
- Seus comportamentos tendem a satisfazer seus objetivos, levando em consideração seus recursos e habilidades disponíveis para isso e variando conforme as percepções, representações e comunicações recebidas.

Para Wooldridge (2009) um agente pode ser definido como um *software* que é capaz de captar fatos dentro de um ambiente e reagir aos mesmos de maneira autônoma, buscando atingir um objetivo previamente estipulado a ele. O agente

também está apto a comunicar-se com outros agentes que interagem no mesmo ambiente.

A Figura 2 disponibiliza uma visão geral de um agente, onde é possível observar que o agente possui sensores que lhes dão uma perspectiva parcial do ambiente. Por meio desta percepção parcial do ambiente o agente pode tomar decisões, utilizando seus mecanismos internos e agir no ambiente por intermédio de seus atuadores.

Figura 2 – Esquema de agente e ambiente



Fonte: Adaptado de Wooldridge (2009).

A dificuldade de uma definição absoluta de agente parte do fato de que, para diferentes campos de aplicação, os atributos associados ao conceito de agente assumem um grau diferente de importância (WOOLDRIDGE, 1999). Por exemplo, em alguns campos de aplicação a capacidade de aprender é importante, no entanto para outros campos esta capacidade pode ser indesejável e até mesmo prejudicial.

Apesar de todas as divergências na definição do conceito de agente existe o consenso em afirmar que autonomia é uma questão central nesta área (WOOLDRIDGE, 2009). Concisamente, agentes normalmente possuem as seguintes características: autonomia, proatividade, reatividade e habilidade social.

3.1.1 Autonomia

A seguinte analogia provê uma noção do que é autonomia no contexto de agentes: autonomia é um espectro, o qual possui em um extremo um ser humano com autonomia completa e no outro extremo um *software* que não possui autonomia. Entre estes dois extremos existem uma entidade capaz de decidir, por ela mesma, como atingir seus objetivos. Essa entidade não possui uma autonomia igual a de um ser humano, pois não pode escolher quais objetivos deseja, porém pode decidir quais ações tomar para atingir seus objetivos (WOOLDRIDGE, 2009).

Outras técnicas de programação, como programação orientada à objetos, sempre seguem uma sequência de regras pré-definidas para realizar uma mesma tarefa, visto que não possuem um controle sobre seu comportamento, portanto não possuem autonomia. Já um agente autônomo pode tomar decisões diferentes em uma mesma situação, pois o seu comportamento é autônomo e, portanto, não determinístico. Por exemplo, dada uma mesma situação, um agente pode apresentar comportamentos diferentes em diferentes execuções.

3.1.2 Proatividade

Segundo Wooldridge (2009) proatividade é a capacidade de tomar a iniciativa de executar ações que levem a satisfazer suas necessidades e atingir seus objetivos. Portanto, se um objetivo for delegado a um agente espera-se que ele tente alcançar este objetivo de alguma maneira, contrastando com outras técnicas de programação como Orientação à Objetos, onde um objeto só executa uma ação quando esta ação é invocada por um terceiro.

3.1.3 Reatividade

Reatividade é a capacidade de perceber mudanças em seu ambiente e reagir a partir delas com o intuito de atingir seus objetivos. Esta característica está presente em muitos sistemas computacionais (WOOLDRIDGE, 2009).

Segundo Pnueli (1986), sistemas reativos são sistemas que não podem ser definidos pela visão relacional ou funcional, visão que considera programas como

funções de estados iniciais para estados terminais. Sistemas que tem características reativas mantem uma interação com seu ambiente e, portanto, devem ser descritos em termos de seu comportamento continuo.

3.1.4 Habilidade Social

Habilidade social diz respeito a capacidade de um agente interagir com outros agentes, incluindo humanos, com o intuito de atingir seus objetivos. Esta característica é definida pela troca de informações (comunicação), habilidade de colaborar com outros agentes, oferecendo serviços (cooperação) e capacidade de assumir compromissos, resolver conflitos e chegar a acordos com outros agentes para assegurar a realização de seus objetivos (negociação) (ALONSO *et al.*, 2008).

3.2 CLASSIFICAÇÕES DE AGENTES

Russel e Norvig (2010) afirmam que agentes podem ser classificados em quatro tipos básicos:

- Agentes reativos simples: selecionam ações com base na percepção atual, ignorando o resto do histórico de percepções;
- Agentes reativos baseados em modelos: possuem uma observação parcial do ambiente, mas ainda assim, por meio de modelagens internas, conseguem acompanhar ações que não podem observar. Estas modelagens internas do ambiente fazem uso do histórico de percepções;
- Agentes baseados em objetivos: controla o estado do ambiente e seus objetivos para escolher as ações que levaram aos seus objetivos;
- Agentes baseados na utilidade: utilizam funções de utilidades, as quais medem suas preferências entre estados do mundo e a partir destas funções de utilidades escolhem que ações tomarem.

3.3 ARQUITETURAS INTERNAS DE AGENTES

Arquiteturas internas de agentes dizem respeito às suas estruturas e operações interiores, as maneiras como os agentes são construídos. Existem algumas

classes de arquiteturas: (i) agentes baseados em lógica: nas quais a tomada de decisão é realizada através de dedução lógica; (ii) agentes reativos: nas quais a tomada de decisões é implementada na forma de mapeamento direto de uma situação para outra; (iii) arquiteturas em camadas: nas quais a tomada de decisão é realizada através de várias camadas de *software*, onde cada uma explícita o raciocínio sobre o meio ambiente em diferentes níveis de abstração; e (iv) *belief-desire-intention*: em que a tomada de decisão depende da manipulação de estruturas de dados que representam as crenças, desejos e intenções do agente (WOOLDRIDGE, 1999).

Esta última classificação de arquitetura é utilizada para implementar os agentes propostos neste trabalho e é explanada de maneira mais abrangente a seguir.

3.3.1 Arquitetura BDI

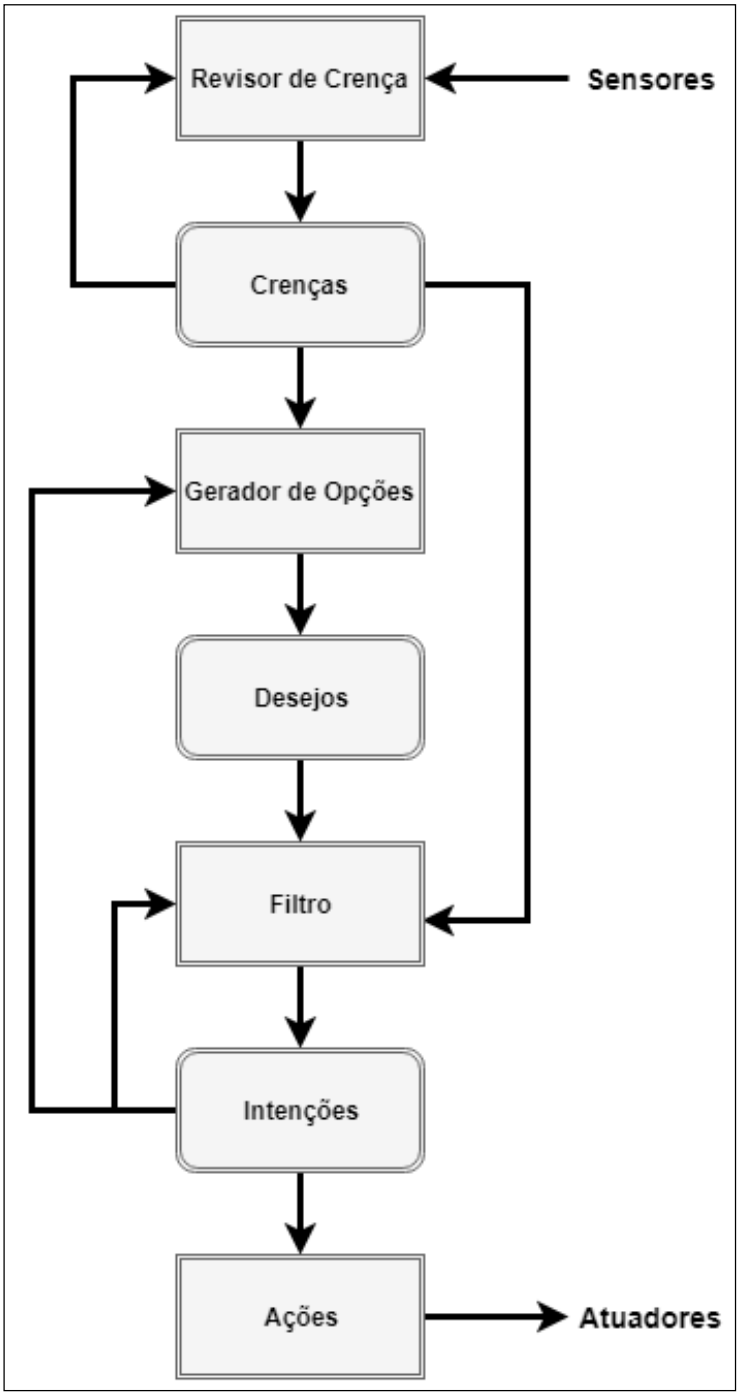
BDI é a sigla para *Belief, Desire and Intention*, em português crença, desejo e intenção. Este tipo de arquitetura de agentes é baseada no modelo do comportamento humano desenvolvido por filósofos, modelo este que teve origem com Bratman (1987), e tem foco no papel nas intenções e no raciocínio prático humano.

Segundo Bordini, Hübner e Wooldridge (2007), conforme citado agentes BDI são baseados na concepção de crença, desejo e intenção, que são os estados mentais e podem ser definidos da seguinte maneira:

- Crença (*belief)*: são informações que o agente possui sobre o mundo. É importante ressaltar que estas informações não necessariamente estão corretas, elas podem estas desatualizadas ou imprecisas;
- Desejo (*desire):* são ações que o agente gostaria de realizar. Possuir um desejo não necessariamente indica que o agente irá tomar atitudes para realizá-lo, mas é uma influência em potencial. De modo sucinto atingir um estado desejável é opcional para um agente e não uma obrigação. Naturalmente é comum que agentes possuam desejos conflitantes;
- Intenções (*intention):* são ações que o agente já decidiu tomar, por meio das quais pretende atingir seus objetivos. O agente analisa suas opções de ações e escolhe entre elas quais contribuirão para atingir seus objetivos, no momento que estas opções são escolhidas tornam-se intenções do agente.

A Figura 3 apresenta uma visão geral das etapas da tomada de decisões em uma arquitetura BDI.

Figura 3 – Visão geral arquitetura BDI



Fonte: Adaptado de Wooldridge (1999).

Os componentes da arquitetura BDI, apresentadas na Figura 3, são definidas da seguinte forma (WOOLDRIDGE, 1999):

- Revisor de Crença: função de revisão de crença (*Belief Revision Function*) que recebe as percepções e as crenças atuais do agente e, por meio delas, assume um novo conjunto de crenças;
- Crenças: conjunto de crenças atuais que o agente tem sobre o ambiente;
- Gerador de opções: determina os desejos com base nas crenças e intenções atuais;
- Desejos: um conjunto de desejos atuais, representando possíveis cursos de ações disponíveis para o agente;
- Filtro: representa o processo de deliberação do agente e determina as novas intenções com base nas suas crenças, desejos e intenções atuais;
- Intenções: um conjunto de intenções atuais, representando o foco atual do agente;
- Ação: determina uma ação a executar com base nas intenções atuais.

3.4 SISTEMAS MULTIAGENTES

Conhecendo o conceito de agente é possível utiliza-lo de maneira a solucionar problemas complexos de forma distribuída, onde agentes situados em um ambiente podem cooperar uns com os outros, por meio das suas habilidades sociais, afim de solucionar problemas que extrapolam suas capacidades individuais. Sistemas que possuem estas características são conhecidos como Sistemas Multiagentes (SMAs).

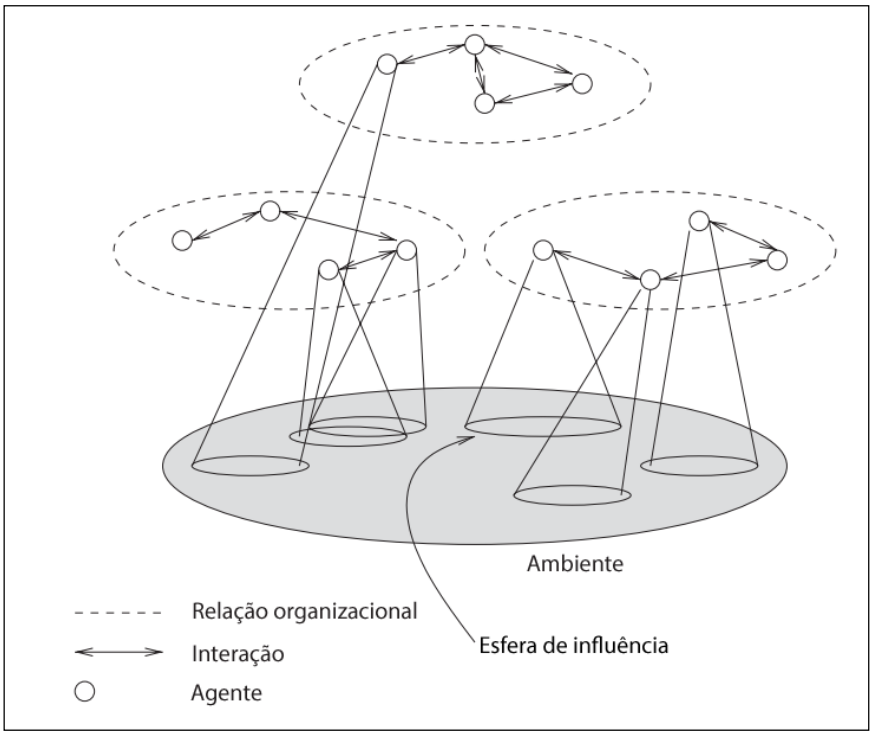
Segundo Ferber (1999) SMAs são sistemas compostos pelos seguintes elementos:

- Um ambiente *E*, isto é, um espaço que dispões em geral de uma métrica;
- Um conjunto de objetos *O*. Estes objetos são situados, isto é, para todo objeto, é possível, em um dado momento, associar uma posição em *E*. Estes objetos são passivos, eles podem ser percebidos, criados, destruídos e modificados pelos agentes;
- Um conjunto *A* de agentes, que são objetos particulares ($A \subseteq O$), os quais representam as entidades ativas do sistema;
- Um conjunto de relações *R* que une os objetos (e agentes) entre eles;
- Um conjunto de operações *Op* que permite aos agentes de *A* perceber, produzir, consumir, transformar e manipular objetos de *O*;

- Operadores encarregados de representar a aplicação destas operações e a reação do mundo à esta tentativa de modificação, chamadas de leis do universo.

A Figura 4 mostra em uma visão geral o funcionamento de um SMA, onde é possível observar vários agentes e suas esferas de influência, que são porções de ambiente as quais eles controlam totalmente ou parcialmente. Em SMAs é comum a sobreposição de esferas de influências, portanto um agente deve levar em consideração as ações de outros agentes para tomar decisões (BORDINI; HÜBNER; WOOLDRIDGE, 2007).

Figura 4 – Estrutura de um SMA



Fonte: Adaptado de BORDINI, HÜBNER e WOOLDRIDGE (2007).

Ainda na Figura 4, é possível observar os agentes situados acima do ambiente, interagindo uns com os outros por meio de suas habilidades sociais. Também é possível perceber que no SMA existem relações organizacionais, que são as esferas de atuação social dos agentes. Sendo assim é possível que dois agentes nunca interajam entre si no sistema quando estes dois não estão na mesma esfera de atuação social.

3.5 FERRAMENTAS

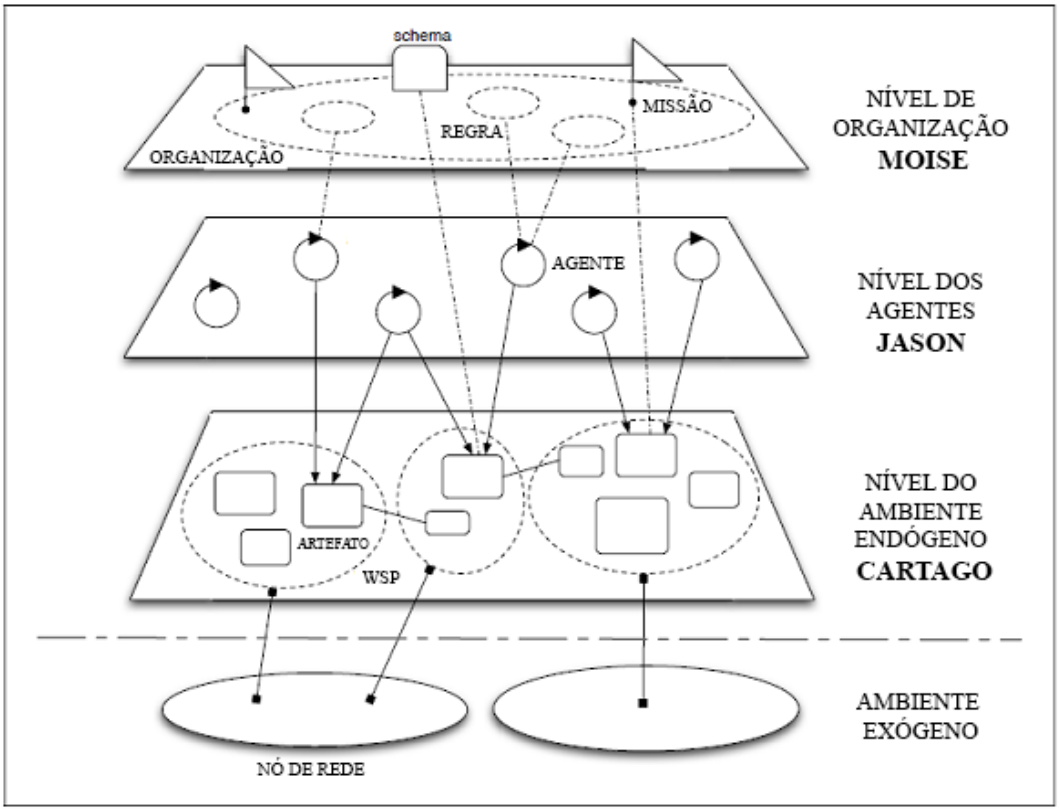
Para o desenvolvimentos do presente trabalho algumas ferramentas foram utilizadas. Esta seção aborda duas ferramentas utilizadas no decorrer deste trabalho.

3.5.1 Framework *JaCaMo*

O JaCaMo é um *framework* especifico para a programação de SMAs baseados na arquitetura BDI. Este *framework* é composto de três plataformas independentes: (i) *Jason*: utilizado para programação dos agentes autônomos com arquitetura BDI; (ii) *Cartago*: utilizado para programação do ambiente compartilhado-distribuído baseado em artefatos, onde os agentes estão situados; e (iii) *Moise:* utilizado para programação da organização do SMA, onde as regras sociais são definidas (BOISSIER *et al.*, 2011).

A Figura 5 traz uma visão geral de como estas três plataformas trabalham de maneira integrada para compor o *framework* JaCaMo.

Figura 5 – Visão geral *framework* JaCaMo



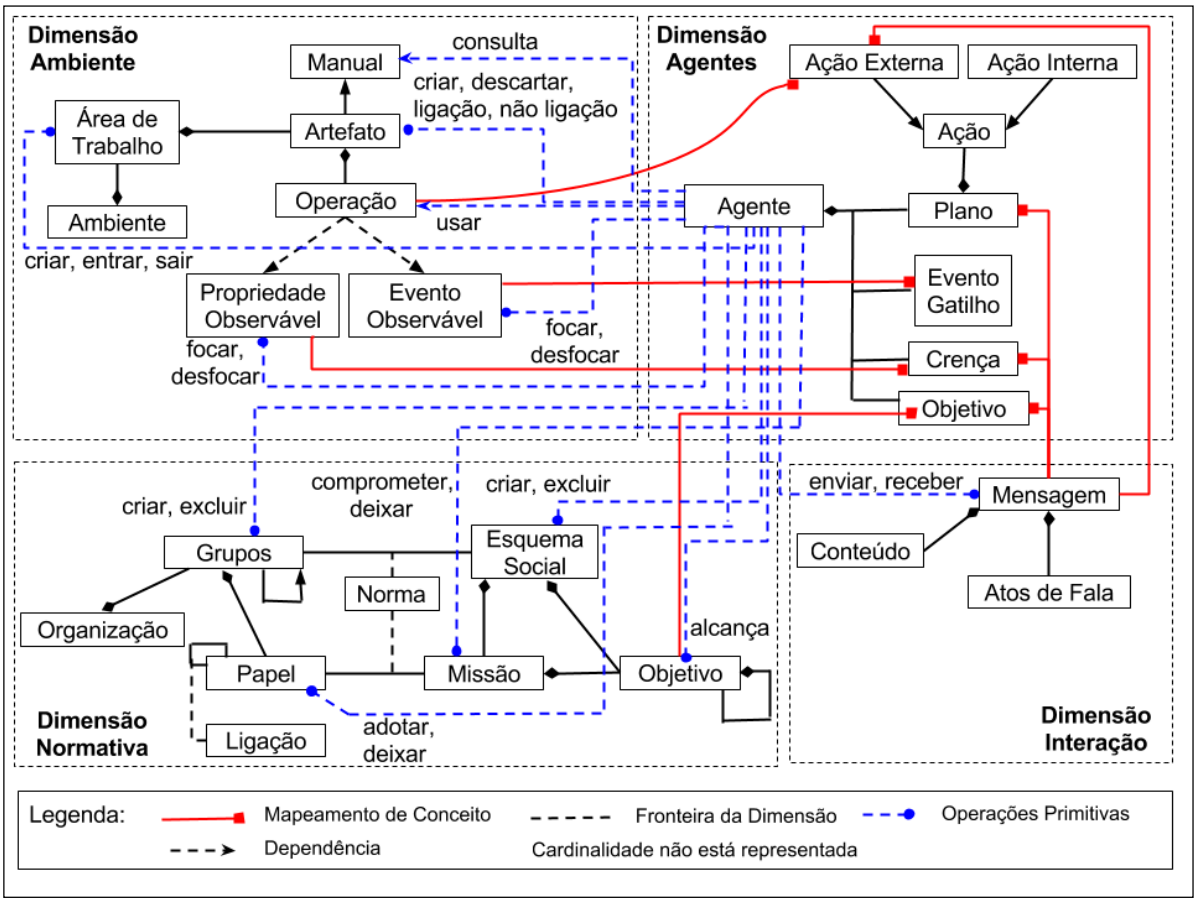
Fonte: Adaptado de JaCaMo (2011).

Na Figura 5 é possível observar os três níveis de abstração que constituem o *framework* JaCaMo *(Jason, Moise e Cartago)* e a maneira como estes se relacionam para compor um SMA. Cada um destes níveis de abstração é composto por uma das plataformas já citadas. Também é possível notar um nível ainda não citado, o ambiente exógeno, que é definido pelos componentes externos com os quais o SMA interage.

Cada uma das três plataformas independentes que compõem o *framework* JaCaMo tem seu próprio conjunto de abstrações e meta-modelos. O JaCaMo define um meta-modelo global de programação que considera todas as abstrações disponíveis em cada uma das plataformas (JACAMO, 2011).

Este meta-modelo do JaCaMo tem como objetivo definir as dependências, conexões e mapeamentos conceituais das sinergias entre as diferentes abstrações das três plataformas que compõem o framework. Na Figura 6 é possível observar um esquema do meta-modelo presente no JaCaMo.

Figura 6 – Meta-modelo JaCaMo



Fonte: Adaptado de JaCaMo (2011).

Na dimensão dos agentes (*Jason*) (Figura 6), é possível perceber que os agentes são compostos por um conjunto de crenças, desejos (objetivos) e intenções (planos), seguindo a arquitetura BDI. Na dimensão do ambiente (*Cartago*) a entidade Ambiente pode ser composta por uma ou várias Áreas de Trabalho, que por sua vez pode ser composta por um ou vários artefatos, os quais provêm um conjunto de operações e propriedades para o SMA. Os artefatos podem ser criados e dispostos dinamicamente na Área de Trabalho, o que torna possível para os agentes atualizar e adaptar a própria infraestrutura em tempo de execução.

Ainda na Figura 6 é possível observar a dimensão normativa (*Moise)*, onde é possível perceber a entidade Grupos que definem os grupos e subgrupos de agentes dentro do SMA, a entidade Missão que define os objetivos (missões) do SMA por meio do esquema social e a entidade Norma que define regras para as missões serem executadas, restringindo o comportamento dos agentes. Na dimensão da interação existe a entidade Mensagem, por meio desta os agentes podem comunicar-se uns com os outros.

3.5.1.1 *Jason*

Jason é um intérprete para uma versão estendida do *AgentSpeak*, que é uma linguagem de programação orientada a agentes. O *Jason* implementa uma semântica operacional dessa linguagem e fornece uma plataforma para o desenvolvimento de SMAs, com muitos recursos customizáveis pelo usuário. O Jason está disponível com código aberto, sendo distribuído sob licença GNU LGPL (JASON, 2011).

A linguagem *Jason* é composta por (BORDINI; HÜBNER; WOOLDRIDGE, 2007):

- Crenças: é uma coleção de literais, onde a informação é representada por meio de predicados. Abaixo um esquema de sintaxe.

  crença(informação1, ... , informaçãoN).

- Objetivos: são estados que um agente deve alcançar de maneira a atingir seu objetivo final. Existem dois tipos de objetivos, os objetivos a serem atingidos (!) e os objetivos de teste (?). Abaixo um esquema de sintaxe.

  !objetivo(informação1, ... , informaçãoN).

?objetivo(informação1, ... , informaçãoN).

- Planos: são as ações a serem tomadas para atingir seus objetivos. Possuem três partes distintas: evento gatilho, contexto e corpo. O evento gatilho define as condições para que a escolha do plano seja realizada. O contexto são as regras necessárias para que o evento seja reconhecido e o plano executado. O corpo são as ações a serem realizadas quando o plano é escolhido. Abaixo um esquema de sintaxe.

eventoGatilho : contexto ← corpo.

O trecho de código disponível na Figura 7 provê um exemplo do funcionamento do *Jason*. Este trecho imprime na saída padrão um cumprimento variando conforme a língua do país onde o agente se situa. Neste código existe a definição de cinco crenças, as quatro primeiras relacionam um país e sua respectiva forma de dizer bom dia e a quinta crença é a crença do país onde o agente se encontra. Existe um único objetivo que é cumprimentar e um único plano para atingir esse objetivo. O gatilho do plano é o objetivo *!cumprimentar* (linha 12), o contexto é o pais onde o agente se encontra e a mensagem que o agente deve imprimir conforme o pais que se encontre. O corpo do plano é uma função que realiza a impressão da mensagem na saída padrão.

Figura 7 – Trecho de código no Jason

```
1  /* Crenças Iniciais */
2  msg(fr, "Bonjour").
3  msg(br, "Bom dia").
4  msg(it, "Buongiorno").
5  msg(us, "Good morning").
6  pais(br).
7
8  /* Objetivos Iniciais */
9  !cumprimentar.
10
11 /* Planos */
12 +!cumprimentar : pais(C) & msg(C,M) <- .print(M).
```

Fonte: Adaptado de JaCaMo (2011).

Como no exemplo o agente possui uma crença de que o país que está é o Brasil (“pais(br).”) a mensagem impressa na saída padrão será “Bom dia”. Caso a crença for que o agente se encontre na Itália (“pais(it).”) a mensagem exibida será “Buongiorno” e assim sucessivamente.

3.5.1.2 *Cartago*

Cartago (Common ARTifact infrastructure for AGents Open environments) é um *framework* que permite implementar e executar ambientes virtuais baseados em artefatos, que são estruturados em *workspaces* (áreas de trabalho), onde agentes de diferentes plataformas podem trabalhar integradamente (CARTAGO, 2011).

A programação de artefatos em *Cartago* é baseada em linguagem Java com algumas funções extras, e, há duas palavras reservadas que necessitam ser destacadas: *@OPERATION* e *OpFeedbackParam*.

A palavra reservada *@OPERATION* é utilizada antes de um método Java e indica que este método pode ser utilizado como uma operação por um agente após a criação do artefato, sendo que operações são ações que podem ser executadas pelos artefatos. Portanto, é possível criar operações visíveis para os agentes e operações internas dos artefatos (CARTAGO, 2011).

A palavra reservada *OpFeedbackParam* indica um parâmetro especial de uma operação. É por meio deste parâmetro que uma agente pode enviar informações para um artefato, o qual por sua vez pode realizar alterações nesse parâmetro e retornar o parâmetro alterado para o agente.

A Figura 8 traz um exemplo simplificado de uma operação de um artefato em Cartago. Onde a operação recebe dois parâmetros: *String s* e *OpFeedbackParam<Object> num.* O parâmetro *String* s é impresso na saída padrão na linha 4. Já o parâmetro *OpFeedbackParam<Object>* num é atribuído a variável aux do tipo inteiro (linha 6) e depois acrescido de 1 (linha 7), no momento em que o método *num.set(aux + 1)* é executado o agente já possui o novo valor do parâmetro num.

Figura 8 – Trecho de código no *Cartago*

```
1    @OPERATION
2    public void operacao(String s, OpFeedbackParam<Object> num) {
3
4        System.out.println(s);
5
6        int aux = (int) num.get();
7        num.set(aux + 1);
8
9    }
```

Fonte: Autoria própria.

A utilização dos artefatos em *Cartago* pelos agentes em *Jason* só é possível pela existência de uma integração entre essas duas ferramentas. O JaCaMo provê essa integração Jason-Cartago, portanto, para que um agente possa utilizar um artefato antes é necessário que ele crie este artefato por meio da ação *makeArtifact*, como é possível visualizar na linha 1 da Figura 9.

Figura 9 – Trecho de código no *Jason*

```
1  makeArtifact(NomeArtefato, "pacote.NomeDoArtefato", [], ArtId);
2  focus(ArtId).
```

Fonte: Autoria própria.

Ainda na Figura 9 é possível perceber na linha 2 a ação *focus*, por meio da qual o agente toma conhecimento da existência do artefato e de todas as suas operações visíveis. A partir da execução da ação *focus* o agente entende que as operações do artefato fazer parte do seu conjunto de ações.

3.5.1.3 *Moise*

Moise é uma plataforma organizacional para SMAs que permite uma especificação explícita de sua organização, baseada em noções de papéis, grupos e missões (MOISE, 2011). O *Moise* é composto de três dimensões (HÜBNER; SICHMAN; BOISSIER, 2007):

- Dimensão funcional: diz respeito ao funcionamento da organização, por exemplo, a especificação de planos globais, as políticas para alocar tarefa a os agentes e a coordenação da execução dos planos;

- Dimensão estrutural: aborda um aspecto mais estático da organização, diz respeito aos papéis (relacionamentos e grupos de papéis). Nesta dimensão são definidas às restrições, obrigações e permissões que um papel impõe a um agente ou a um grupo de agentes; e
- Dimensão normativa: diz respeito às normas de alto nível que os agentes devem seguir, especifica a relação entre a dimensão funcional e a dimensão estrutural.

Para utilizar o *Moise* é necessário especificar as três dimensões em um arquivo .xml. Na Figura 10 é possível verificar um exemplo de especificação estrutural no *Moise*, da linha três a oito são definidos os papeis, no caso existem dois leiloeiro e participante. Da linha 11 a 21 é definido um grupo, o *grupoLeilao*, onde podemos perceber que este grupo pode ser composto de no mínimo um leiloeiro e zero participante e no máximo de um leiloeiro e 300 participantes (linhas 14 e 15). Na linha 18 e 19 é possível perceber a definição de uma relação de autoridade do leiloeiro sobre o participante.

Figura 10 – Trecho de código no Moise

```
1  <structural-specification>
2
3  <role-definitions>
4
5      <role id="leiloeiro" />
6      <role id="participante" />
7
8  </role-definitions>
9
10
11 <group-specification id="grupoLeilao">
12
13     <roles>
14         <role id="leiloeiro"  min="1" max="1"/>
15         <role id="participante" min="0" max="300"/>
16     </roles>
17
18    <link from="leiloeiro" to="participante" type="authority"
19     scope="inter-group" extends-subgroups="true" />
20
21 </group-specification>
22
23 </structural-specification>
```

Fonte: Adaptado de JaCaMo (2011).

A Figura 11 traz um exemplo de especificação funcional e normativa utilizando *Moise*. Da linha 26 à 45 é apresentado o esquema do SMA, que é definido como uma árvore de decomposição de uma meta global, onde a decomposição é feita em planos (HÜBNER; SICHMAN; BOISSIER, 2010). Nas linhas 28 à 34 é possível observar a definição do objetivo geral da organização, que se subdivide em outros objetivos *(iniciar, lance e decidir)*. Também é possível observar a obrigatoriedade dos objetivos serem realizados de maneira sequencial pela especificação *sequence* na linha 29. Da linha 26 a 39 é especificado a missão do leiloeiro, que se subdivide nos objetivos *iniciar* e *decidir*. Da linha 41 a 43 é especificada a missão do participante, que possui apenas o objetivo lance.

Figura 11 – Trecho de código no Moise

```
25  <functional-specification>
26      <scheme id="fazerLeilao">
27
28          <goal id="leilao">
29              <plan operator="sequence">
30                  <goal id="iniciar"/>
31                  <goal id="lance"/>
32                  <goal id="decidir"/>
33              </plan>
34          </goal>
35
36          <mission id="missaoLeiloeiro" >
37              <goal id="iniciar" />
38              <goal id="decidir" />
39          </mission>
40
41          <mission id="missaoParicipante" >
42              <goal id="lance" />
43          </mission>
44
45      </scheme>
46  </functional-specification>
47
48  <normative-specification>
49      <norm id="n1"  type="obligation"
50            role="leiloeiro"   mission="missaoLeiloeiro"/>
51      <norm id="n2"  type="permission"
52            role="participante"  mission="missaoParicipante" />
53  </normative-specification>
```

Fonte: Adaptado de JaCaMo (2011).

Ainda na Figura 11 é possível observar a especificação normativa (linha 48 a 53). Existem duas normas definidas, a primeira especifica que o leiloeiro deve

obrigatoriamente executar a missão *missaoLeiloeiro* e a segunda especifica que o participante pode ou não executar a missão *missãoParticipante*.

3.5.2 Prometheus

Prometheus é uma metodologia para o desenvolvimento de SMAs, por meio desta é possível documentar a especificação, design e implementação de agentes inteligentes desde o início do processo de desenvolvimento do SMA até o final. Esta metodologia é composta de três partes Winikoff e Padgham (2004):

- Especificação do sistema: identificar objetivos dos agentes, desenvolver casos de uso, descrever ações, percepções e interações dos agentes;
- Design arquitetural: definir o grupo de funcionalidades que determinam o tipo de dados que o agente utiliza, os tipos de agente, análise e elaboração de diagramas de visão geral do SMA; e
- Design detalhado: desenvolver diagramas de processos, diagramas de análise dos agentes e definir detalhes dos eventos, planos e crenças.

3.5.3 SUMO

O SUMO é uma ferramenta de simulação de código livre, desenvolvida pelo Instituto de Transporte da Alemanha com o intuito de aprimorar os resultados finais de um projeto e suas análises, além de facilitar os testes de novos algoritmos que são desenvolvidos nessa área (KRAJZEWICZ et al., 2012).

O simulador oferece variadas funcionalidades ao usuário, como simulação de veículos, pedestres, bicicletas, transporte público; controle de tempo de semáforos; importação de ambientes reais através da ferramenta *NETCONVERT* em conjunto com *OpenStreetMaps*; *TraCI*, que permite obter valores da simulação e manipulá-los em tempo real, dentre outras possibilidades.

A simulação do SUMO têm como entrada as rotas (*routes*) e uma rede (*network*), que é o ambiente onde a simulação ocorre, descrita em arquivo de formato *XML*. Uma rede é composta por nós (*nodes* ou cruzamentos), vias (*edges*), tipos (*types*), tanto de vias como de veículos, e conexões (*connections*) entre as faixas de uma via.

3.6 NEGOCIAÇÃO ENTRE AGENTES

Em SMAs os agentes interagem entre si com o objetivo de solucionar problemas e atingir seus objetivos. Um dos tipos de interação é a negociação, utilizada para solucionar situações onde os agentes possuem objetivos conflitantes.

Negociação é processo pelo qual uma decisão conjunta é tomada por duas ou mais partes. Onde as partes primeiramente verbalizam demandas contraditórias e, em seguida, avançam para o acordo por um processo de concessão ou busca de novas alternativas (HUHNS; STEPHENS, 1999). Segundo Wooldridge (2009) toda configuração de negociação é composta de quatro componentes:

- Um conjunto de negociações, que representa o espaço de possíveis propostas que os agentes podem fazer;
- Um protocolo, que define as propostas legais que os agentes podem fazer;
- Uma coleção de estratégias, uma para cada agente, que determinam o que os agentes farão. Estas estratégias normalmente são privadas, isto é, não são visíveis para outros agentes participantes da negociação; e
- Uma regra que determina quando um acordo foi atingido e o que é esse acordo.

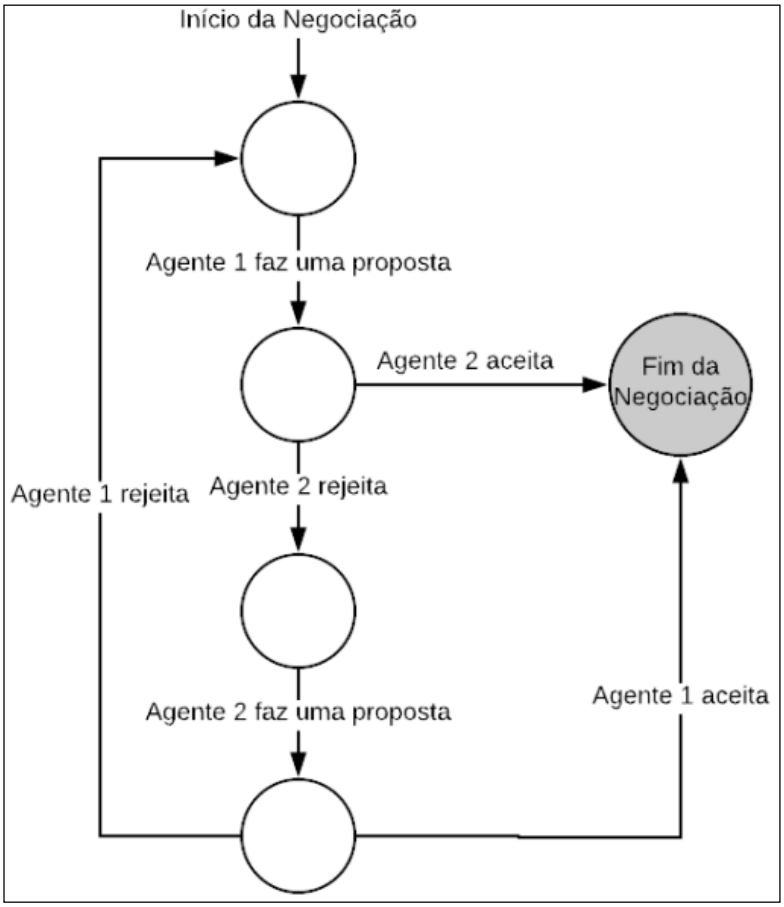
Negociações acontecem normalmente com os agentes trocando propostas em uma série de rodadas, com uma proposta sendo realizada a cada rodada, até chegarem a um acordo definido na regra de negociação ou interrompam a negociação. A proposta que um agente faz é definida pelo seu conjunto de estratégias e deve ser válida, como definido pelo protocolo. Quando um acordo é fechado, conforme definido pela regra de acordo, a negociação termina com o contrato do acordo (WOOLDRIDGE, 2009). O autômato ilustrado na Figura 12 representa essas rodadas da negociação.

A complexidade dos processos de negociação entre os agentes cresce conforme a quantidade de atributos e a quantidade de agentes envolvidos, pois o crescimento desses fatores gera um problema exponencial (WOOLDRIDGE, 2009).

Uma negociação simples se dá quando dois agentes estão negociando apenas sobre um atributo de algum recurso, como o valor pago por uma vaga de

estacionamento. Esta é uma negociação simples de ser analisada e encontrar as concessões que cada agente deve fazer por ter apenas um atributo para um produto, onde o agente vendedor quer que o preço aumente e o agente comprador quer o menor preço possível.

Figura 12 – Autômato de uma negociação



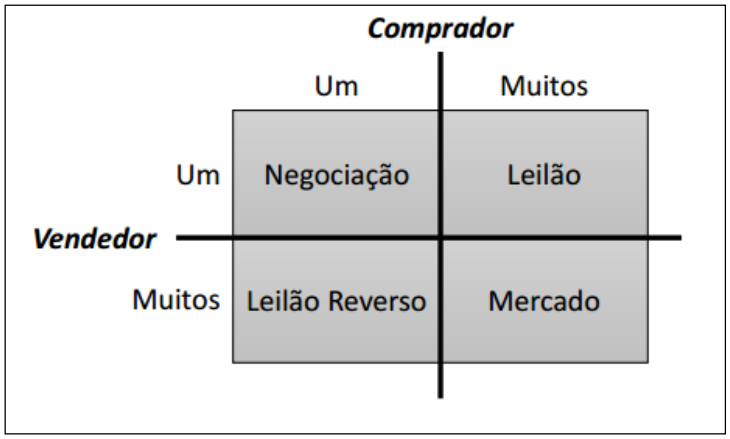
Fonte: Adaptado de Wooldridge (2009).

Uma negociação de múltiplos atributos gera um crescimento exponencial de possibilidades para a conciliação, sendo complexo analisar quais são as concessões que devem ser feitas pelos agentes para alcançar o acordo. Um exemplo é a compra de um imóvel, onde não somente o preço é o atributo da negociação da compra. O preço até pode ser o principal, mas a localização, vizinhança, garagem, distribuição dos cômodos, entre outros atributos também são analisados (WOOLDRIDGE, 2009).

Em relação a quantidade de agentes em uma negociação, além da classificação já apresentada na introdução, existe outra mais abrangente, onde generalizando pela quantidade de agentes envolvidos é possível definir quatro configurações possíveis: (i) negociação bilateral, um vendedor e um comprador

negociam diretamente; (ii) negociação multilateral 1-N (leilão clássico), muitos compradores e um vendedor negociam; (iii) negociação multilateral N-1 (leilão reverso), muitos vendedores e um comprador negociam; e (iv) negociação multilateral N-M (mercado), muitos compradores e muitos vendedores negociam. A Figura 13 apresenta um esquema desta classificação de negociação.

Figura 13 – Classificação de negociação



Fonte: (XIE; LI; SUN, 2004).

Segundo Huhns e Stephens (1999), para um mecanismo de negociação ser considerado ideal ele deve ser:

- Eficiente: os agentes não devem desperdiçar recursos para chegar a um acordo;
- Estável: nenhum agente deve desviar-se de sua coleção de estratégias;
- Simples: o mecanismo de negociação não deve impor uma sobrecarga aos agentes;
- Distribuído: o mecanismo de negociação não deve possuir uma entidade centralizada de decisão;
- Simétrico: o mecanismo de negociação não deve ser tendencioso para nenhum agente por razões arbitrárias ou inapropriadas.

3.6.1 Protocolos de negociação

Um protocolo de negociação é um conjunto de regras que especifica a gama de movimentos legais disponíveis para cada agente em qualquer fase de um processo

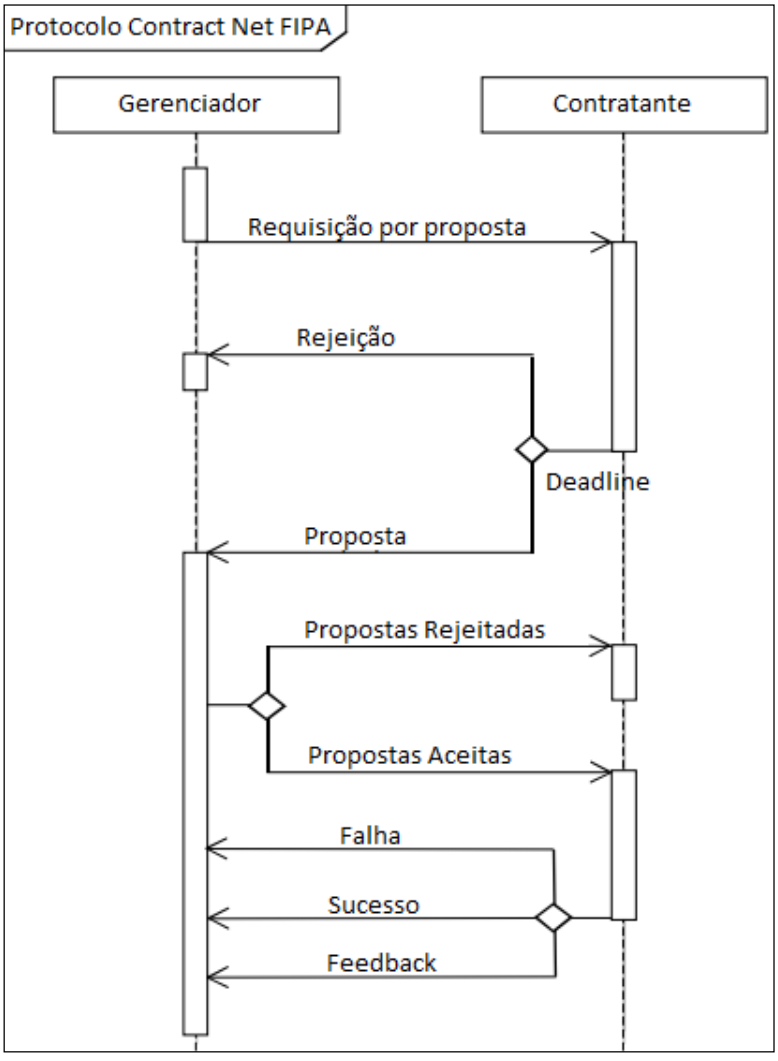
de negociação (ENDRISS, 2006). Deste modo, o protocolo de negociação define o que os agentes podem fazer em uma negociação e quando eles podem fazer.

A seguir três protocolos de negociação são apresentados, bem como suas principais características e métodos de funcionamento.

3.6.1.1 Protocolo Contract Net

O protocolo *Contract Net* foi proposto por Smith (1988), sendo um mecanismo de negociação para sistemas distribuídos e funcionando como um protocolo de licitação, com o objetivo de alocar recursos. O protocolo utiliza uma negociação do tipo leilão (muitos para um), que permite encontrar o agente mais adequado para determinado recurso ou tarefa, comparando propostas apresentadas pelos agentes.

Figura 14 – Protocolo *Contract Net*



Fonte: Adaptado de FIPA (2002).

O diagrama de sequência, apresentado na Figura 14, indica que a negociação no protocolo *Contract Net* é iniciada quando um agente gerenciador dispara uma requisição para outros agentes buscando propostas para alocar algum recurso, especificando as condições para o negócio. Os agentes contratantes recebem essa requisição e podem apresentar suas propostas, ou recusar a negociação. Quando o *deadline* chega ao fim, o gerenciador avalia as propostas recebidas, seleciona qual agente receberá o recurso, ou cancela a negociação se não receber propostas satisfatórias. Os contratantes receberão a resposta de sua proposta, recusando ou confirmando o acordo. Ao fim da negociação o contratante envia uma mensagem ao gerenciador que pode informar o sucesso, trazer um feedback ou informar falha no negócio (FIPA, 2002).

3.6.1.2 Protocolo monotônico de concessão

O Protocolo Monotônico de Concessão (*Monotonic Concession Protocol*) é um protocolo de negociação para alocação de recursos que, formaliza uma das formas mais naturais de negociação entre duas entidades. Este protocolo tornou-se referência para mecanismos de negociação pois suas propriedades formais são claras e existem muitas formas de generalizar suas definições para contextos de negociações multilaterais (ENDRISS, 2006). As regras deste protocolo são definidas da seguinte maneira (WOOLDRIDGE, 2009):

- A negociação é realizada em uma série de rodadas;
- Na primeira rodada ambos agentes propõem um acordo por meio de seu conjunto de negociações;
- Um acordo é alcançado se dois agentes propuserem ofertas e um dos agentes descobre, por meio de mecanismos internos, que o acordo proposto pelo outro é pelo menos tão bom ou melhor que a proposta que fez;
- A regra para determinar o que é o acordo é a seguinte: se as ofertas dos dois agentes coincidem ou as duas ofertas excedem a do outro agente, uma das propostas é selecionada aleatoriamente. Se apenas uma

proposta exceder ou corresponder à proposta do outro agente, então este é o acordo selecionado;

- Se nenhum acordo for alcançado, a negociação prosseguirá para outra rodada de propostas simultâneas. Na rodada *i + 1*, nenhum agente pode fazer uma proposta que é menos satisfatória para outro agente do que a proposta da rodada *i*;
- Se nenhum dos agentes fizer uma concessão em alguma rodada *i > 0* a negociação termina com um conflito de acordo.

Este protocolo garante um fim da negociação depois de um número finito de rodadas, mesmo que sem acordo. Porém, segundo Endriss (2006) não é possível garantir que o fim da negociação se dará rapidamente, visto que o número de rodadas da negociação pode chegar até ao número de agentes envolvidos, elevado a quantidade de recursos a serem alocados.

3.6.1.3 Faratin

Protocolo proposto por Faratin, Sierra e Jennings (1998), foi desenvolvido baseado em considerações práticas e percepções decorrentes do desenvolvimento de um mecanismo de negociação de agentes para o gerenciamento de processos de negócios. Este mecanismo é orientado a serviços, onde um cliente requisita um serviço a um servidor. É importante ressaltar que os agentes podem assumir dois papeis distintos, tanto de servidor, que presta o serviço, como de cliente, que requisita o serviço.

Segundo Faratin, Sierra e Jennings (1998) o processo de negociação é uma sucessão alternada de ofertas e contraofertas. Este processo continua até ambas as partes cheguem a um acordo ou quando uma das partes desiste da negociação. Os agentes possuem um tempo limite para terminar a negociação, se este tempo for extrapolado o agente informa a o outro agente que desistiu da negociação.

Neste protocolo existe o conceito de *Thread* de negociação, que é uma sequência de troca de propostas e contrapropostas ocorridas estre dois agentes. Uma *Thread* de negociação é formalmente definida da seguinte maneira (FARATIN; SIERRA; JENNINGS, 1998):

- Um conjunto de propostas e contrapropostas;
- As propostas e contrapropostas são realizadas alternadamente entre os agentes, e, podem ser ordenadas cronologicamente;
- Cada *Thread* de negociação é exclusiva de dois agentes, onde cada *Thread* possui propostas e contrapropostas de apenas dois agentes;
- Uma *Thread* de negociação é definida como ativa quando nenhum dos agentes aceitou uma proposta ou desistiu da negociação.

Em Faratin, Sierra e Jennings (1998) também são propostas algumas táticas para a geração de propostas. Estas táticas podem ser divididas em três categorias, conforme suas variáveis de dependência:

- Tempo: utilizada quando um agente tem um prazo de tempo pelo qual um acordo deve estar em vigor, essas táticas modelam o fato de que o agente irá conceder mais rapidamente à medida que o prazo se aproximar. Uma função dependendo do tempo é o que diferencia as táticas deste conjunto;
- Recurso: utilizada quando existem recursos limitados sendo negociados. Uma função dependendo da quantidade de recurso ainda disponível é o que diferencia as táticas deste conjunto;
- Comportamento: utilizada em situações onde um agente não está sofrendo grande pressão para chegar a um acordo, na tentativa de evitar que ele seja explorado por outros agentes. As táticas são definidas baseando-se, até certo ponto, na imitação do comportamento do outro agente participante da negociação.

4 DESENVOLVIMENTO DO SISTEMA MULTIAGENTE

Neste capítulo são apresentados os passos para o desenvolvimento de um *Smart Parking,* baseado em um Sistema Multiagente com um mecanismo de negociação descentralizado, utilizando o *framework* JaCaMo. Também é apresentada a maneira como o estacionamento deve operar e como foram abstraídas as funções para o SMA. O Sistema Multiagente criado é nomeado de MAPS-OPEN, pois se trata de um SMA aberto, onde os agentes podem entrar e sair do sistema a qualquer momento e podem trocar mensagens com quaisquer agentes do sistema, sem a existência de um agente centralizador.

Segundo O'Hare e Jennings (1996), para o desenvolvimento de agentes autônomos com capacidades sofisticadas e flexíveis de negociações, é necessário a definição de três itens: i) protocolo de negociação, o qual define as ações que os agentes podem tomar em uma negociação; ii) o problema que a negociação deseja solucionar; e iii) modelo de raciocínio, o qual define as ofertas iniciais, a gama de ofertas aceitáveis, as contraofertas, quando a negociação deve ser abandonada e quando um acordo deve ser fechado.

Para o desenvolvimento do Sistema Multiagente apresentado neste trabalho, a metodologia definida por O'Hare e Jennings (1996) é seguida. O item ii da metodologia, que é a definição do problema que a negociação deseja solucionar, no presente trabalho é alocação de vagas de estacionamento. Os itens i e iii (respectivamente, protocolo de negociação e o modelo de raciocínio) são apresentados nas seções seguintes.

O restante deste capítulo está organizado da seguinte maneira, na seção 4.1 é apresentado o modelo de raciocínio, na seção 4.2 é apresentado o protocolo de negociação e na seção 4.3 é apresentada a implementação do Sistema Multiagente. As duas últimas seções apresentam os recursos adicionais desenvolvidos para o sistema, na seção 6.4 é apresentado o desenvolvimento da interface gráfica utilizada pelo MAPS-OPEN e na seção 6.5 é apresentado a conexão do Sistema Multiagente como SUMO, que é um simulador de mobilidade urbana.

4.1 MODELO DE RACIOCÍNIO

A modelagem computacional pode facilitar processos de negociação, computando uma ampla gama de alternativas e examinando seus resultados em busca de tendências (OLIVER, 1996). Para modelar computacionalmente problemas utilizando técnicas de negociação em SMA, os agentes devem ser capazes de gerar, por meio de táticas, ofertas e contraofertas. Segundo Faratin, Sierra e Jennings (1998), táticas são o conjunto de funções que determinam como calcular o quão valioso é um determinado recurso, por meio de características como preço, volume, duração, qualidade, entre outros e considerando um critério, como tempo, recursos, distância, entre outros.

Táticas são geradas por combinações lineares de funções simples, sendo que a imagem da função é o conjunto utilizado para definir o valor de um recurso perante um agente e o critério utilizado para esta definição, é o domínio da função (FARATIN; SIERRA; JENNINGS, 1998). Lembrando que, o domínio é o conjunto de todas as entradas possíveis de uma função e a imagem é conjunto de todas as saídas oriundas do conjunto das entradas. No decorrer desta seção, serão apresentadas as funções geradoras de táticas, utilizadas para implementar o protocolo de negociação que será usado no MAPS-OPEN.

O modelo de raciocínio apresentado neste trabalho assume a existência de dois tipos de agente no SMA: o agente *driver*, que é o agente que utiliza o sistema, podendo estacionar e deixar uma vaga e o agente *parkingSpotController*, responsável por controlar uma vaga. Os agentes *drivers* podem assumir dois papeis distintos: *seller*, quando o agente está deixando uma vaga e pretende vendê-la e *buyer*, quando o agente está procurando uma vaga para estacionar. Neste modelo a negociação é iniciada pelo agente *driver* no papel de *seller*, quando ele anuncia para todos os agentes dos sistema que está deixando uma vaga e deseja vendê-la (DUCHEIKO; ALVES, 2018).

Os *drivers buyer* que recebem esta mensagem de um *seller*, informando que a vaga está disponível para venda, vão responder enviando uma proposta de compra da vaga. A proposta de compra enviada para o *seller* é gerada com base na Função 1, apresentada a seguir e desenvolvida pelos autores deste trabalho. Os *drivers* que recebem a oferta de vaga do *seller* e não estão procurando uma vaga ignoram a oferta. No modelo de raciocínio são utilizadas unidades gerais de medida e também de valor

de venda do recurso. Isso foi determinado para generalizar as unidades do modelo e não atribuir uma medida ou valor específico de sistema métrico ou monetário.

$$p = \lambda - \frac{Dist(D,\ V)}{\alpha}$$

Função 1

onde:

• p = proposta do agente no papel de *buyer*, em unidade de venda;

• $Dist$ = distância entre dois pontos, em unidade de medida;

• D = ponto da localização onde o *driver buyer* deseja obter a vaga;

• V = ponto onde se localiza a vaga que o *driver seller* pretende negociar;

• λ = valor máximo que um driver pode pagar por uma vaga; e

• α = distância máxima entre os pontos D e V, que um *driver buyer* aceita negociar a vaga (sendo, $\alpha > 0$).

A Função 1, tem como imagem o conjunto de valores que um *buyer* está disposto a pagar por uma vaga e como domínio o conjunto das distâncias entre o ponto onde o agente deseja a vaga (D) e o ponto da vaga que está sendo negociada (V). A Função 1 garante que, quanto mais próxima a vaga é do local onde *buyer* deseja estacionar, maior é o valor que ele estará disposto a pagar. Portanto, a função varia conforme a proximidade entre a vaga que está sendo ofertada e o local onde o *buyer* deseja a vaga.

Por exemplo, sendo λ o valor máximo que um *driver* pode pagar para obter uma vaga igual a 10 e α (distância máxima entre os pontos D e V que um *driver* aceita) um valor qualquer. Assumindo os valores descritos, um *driver buyer* irá pagar para o *driver seller* λ unidades de venda, no caso 10 unidades, quando distância entre ponto desejado (D) e o ponto da vaga (V) é igual a zero.

Um *driver* não aceitará uma vaga onde a distância entre os pontos D e V for maior que $\alpha \times 10$, pois neste caso o valor que um *buyer* irá pagar pela vaga é negativo e logicamente, o *driver seller* não irá aceitar um valor negativo. Portanto, α limita a distância máxima entre os pontos D e V. A proposta enviada pelo *buyer* para

o *seller* deve ser analisada, isto acontece por meio da Função 2, que foi desenvolvida com base no trabalho de Faratin, Sierra e Jennings (1998).

$$c = \begin{cases} Se\ p\ \geq m\ \ Então\ ACEITAR \\ Se\ p \leq \dfrac{m}{2}\ Então\ REJEITAR \\ Se\ não\ \ \ envia\ CP(m) \end{cases}$$

Função 2

Onde:

- c = contraproposta do agente *driver seller*;
- p = proposta do agente *buyer*;
- m = Média do Valor de Venda; e
- $envia\ CP()$ = função que envia uma contraproposta para o *buyer*.

A Função 2 é utilizada para analisar a proposta do *driver buyer,* com base no valor médio de venda da vaga que está sendo negociada. A variável m é justamente a média do valor de venda da vaga que está sendo negociada e será obtido por meio de um histórico contendo as últimas negociações realizadas com sucesso, que cada agente *parkingSpotController* do sistema possui registrado. Inicialmente, quando nenhum agente estacionou em uma vaga, o valor médio de venda da vaga é definido ao iniciar o sistema.

Com base na Função 2, a proposta recebida é aceita e o acordo é fechado se a proposta for maior ou igual ao valor médio de venda das vagas próximas e rejeitado se for menor que a metade do valor médio das vagas próximas, em ambos os casos a negociação entre estes dois agentes acaba. Caso contrário, uma contraproposta é gerada com base no valor médio (m) de venda da vaga e enviada para o agente *buyer*. Se o agente *driver buyer* receber esta contraproposta, a Função 3 é utilizada para avaliar a mesma.

$$u = \begin{cases} Se\ \ |p - c| \leq \delta *\ p\ \ Então\ ACEITAR \\ Se\ não\ \ REJEITAR \end{cases}$$

Função 3

onde:

- u = Ultimato, proposta final;
- p = proposta inicial do *buyer*;
- c = contraproposta recebida do *seller*; e
- δ = variação da faixa de fechamento de acordo (sendo, $0 < \delta \leq 1$).

A Função 3, desenvolvida pelos do presente trabalho, garante um fim a negociação, visto que, ou a proposta é aceita e os agentes entram em um acordo, ou a proposta é rejeitada sem que nenhum acordo seja fechado. A proposta é aceita quando, a diferença entre a proposta inicial da negociação (p) e a contraproposta recebida (c) é inferior a δ. O valor de δ é variável, quanto mais próximo de 1, maior a gama de contrapropostas que serão aceitas e quanto mais próximo de 0, menor a gama de contraproposta que serão aceitas.

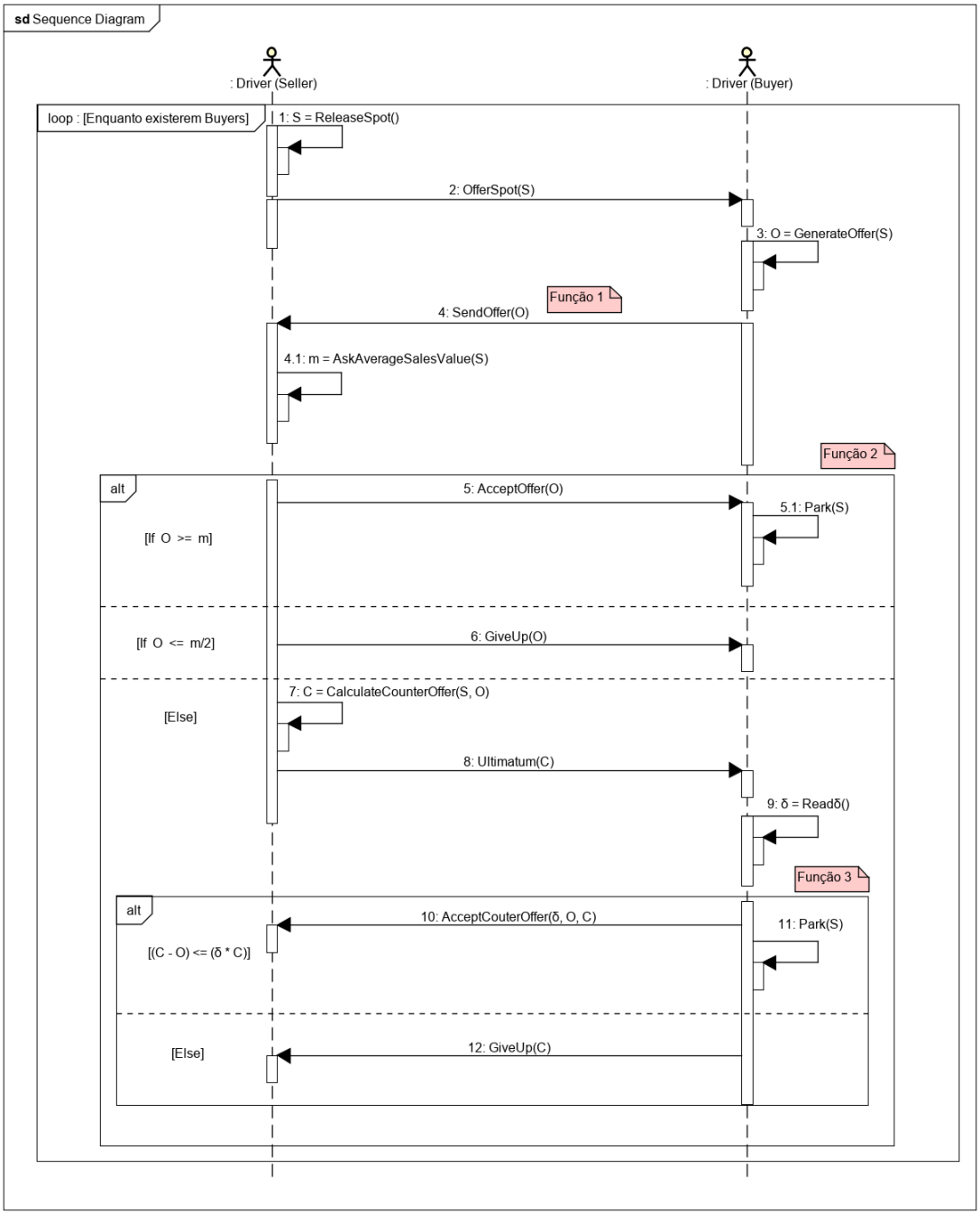
As negociações entre o agente *seller* e *buyer*, explanadas nos parágrafos anteriores, podem acontecer entre um agente *seller* negociando com vários *buyers* e estes buyers podem estar negociando com vários outros agentes *sellers,* uma vez que é uma negociação descentralizada. O agente *seller* irá fechar acordo com o primeiro agente *buyer* que lançar uma oferta aceitável pelo modelo de raciocínio. Porém, em um mesmo instante um agente só negocia com um outro agente, para evitar inconsistências no sistema. Por exemplo, um agente vender uma mesma vaga para dois agentes ao mesmo tempo.

Caso o agente *seller* não consiga fechar um acordo com nenhum *buyer*, o *seller* aguarda por um determinado período de tempo (o qual deverá ser estabelecido antes de iniciar o sistema) e envia uma nova mensagem ofertando a vaga para todos os *drivers* do sistema. Após um número de tentativas de venda sem sucesso, o agente *seller* desiste de vender a vaga e libera a mesma. Este número de tentativas é definido antes de iniciar a execução do sistema. Lembrando que, se trata de um SMA aberto onde os agentes podem entrar e sair do sistema, portanto possivelmente novos agentes aptos a fechar um acordo podem chegar ao estacionamento desejando estacionar.

4.2 PROTOCOLO DE NEGOCIAÇÃO

A Figura 15 apresenta o protocolo de negociação proposto no presente trabalho, o qual é utilizado para implementar o SMA para alocação de vagas de estacionamento, com mecanismo de negociação descentralizado. O protocolo descrito nesta seção utiliza como base o modelo de raciocínio apresentado na seção anterior.

Figura 15 – Diagrama do Protocolo de Negociação



Fonte: Autoria Própria.

Este protocolo define que, sempre a negociação deverá ser iniciada pelo agente *seller*, informando os outros *drivers* do sistema que possui uma vaga para negociar (na Figura 15, mensagem *2:OfferSpot(S)*). Então, os agentes *drivers* que recebem esta mensagem e estão procurando uma vaga (agentes *buyers*) geram uma proposta, utilizando a Função 1, e a enviam para o *driver seller* (na Figura 15, mensagem *4:SendOffer(O)*). Depois, o *seller* analisa estas propostas utilizando a Função 2, podendo tomar as seguintes decisões: i) aceitar a proposta (na Figura 15, mensagem *5:AcceptOffer(O)*), onde a negociação acaba e o acordo é fechado; ii) rejeitar proposta (na Figura 15, mensagem *6:GiveUp(O)*), onde a negociação entre os dois agentes acaba; ou iii) enviar contraproposta (na Figura 15, mensagem *8:Ultimatum(C)*), onde o agente *seller* gera uma contraproposta, ainda utilizando a Função 2, e envia para o agente *buyer*.

Quando o agente *buyer* recebe esta contraposta ele a analisa, utilizando a Função 3, podendo tomar duas ações: i) aceitar a contraproposta (na Figura 15, mensagem *10:AcceptCounterOffer(C))*, onde um acordo entre os dois *drivers* é fechado e a negociação acaba; ou ii) rejeitar contraproposta (na Figura 15, mensagem *10:GiveUp(C))*, onde a negociação entre estes dois agentes acaba e o agente *seller* inicia uma negociação com outro *buyer.* Este processo se repete até que não haja mais agentes para negociar.

É importante ressaltar que, no início do sistema nenhum agente será dono de uma vaga, todas as vagas estarão livres e não existe agente para vender a vaga. Neste caso, a alocação se dá de maneira simples, o agente *buyer* estaciona na primeira vaga livre que está dentro da distância máxima aceitável pelo agente. Após um este agente *buyer* estacionar e permanecer o tempo estabelecido na vaga ele assume o papel de *seller*, iniciando o processo de negociação com outros agentes, tentando vender a vaga que está deixando para outro agente *buyer.*

4.3 IMPLEMENTAÇÃO DO SISTEMA MULTIAGENTE

Esta seção descreve a implementação do sistema proposto nas seções anteriores deste trabalho, utilizando o Modelo de Raciocínio e o Protocolo de negociação apresentados, os primeiros resultados desta implementação são

apresentados em Ducheiko, Borges e Alves (2018). Para implementar o sistema foram utilizados as três camadas do framework JaCaMo. Na camada do Jason foram definidos dois tipos de agentes, os agentes drivers que utilizam o sistema e negociam vagas entre si e os agentes parkingSpotController, que possuem um artefato Spot relacionado e controlam o mesmo.

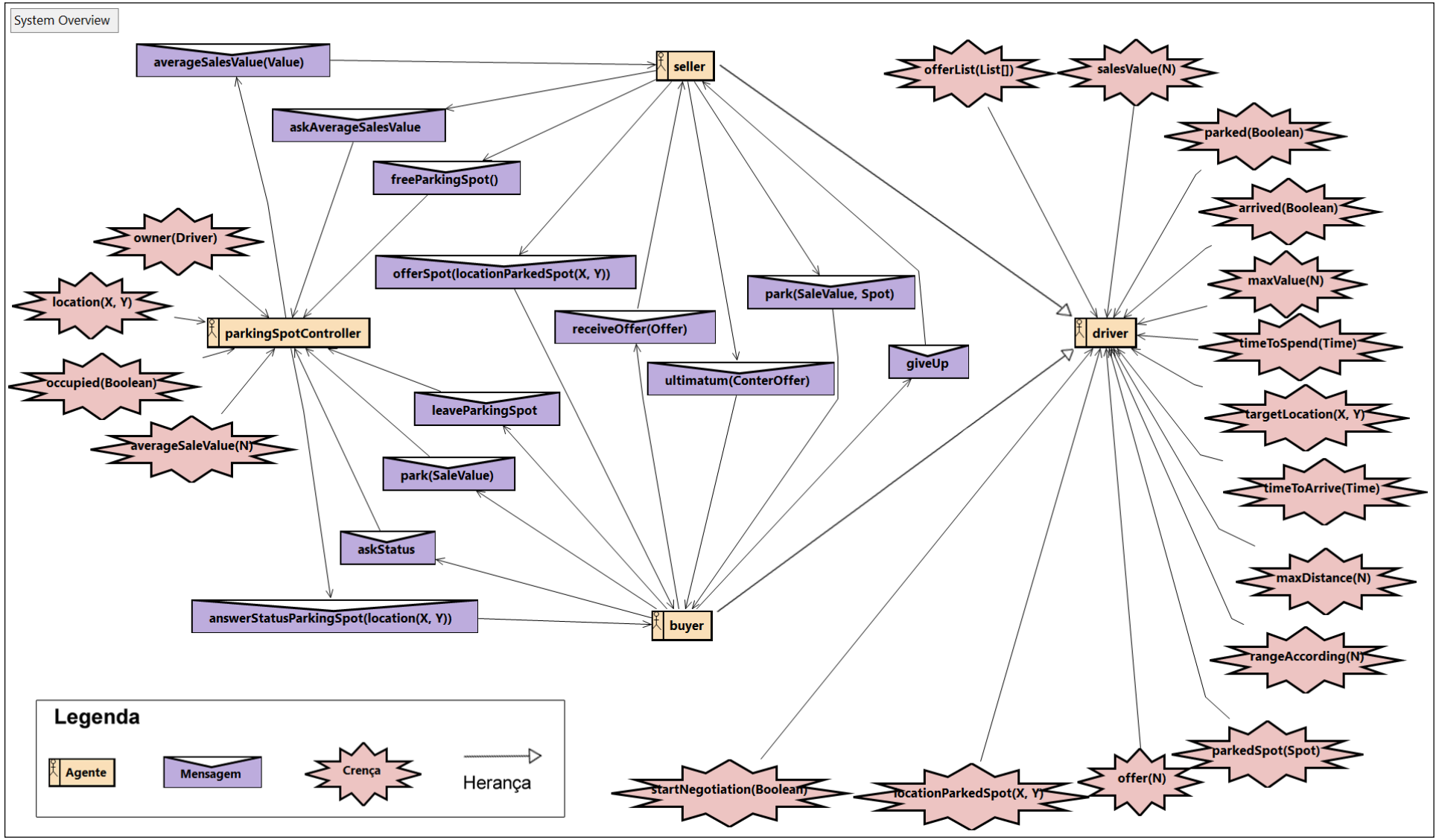
Na camada do Cartago, foram definidos dois artefatos: o artefato *ParkingSpot*, que é a vaga a qual o agente *parkingSpotController* controla e o artefato *ProposalGenerator*, que é um artefato gerador de proposta, o qual os agentes drivers utilizam para calcular as propostas da negociação. Na camada do *Moise*, foi feita a especificação organizacional do sistema, onde foram definidos os dois papeis que o agente *driver* pode assumir, *seller* e *buyer.*

Na Figura 16 é apresentado um diagrama da visão geral do Sistema Multiagente utilizando a metodologia *Prometheus*, vista anteriormente na Seção 3.5.1, para ilustrar a interação entre os agentes bem como suas ações e crenças. No diagrama é possível verificar os dois agentes desenvolvidos em *Jason* e a relação de herança entre o agente *driver* e os dois papeis distintos que ele pode assumir dentro do sistema: i) *buyer*: quando está procurando uma vaga para estacionar; e ii) *seller*: quando já possui uma vaga e desejam vender a mesma para um outro agente estacionar.

Quando um agente chega no estacionamento, a primeira ação tomada é verificar se existe alguma vaga livre e que está dentro da distância máxima aceitável, para estacionar. A Figura 17 apresenta a parte do diagrama de visão geral do MAPS-OPEN, que especifica a troca de mensagens que ocorre quando um *buyer* chega no estacionamento, onde inicialmente o *buyer* envia a mensagem *askStatus* para todos os agentes *parkingSpotController*, perguntando se a vaga está vazia.

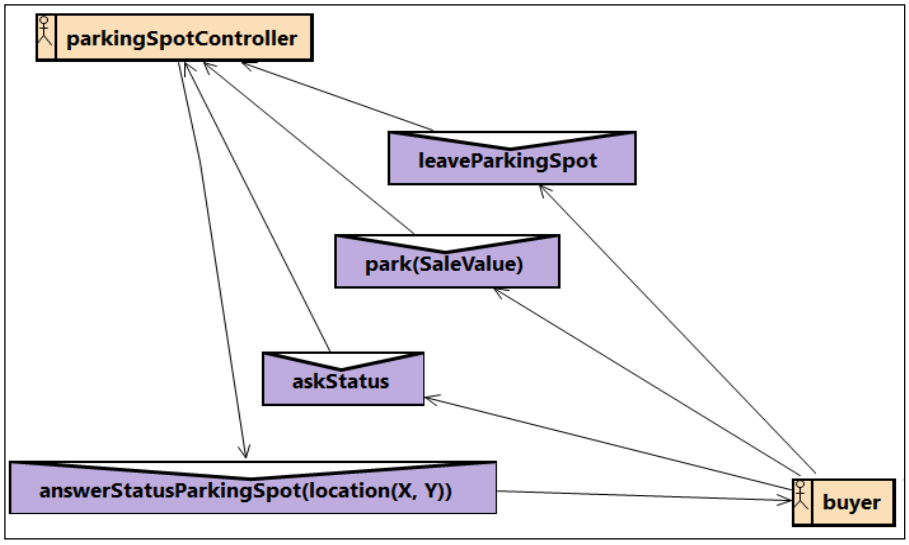
O *parkingSpotController* responde esta mensagem se a vaga efetivamente estiver vazia, com a mensagem *answerStatusParkingSpot.* Caso o *buyer* receber esta resposta, ele analisa e se a distância da vaga em questão está dentro da distância máxima aceitável estaciona na vaga, enviando a mensagem *park* para o agente *parkingSpotController*. Neste caso, a vaga foi ocupada pela primeira vez por um *driver* e ele se torna dono temporariamente deste recurso, até que venda para outro *driver*.

Figura 16 – Diagrama Visão Geral do SMA



Fonte: Autoria Própria

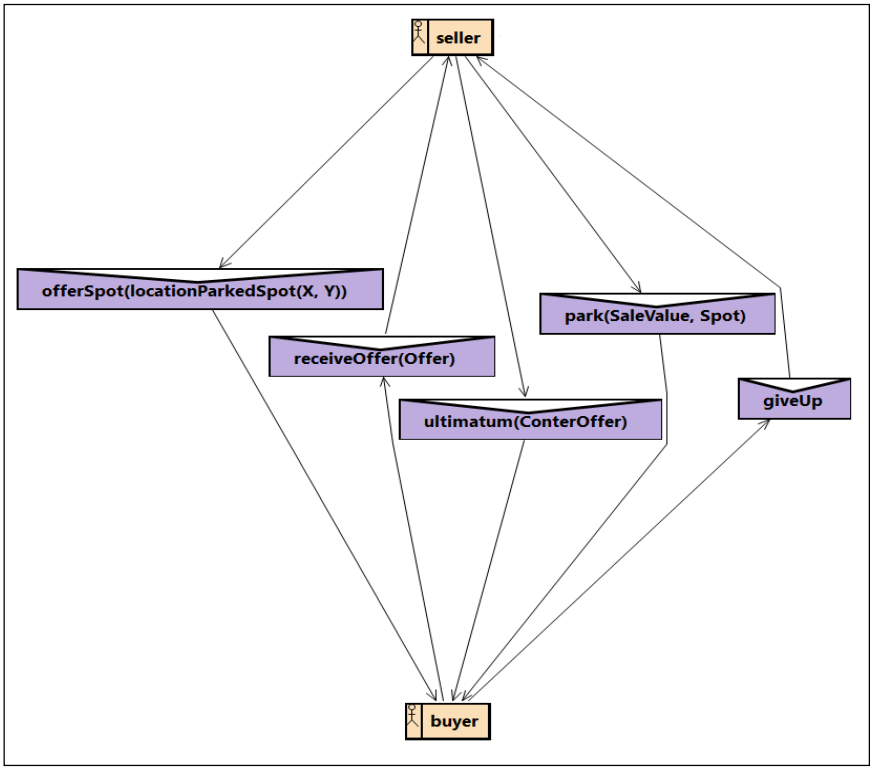
Figura 17 – Negociação entre *Buyer* e *ParkingSpotController*



Fonte: Autoria Própria

Quando o *buyer* estaciona na vaga, ele permanece o tempo previamente estipulado estacionado. Depois deste tempo ele muda de papel, passando a executar o papel de *seller*, onde tenta vender a vaga de estacionamento para outro *buyer.* A Figura 18 apresenta a parte do diagrama de visão geral do SMA que especifica a troca de mensagens que ocorre entre um *seller* e um *buyer* na negociação.

Figura 18 – Negociação entre *Seller* e *Buyer*



Fonte: Autoria Própria.

Inicialmente, o *seller* envia a mensagem *OfferSpot* para todos os *drivers* do sistema, informando que tem uma vaga disponível para venda. Os *drivers* que recebem esta mensagem de oferta e estão procurando por uma vaga de estacionamento, respondem com a mensagem *receiveOffer(Offer)*, enviando uma proposta de compra para o *seller.*

O *seller* analisa esta proposta, podendo tomar uma das seguintes decisões: i) aceitar a proposta, enviando a mensagem *park* para o *buyer*; ii) rejeitar proposta, onde a negociação entre os dois agentes acaba; ou iii) enviar contraproposta, por meio da mensagem *Ultimatum*. Caso o *buyer* receba esta contraposta ele a analisa, podendo tomar duas ações: i) aceitar a contraproposta, estacionando na vaga em questão; ou ii) rejeitar contraproposta, enviando a mensagem *GiveUp* para o *seller*.

Conforme citado, existem dois papeis no sistema, os quais foram implementados utilizado a camada organizacional do JaCaMo, o *Moise*. Nesta camada é possível realizar uma especificação social do SMA, definindo papeis e tarefas para cada agente dentro do sistema (o Apêndice A apresenta a especificação organizacional do sistema integralmente, utilizando o *Moise*).

O Código 1 apresenta especificação estrutural do sistema utilizando o *Moise*, onde são definidos os papeis, as relações entre os papeis e os grupos do SMA. Nas linhas 16 à 24 do código é possível verificar a definição dos papeis do sistema, onde observarmos a especificação dos papeis: *parkingSpotController*, *seller* e *buyer*, estes dois últimos têm uma relação de herança com o papel *driver*, isto é, eles herdam as características do papel.

Código 1 – Especificação Estrutural no *Moise*

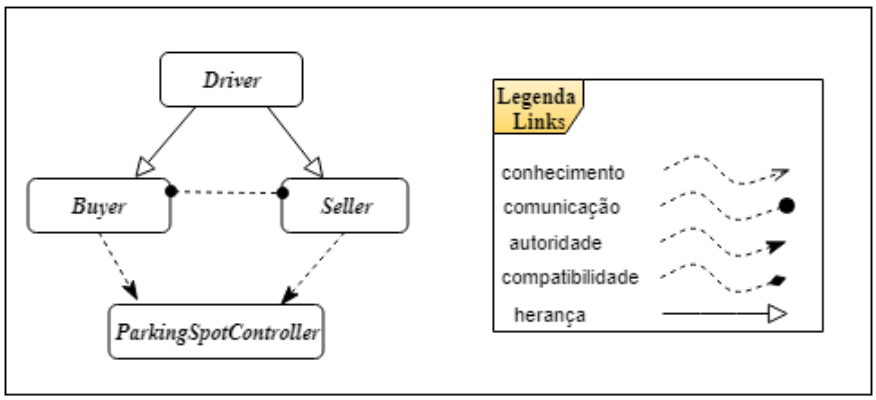
```
14 <structural-specification>
15
16 <role-definitions>
17     <role id="driver" />
18     <role id="buyer">
19     <extends role="driver"/>
20     </role>
21     <role id="seller">
22     <extends role="driver"/>
23     </role>
```

```
24  </role-definitions>
25
26  <group-specification id="parkingLot">
27
28      <roles>
29          <role id="buyer"/>
30          <role id="seller"/>
31          <role id="parkingSpotController"/>
32      </roles>
33
34      <links>
          <link   from="buyer"   to="seller"   type="communication"
35  scope="intra-group" bi-dir="true"/>
          <link   from="seller"   to="buyer"   type="communication"
36  scope="intra-group" bi-dir="true"/>
          <link       from="buyer"       to="parkingSpotController"
37  type="authority"  scope="intra-group" bi-dir="false"/>
          <link       from="seller"       to="parkingSpotController"
38  type="authority"  scope="intra-group" bi-dir="false"/>
39      </links>
```

Fonte: Autoria Própria.

A Figura 19 apresenta um diagrama dos relacionamentos dos papeis definidos no *Moise.* Onde é possível verificar que o papel *driver* é especializado nos papeis *buyer* e *seller*, estes dois possuem uma relação de comunicação entre si e uma relação de autoridade perante o papel *ParkingSpotController.*

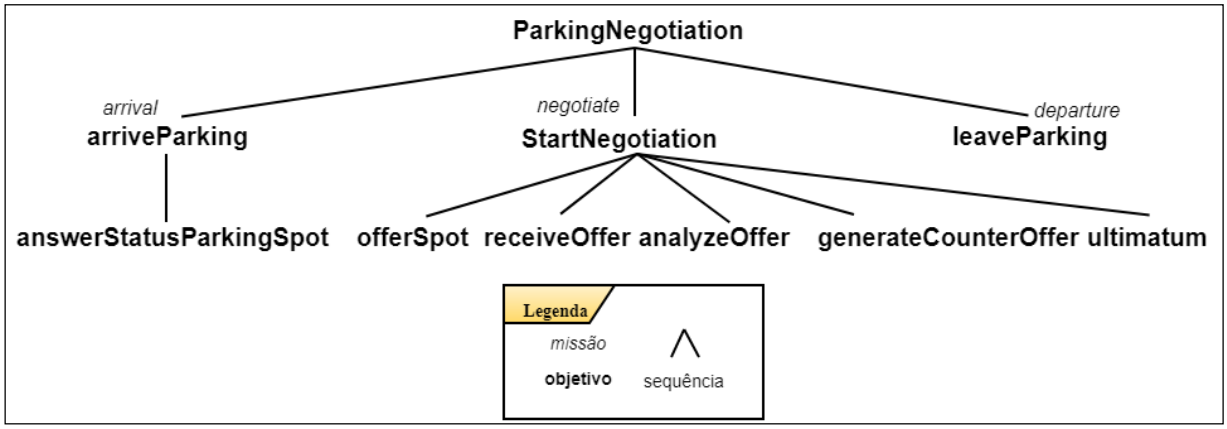
Figura 19 – Relacionamentos dos Papeis no *Moise*



Fonte: Autoria Própria.

No Código 1, Nas linhas 34 à 39, está a especificação das relações entre os papeis, onde existe uma relação de comunicação entre os papeis *buyer* e *seller* e uma relação de autoridade dos papeis *buyer* e *seller* sobre o *parkingSpotController*.

Figura 20 – Esquema Especificação Moise



Fonte: Autoria Própria.

O Código 2 apresenta a especificação funcional do sistema, onde são definidos o esquema e as missões. A Figura 20 apresenta a árvore de decomposição do esquema do SMA, definido nas linhas 52 à 92, onde é possível verificar que, os objetivos e missões são executados de maneira sequencial da esquerda para direita.

Código 2 – Especificação Funcional no *Moise*

```
51 <functional-specification>
52     <scheme id="parkingLotScheme">
53
54         <goal id="ParkingNegotiation">
55
56             <plan operator="sequence">
57
58                 <goal id="arriveParking">
59                     <plan operator="sequence">
60                         <goal id="answerStatusParkingSpot"></goal>
61                     </plan>
62                 </goal>
63
64                 <goal id="startNegotiation">
```

```
65                      <plan operator="sequence">
66                          <goal id="offerSpot"></goal>
67                          <goal id="receiveOffer"></goal>
68                          <goal id="analyzeOffer"></goal>
69                          <goal id="generateCounterOffer"></goal>
70                          <goal id="ultimatum"></goal>
71                      </plan>
72                  </goal>
73
74                  <goal id="leaveParking"/>
75
76              </plan>
77
78          </goal>
79
80          <mission id="arrival">
81              <goal id="arriveParking"/>
82          </mission>
83
84          <mission id="negotiate">
85              <goal id="startNegotiation"/>
86          </mission>
87
88          <mission id="departure">
89              <goal id="leaveParking"/>
90          </mission>
91
92      </scheme>
93  </functional-specification>
```

Fonte: Autoria Própria.

O objetivo *ParkingNegotiation* é decomposto em três missões: i) *arrival*, que é decomposta em *arriveParking*, que por sua vez é decomposta em *answerStatusParkingSpot*; ii) *negotiate*, que é decomposta em *StartNegotiation*, que por sua vez é decomposta em *generateOffer*, *analyzeOffer*, *generateCounterOffer* e *ultimatum*; e iii) *departure,* que é decomposta em *leaveParking.* É importante ressaltar

que, para cada objetivo definido no *Moise* existe um plano de execução correspondente no *Jason.*

No Código 2, nas linhas 80 à 90, são definidas as missões do sistema. Inicialmente missão *arrival* é executada pelo *buyer*, iniciando pelo *arriveParking*, o qual é decomposto no objetivo *answerStatusParkingSpot*, onde o agente *buyer* envia uma mensagem para todos os agentes *parkingSpotController* do sistema perguntando se a vaga pela qual ele é responsável está livre. Na primeira resposta assertiva, o agente *buyer* estaciona na vaga livre e o agente *parkingSpotController* muda o status da vaga para ocupada, caso não haja vagas livres a segunda missão é executada.

A implementação do plano *arriveParking* pode ser visualizado no Código 3, onde inicialmente o *driver* adota o papel de *buyer* na linha 3, depois aguarda o tempo definido previamente para entrar no estacionamento e por final envia uma mensagem *broadcast*, na linha 12, para todos os agentes *parkingSpotController,* perguntando se a vaga que o agente controla está vazia.

Código 3 – Implementação plano *arriveParking* em *Jason*

```
1  +!arriveParking: timeToArrive(TimeToArrive) &
2  broadcastSentNumber(N) & targetLocation(X,Y) <-
3     adoptRole(buyer)[artifact_id(parkingLot_org)];
4     .my_name(Me);
5     startDriver(Me); //utilizado na interface gráfica
6     .print("Aguardando para entrar - ", TimeToArrive);
7     .wait(TimeToArrive);
8     -+arrived(true);
9     arrivalTime(Me, X, Y, MaxValue); //utilizado para gerar
   relatório
10    .print("Entrando");
11    -+broadcastSentNumber(N+1); //utilizado para gerar relatório
12    .broadcast(achieve, askStatus).
```

Fonte: Autoria Própria.

Quando esta mensagem *broadcast* enviada pelo agente *driver buyer* chega para o agente *parkingSpotController*, ele executa o plano *askStatus*, este plano envia uma mensagem de resposta para o agente *buyer* caso a vaga esteja vazia. Caso o

agente receba a mensagem que a vaga está vazia, ele confere se a vaga em questão está dentro do limite de distância máxima definido. Em caso assertivo estaciona nesta vaga. Se o *buyer* não receber nenhuma resposta de um agente *parkingSpotController* informando que a vaga está vazia, ele aguarda até que um *seller* inicie uma negociação.

A segunda missão executada pelo esquema definido no *Moise* é a *negotiate*, que inicia com o objetivo *StartNegotiation*, executando o plano *StartNegotiation*, o qual pode ser visualizado no Código 4. Este plano é executado pelos agentes *seller* na tentativa de vender a vaga.

O plano *StartNegotiation* verifica se: i) está no momento de enviar um novo *broadcast* ofertando a vaga (Código 4, linha 6), quando a lista de ofertas recebidas (*OfferList)* está vazia; ii) o *seller* já tentou vender a vaga o número máximo de vezes definido (Código 4, linha 8) e deve desistir de vender a vaga; e iii) está no momento de iniciar uma nova negociação com um *buyer* (Código 4, linha 26), isso acontece quando a lista de ofertas não está vazia e a negociação com o último *driver buyer* já encerrou.

Código 4 – Implementação plano StartNegotiation em *Jason*

```
1  +!startNegotiation: offerList(OfferList) &
2  locationParkedSpot(LocationParkedSpotX, LocationParkedSpotY)&
3  parkedSpot(ParkedSpot) & messageSentNumber(MessageSentNumber)
4  & broadcastSentNumber(BroadcastSentNumber) &
5  startNegotiation(StartNegotiationBoolean) & broadcastToSell(B)<-
6        if(.empty(OfferList)){
7
8              if(BroadcastSentNumber > B){
9                    .send(ParkedSpot, achieve, freeParkingSpot);
10                   .my_name(Me);
11                   freePark(Me, ParkedSpot); //interface
12                   sell(Me, MessageSentNumber,
13                   BroadcastSentNumber,
14                   "Gave up");
15                   .print("Desistiu de vender a vaga");
16                   .fail_goal(startNegotiation);
17             }
```

```
18
19              -+broadcastSentNumber(BroadcastSentNumber+1);
20              -+startNegotiation(true);
21              .broadcast(achieve,
22           offerSpot(locationParkedSpot(LocationParkedSpotX,
23               LocationParkedSpotY)));
24              .print("Enviando broadcast");
25        } else{
26              if(StartNegotiationBoolean = true){
27                    -+startNegotiation(false);
28                    .nth(0, OfferList, Driver);
29                    .nth(1, OfferList, Offer);
30                    !analyzeOffer(driver(Driver), offer(Offer));
31              }
32        }
33        .wait(1000)
34        !startNegotiation;.
```

Fonte: Autoria Própria.

Quando os agentes recebem a mensagem o*fferSpot*, enviada por um *seller* (Código 4, linha 22) o plano o*fferSpot* é executado, que pode ser visualizado no Código 5. Segundo a definição do *Moise* (Figura 20), o plano o*fferSpot* é o primeiro sub plano a ser executado do plano *StartNegotiation*. Utilizando plano o*fferSpot,* os *buyers* manifestam seu interesse pela oferta de vaga recebida, enviando como resposta uma oferta de compra para o *seller.* A oferta é gerada utilizando o artefato *ProposalGenerator,* que encapsula a Função 1 do Modelo de Raciocínio na operação *generateOffer*.

Código 5 – Implementação plano OfferSpot em *Jason*

```
1  +!OfferSpot(locationParkedSpot(LocationParkedSpotX,
2  LocationParkedSpotY))[source(AG)]:
3  messageSentNumber(N) &
4  maxValue(MaxValue) &
5  maxDistance(MaxDistance) &
6  targetLocation(TargetLocationX,
7  TargetLocationY) &
```

```
8  parked(Parked) &
9  arrived(Arrived) <-
10       if (Arrived = true){
11            if (Parked = false){
12                 .print("Oferta de vaga recebida de ",  AG, ",
13                 iniciando negociação");
14                 generateOffer(MaxValue,
15                 MaxDistance,
16                 LocationParkedSpotX,
17                 LocationParkedSpotY,
18                 TargetLocationX,
19                 TargetLocationY,
20                 Offer);
21                 .send(AG, achieve,
22                 receiveOffer(offer(Offer)));
23                 -+messageSentNumber(N+1);
24                 -+offer(Offer);
25                 .print("Proposta ", Offer,
26                 " enviada para o agente ", AG)}
27            else{
28                 .print("Oferta de vaga recebida de ",
29                 AG, " ignorada, já tenho vaga.")
30            }
31       }.
```

Fonte: Autoria Própria.

Na linha 14 do Código 5, a operação *generateOffer* é invocada e a oferta é gerada, então esta oferta é enviada como resposta para o agente *seller* na linha 21. Caso o agente não tenha interesse em comprar uma vaga, isso significa que ele não está executando o papel de *buyer,* sendo assim ele não responde a oferta de vaga (Código 5, linhas 27 à 30).

Quando o agente no papel de *seller recebe* esta resposta ele processa a oferta usando o plano *AnalyzeOffer*, o qual segundo a definição do Moise (Figura 20) é o segundo sub plano a ser executado do plano StartNegotiation. O plano *AnalyzeOffer* pode ser verificado no Código 6, este plano encapsula a Função 2 e pode ocasionar

três estados possíveis para o agente *driver seller*: aceitar a proposta recebida do *buyer* (Código 6, linhas 13 à 26), rejeitar a proposta (Código 6, linhas 27 à 32) ou enviar uma contra oferta para o *buyer* (Código 6, linhas 33 à 37). Nas linhas 8 e 9 do Código 6, é possível verificar o agente perguntando o valor médio de venda da vaga em questão para o agente *parkingSpotController (*agente responsável pela vaga), sendo este valor utilizado pela Função 2.

Código 6 – Implementação plano analyzeOffer em *Jason*

```
1  +!analyzeOffer(driver(Driver),
2  offer(Offer)):
3  parkedSpot(ParkedSpot) &
4  locationParkedSpot(X, Y) &
5  messageSentNumber(MessageSentNumber) &
6  broadcastSentNumber(BroadcastSentNumber) <-
7
8        ?averageSaleValue(M);
9        -+averageSaleValue(M);
10
11       .print("Negociando com: ", Driver);
12
13       if (Offer >= M){
14             .print("Negociação encerrada,
15             vaga vendida para o agente ",
16             Driver);
17             .my_name(Me);
18             sell(Me, MessageSentNumber+1,
19             BroadcastSentNumber, "Sell");
20             .send(Driver, achieve,
21             park(parkedSpot(ParkedSpot),
22             locationParkedSpot(X, Y),
23             salesValue(Offer)));
24             -+messageSentNumber(
25             MessageSentNumber+1);
26       }
27       if (Offer < M/2){
```

```
28              .print("Negociação encerrada
29              com o agente ", Driver, ",
30              sem trato");
31              !giveUp;
32        }
33        if ((Offer < M) & (Offer >= M/2)){
34              .print("Gerar contra oferta!!!!");
35              !generateCounterOffer(
36              source(Driver));
37        }.
```

Fonte: Autoria Própria.

Se o modelo de raciocínio causar que o agente entre no último caso (Código 6, linhas 13 à 37), isto é, enviar uma contra oferta, o plano *generateCounterOffer* é executado, o qual segundo a definição do Moise (Figura 20) é o terceiro sub plano a ser executado do plano *StartNegotiation*. Se o *seller* executar o plano *GenerateCounterOffer* uma contra oferta é enviada para o *buyer*, esta contra oferta é o valor médio de venda da vaga em questão.

Caso o agente *buyer* receba a contra oferta, o plano *ultimatum* é executado, o qual segundo a definição do Moise (Figura 20), é o último sub plano a ser executado do plano *StartNegotiation.* O plano *ultimatum* pode ser verificado no Código 7, este plano garante um fim a negociação, encapsulando a Função 3 e podendo gerar dois estados possíveis: o *buyer* aceita a contra oferta e estaciona na vaga (Código 7, linhas 12 à 19) ou recusa e informa o *driver seller* que desistiu da negociação (Código 7, linhas 20 à 25).

Código 7 – Implementação plano ultimatum em *Jason*

```
1  +!ultimatum(counterOffer(CounterOffer),
2  parkedSpot(ParkedSpot),
3  locationParkedSpot(X, Y) )[source(AG)]:
4  offerList(OfferList) &
5  rangeAccording(RangeAccording) &
6  offer(Offer) &
7  messageSentNumber(N) <-
8        .print("Contra oferta ",
```

```
9         CounterOffer, " recebida do agente ",
10        AG);
11
12         if ((CounterOffer - Offer) <=
13         (Offer*RangeAccording)){
14              .print("Contra proposta aceita
15              do agente ", AG);
16              .send(AG, achieve, counterOfferAccepted);
17              !park(parkedSpot(ParkedSpot),
18              locationParkedSpot(X, Y),
19              salesValue(CounterOffer));
20        } else {
21              .print("Contra proposta rejeitada, sem trato.
22              Encerrada negociação com o agente ", AG);
23              .send(AG, achieve, giveUp);

24              -+messageSentNumber(N+1);
25        }.
```

Fonte: Autoria Própria.

Quando um agente fecha uma negociação e consegue uma vaga para estacionar, ele executa o plano *park,* onde muda o papel de *buyer* para *seller,* aguardando o tempo definido que deve permanecer na vaga. Depois de aguardar este tempo executa a missão *departure*, que segundo a definição do Moise (Figura 20) realiza o plano *leaveParking,* o qual pode ser verificado no Código 8. Na linha 6, envia uma mensagem para o *parkingSpotController*, informando que está deixando a vaga e por último inicia o plano startNegotiation, iniciando as negociações para venda da vaga (linha 10).

Código 8 – Implementação plano leaveParking em *Jason*

```
1  +!leaveParking: parkedSpot(ParkedSpot)
2  & messageSentNumber(MessageSentNumber) <-
3        .my_name(Me);
4        departureTime(Me);
5        .print("Saindo da vaga ", ParkedSpot);
```

```
6       .send(ParkedSpot, achieve, leaveParkingSpot);
7       -+messageSentNumber(MessageSentNumber+1);
8       leaveParkedSpot(Me);
9       leaveSpotSumo(Me, ParkedSpot);
10      !startNegotiation.
```

Fonte: Autoria Própria.

Na parte final da especificação do *Moise* disponível no Código 9, nas linhas 96 à 102, está a especificação normativa do sistema, onde são definidas por meio de normas as missões que cada papel deve executar.

Código 9 – Especificação Normativa no *Moise*

```
96  <normative-specification>
97      <norm      id="norm1"      type="obligation"      role="buyer"
    mission="arrival"/>
98
99      <norm      id="norm2"      type="permission"      role="driver"
    mission="negotiate"/>
100
101     <norm      id="norm3"      type="permission"      role="seller"
    mission="departure"/>
102 </normative-specification>
```

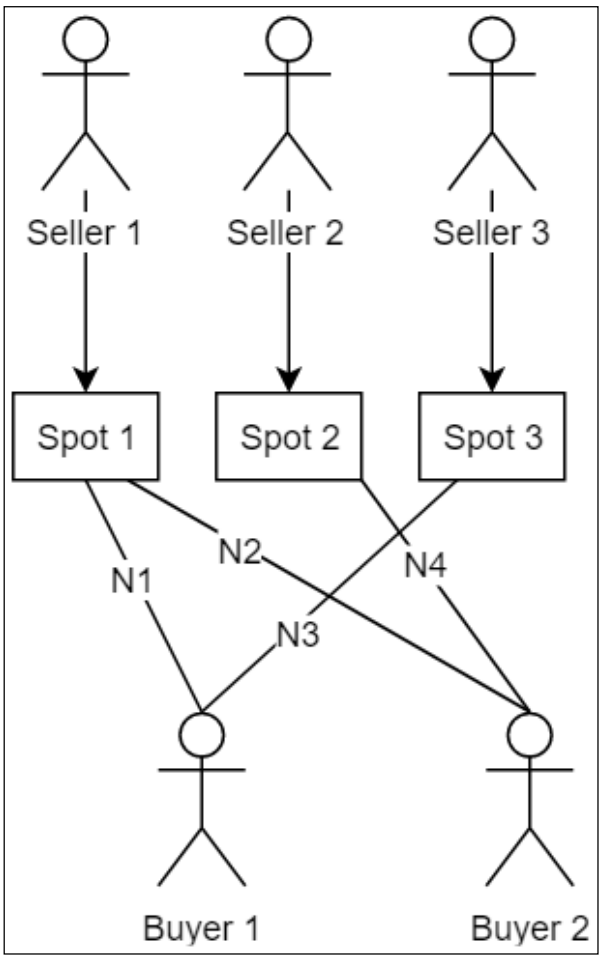
Fonte: Autoria Própria.

No Código 9 existem três normas definidas no sistema (linhas 97 à 101): i) *norm1,* que define que o *driver buyer* tem obrigação de executar a missão *arrival*; ii) *norm2*, que define que os agentes *driver* podem executar a missão *negotiate*, como os papeis *buyer* e *seller* herdam as características do *driver* ambos os papeis podem executar esta missão; e iii) *norm3*, onde define que o agente no papel *seller* tem a permissão de executar a missão *departure*.

A Figura 21 exemplifica como acontece as negociação entre os agentes no SMA apresentado neste trabalho. Onde é possível visualizar três agentes *sellers* (*Seller* 1, *Seller 2 e Seller* 3), juntamente com suas respectivas vagas (*Spot* 1, *Spot* 2 e *Spot* 3) que desejam vender e os agentes *buyes* (*Buyer* 1 e *Buyer* 2), que estão

interessados em adquirir uma vaga. Também é possível verificar as *threads* de negociação (N1, N2, N3 e N4) que acontecem entre os agentes *Seller* e *Buyer.*

Figura 21 – Esquema de Negociação do SMA



Fonte: Autoria Própria

Na Figura 21 é possível visualizar que o agente *Seller* 1 está negociando a vaga *Spot* 1 com dois *Buyers*, o *Buyer* 1 e o *Buyer* 2. Estas *threads* de negociação não acontecem de modo paralelo, o sistema foi implementado desta maneira para evitar inconsistências como, por exemplo, a possibilidade de que se as *threads* forem paralelas os dois agentes fecharem o acordo no mesmo instante e ambos conseguirem a mesma vaga, no caso do exemplo o *Spot* 1 seria alocado tanto para o *Buyer 1* quanto para o *Buyer* 2.

Por esta razão, localmente as negociações entre *sellers* e *buyer* são centralizadas, pois a negociação de uma vaga pode se dar entre um s*eller* e vários *buyers.* Porém, um buyer consegue negociar paralelamente com vários *sellers*, como é possível visualizar na Figura 21, onde o agente *Buyer* 1 negocia com os agentes

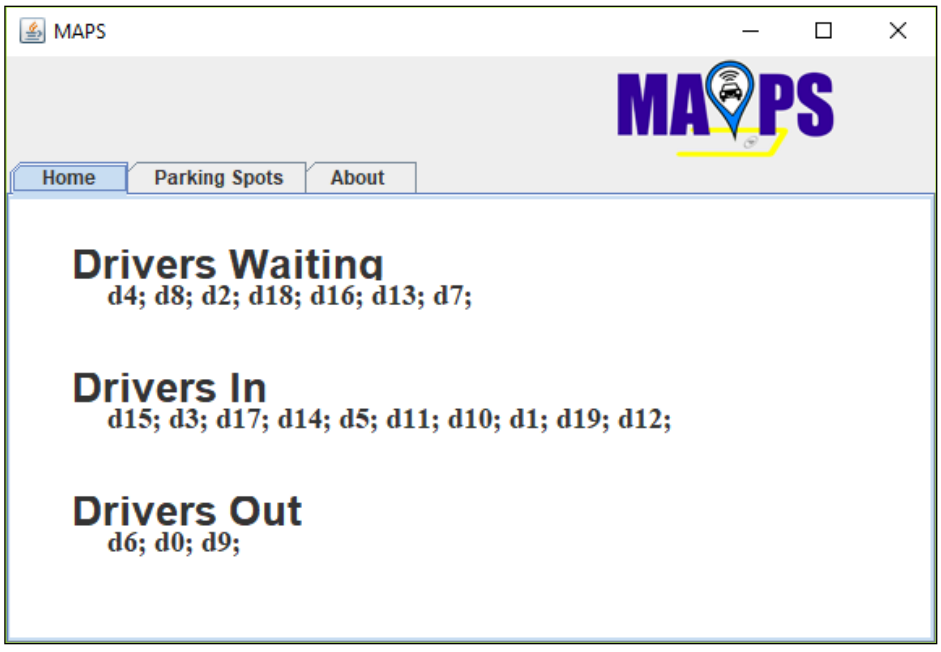
Seller 1 e *Seller* 3 simultaneamente. Portanto de modo global os sistema é descentralizado, pois é composto de várias *threads* de negociações centralizadas que podem ser paralelas ou não.

4.4 INTERFACE GRÁFICA

Para auxiliar na visualização das ações dos agentes no SMA, foi implementada uma interface gráfica, por meio de um artefato no *Cartago* chamado *GraphicalInterface.* A interface foi implementada com base nos resultados apresentados em Ducheiko e Alves (2016), onde uma interface gráfica foi desenvolvida para a versão inicial do Projeto MAPS, que utilizava um mecanismo de negociação centralizado.

Na tela inicial da interface gráfica, que pode ser visualizada na Figura 22, o usuário possui a relação dos *drivers* que estão aguardando uma vaga de estacionamento (*Drivers Waiting)*, dos *drivers* que estão dentro do estacionamento utilizando uma vaga (*Drivers In*) e dos *drivers* que já deixaram o estacionamento (*Drivers Out).*

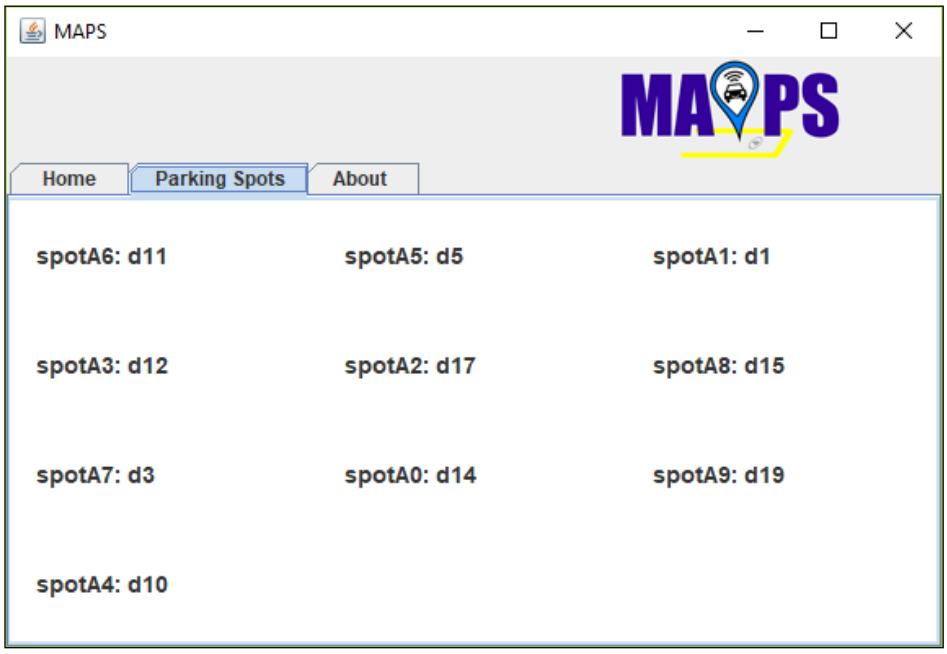
Figura 22 – Aba *Home* da Interface Gráfica



Fonte: Autoria Própria

Na aba *Parking Spots*, que pode ser visualizada na Figura 23, o usuário possui informações sobre as vagas de estacionamentos do sistema. Por meio desta aba, é possível verificar qual o agente *driver* que está estacionado em uma determinada vaga ou se a vaga está vazia.

Figura 23 – Aba *Parking Spots* da Interface Gráfica



Fonte: Autoria Própria.

O desenvolvimento desta interface gráfica proporciona a visualização rápida de informações da execução do Sistema Multiagente. Deste modo, ajudando a identificar erros, ajudando a realizar aperfeiçoamentos no SMA e facilitando a geração de cenários de testes, os quais serão abordados no Capítulo 7.

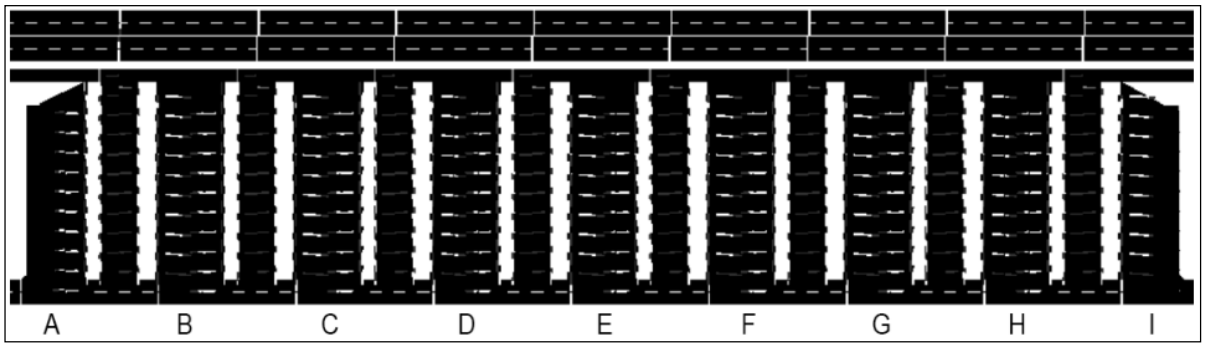
4.5 CONEXÃO COM O SUMO

A conexão do SUMO com o Sistema Multiagente desenvolvido se deu com base em um *middleware*, desenvolvido no trabalho de Heijmeijer e Alves (2017), onde é apresentado a interligação do SMA inicial do Projeto MAPS com a ferramenta SUMO. A implementação utiliza a biblioteca *Traci4J*, desenvolvida na linguagem Java, que cria uma interface de comunicação com o SUMO, permitindo controlar e obter informações de uma simulação. A conexão é feita por um artefato no *Cartago,*

chamado *Simulation* e utiliza uma arquitetura cliente-servidor, onde o SMA trabalha como cliente e o *middleware* como servidor

O modelo de estacionamento utilizado nas simulações é o mesmo proposto por Heijmeijer e Alves (2017). A especificação do modelo pode ser visualizada na Figura 24 e possui 160 vagas divididas em 9 setores: A, B, C, D, E, F, G, H e I. Este modelo simula um ambiente semelhante a um estacionamento de um aeroporto, com algumas adaptações.

Figura 24 – Modelo de Estacionamento Utilizado



Fonte: Adaptado de Heijmeijer e Alves (2017).

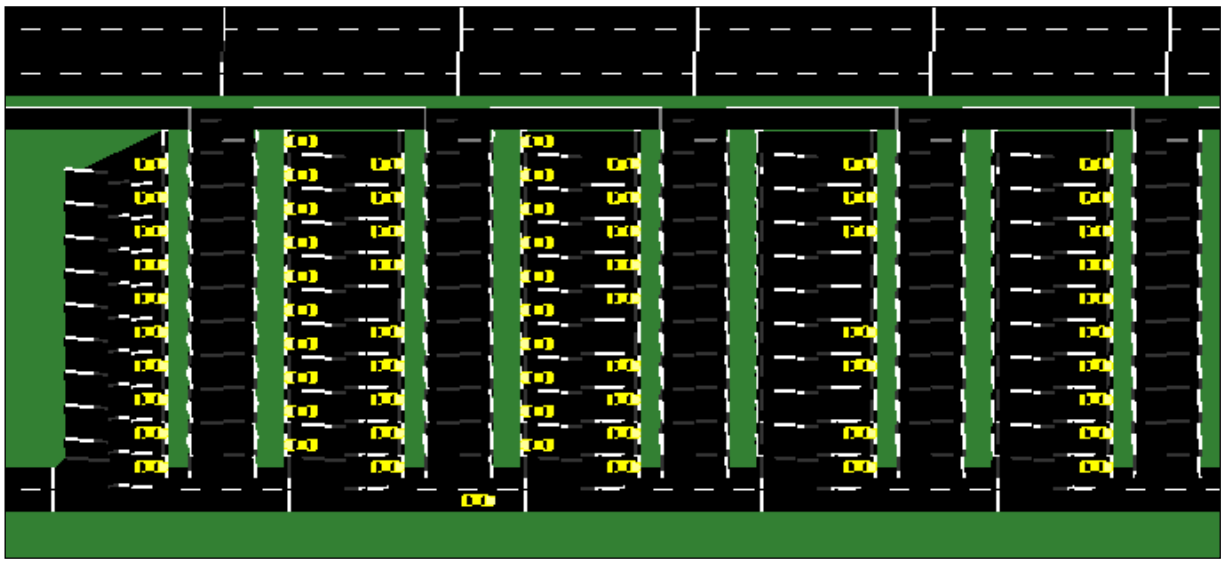
Quando o SMA deseja efetuar alguma alteração no SUMO, o artefato *Simulation* abre uma comunicação com o *middleware* e encaminha um protocolo, que define a ação que deve ser tomada pelo simulador SUMO. Abaixo é possível visualizar o protocolo definido para implementar a conexão no presente trabalho (adaptado de Heijmeijer e Alves (2017)):

Protocolo = <idDriver, tipoProtocolo, idSpot>

onde:

- *idDriver*: é um campo identificador do agente driver, sempre preenchido com a letra D;
- *tipoProtocolo*: tipo do protocolo, que indica a ação que deve ser tomada pelo SUMO: PS (*Parking Spot*) para informar que um agente deve estacionar em uma vaga, LS (*Leaving Spot*) para informar que um agente deve deixar o estacionamento.
- *idSpot*: identificação da vaga a ser ocupada ou desocupada.

Figura 25 – Simulação do SMA com o SUMO



Fonte: Autoria Própria.

A Figura 25 apresenta a execução do SMA utilizando o Simulador SUMO, onde foram utilizados os setores A, B, C, D e E, totalizando 70 vagas de estacionamento. Na simulação foram utilizados 150 *drivers*, com distância máxima para estacionar igual à 200 unidades de medida, valor máximo a pagar igual à 10 unidades de venda. É importante ressaltar que os agentes *drivers* entram e saem do sistema, por esta razão não é possível visualizar os 150 *drivers* na Figura 25, somente os *drivers* que estão efetivamente dentro do estacionamento aparecem na simulação gráfica.

A implantação desta conexão com o simular SUMO complementa a interface gráfica apresentada na seção anterior, pois possibilita observar o sistema de modo mais efetivo, proporcionando uma visualização em tempo de execução das vagas e dos *drivers* no sistema, auxiliando no entendimento dos fenômenos que acontece na execução do SMA. Deste modo, ajudando a identificar erros, oferecendo uma visualização mais clara de como os agentes ocupam as vagas no decorrer das negociações e ajudando a aperfeiçoar o Sistema Multiagente.

5 RESULTADOS

O SMA desenvolvido no decorrer do presente trabalho foi implementado com o intuito de oferecer uma nova abordagem de alocação de vagas para o Projeto MAPS, utilizando um mecanismo de negociação descentralizado. Na sequência deste capítulo são apresentadas os resultados obtidos por meio da execução de cenários de testes, com o objetivo de validar o funcionamento do SMA e verificar como o mecanismo de negociação se comporta com alterações de parâmetros do modelo de raciocínio.

É importante ressaltar que, o objetivo deste capítulo é analisar somente o funcionamento do SMA desenvolvido. Portanto, não são abordadas características como abrir e fechar cancelas do estacionamento, a maneira como é verificado se um *driver* estacionou em uma vaga e assume-se que todos os *drivers* obedecem rigorosamente ao protocolo de negociação apresentado.

5.1 CENÁRIOS DE TESTES

Para implementar os cenários de testes é necessário configurar o ambiente de simulação e os agentes. A configuração envolve o *hardware* utilizado, as ferramentas de *software* e os arquivos de configuração do SMA. Todos os experimentos foram executados utilizando as seguintes configurações de *hardware*:

- CPU: Intel® Core™ i7-3230M @ 2.6GHz;
- Memória: 16 GB RAM;
- GPU: NVIDIA Geforce GTX 970.

As ferramentas de *software* utilizadas no desenvolvimento do dos cenários de testes são enunciadas a seguir:

- Linux Mint 64bits;
- SUMO 0.27.1-win64 (http://sumo.dlr.de/wiki/Installing);
- TraCI4J (https://github.com/egueli/TraCI4J);
- Eclipse Java Photon 64bits (https://eclipse.org/downloads/);
- Framework JaCaMo 0.7-SNAPSHOT (https://sourceforge.net/projects/jacamo/files).

As configurações do SMA variam conforme as análises implementadas. A seguir, são apresentados os experimentos realizados juntamente com o objeto analisado:

- **Experimento 1:** análise da variação da quantidade de *drivers* no sistema;
- **Experimento 2:** análise da variação da distância máxima aceitável por um *driver*, variável α da Função 1;
- **Experimento 3:** análise da variação do valor máximo que um *driver* está disposto a pagar por uma vaga, variável λ da Função 1;
- **Experimento 4:** análise da variação da faixa de fechamento de acordo, variável δ da Função 3.

5.2 EXPERIMENTO 1

Conforme mencionado, o objeto de estudo desta seção é a variação do número de *drivers* no sistema. Para tanto, foram executados 10 cenários de testes, todos com 160 vagas de estacionamento. As configurações das variáveis do modelo de raciocínio em todos os cenários de teste foram definidas sendo α = 25, λ = 10 e δ = 0,2. A configuração dos parâmetros de cada cenário de teste pode ser visualizada na Tabela 1.

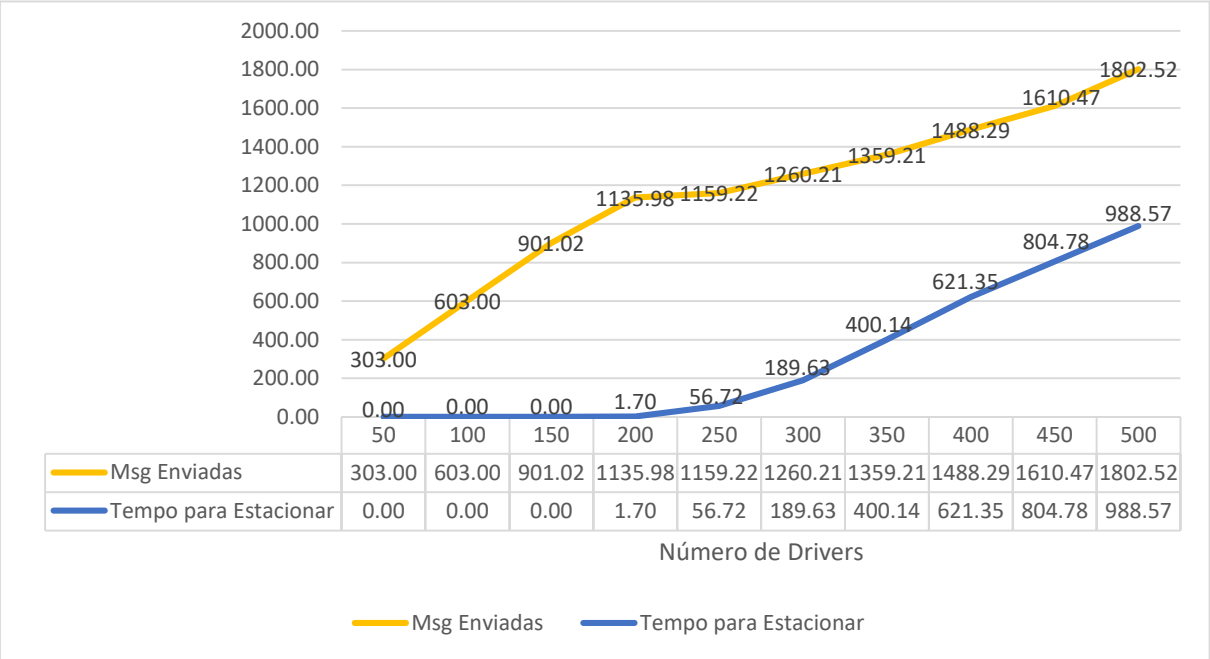
Tabela 1 – Configurações dos Cenários do Experimento 1

Cenário	Número de *Drivers*
Cenário 1	50
Cenário 2	100
Cenário 3	150
Cenário 4	200
Cenário 5	250
Cenário 6	300
Cenário 7	350
Cenário 8	400
Cenário 9	450
Cenário 10	500

Fonte: Autoria Própria.

O Gráfico 1 apresenta a variação do número médio de mensagens enviadas e o tempo médio para estacionar por *driver* em cada cenário de teste. No Gráfico 1 os valores médios do tempo para estacionar são multiplicados em 10 vezes, para que a escala fique mais próxima dos valores médios de mensagens enviadas e assim fazendo com que a análise fique mais clara.

Gráfico 1 – Número de Mensagens Médias Enviadas e Tempo Médio para Estacionar em Relação ao Número de *Drivers*



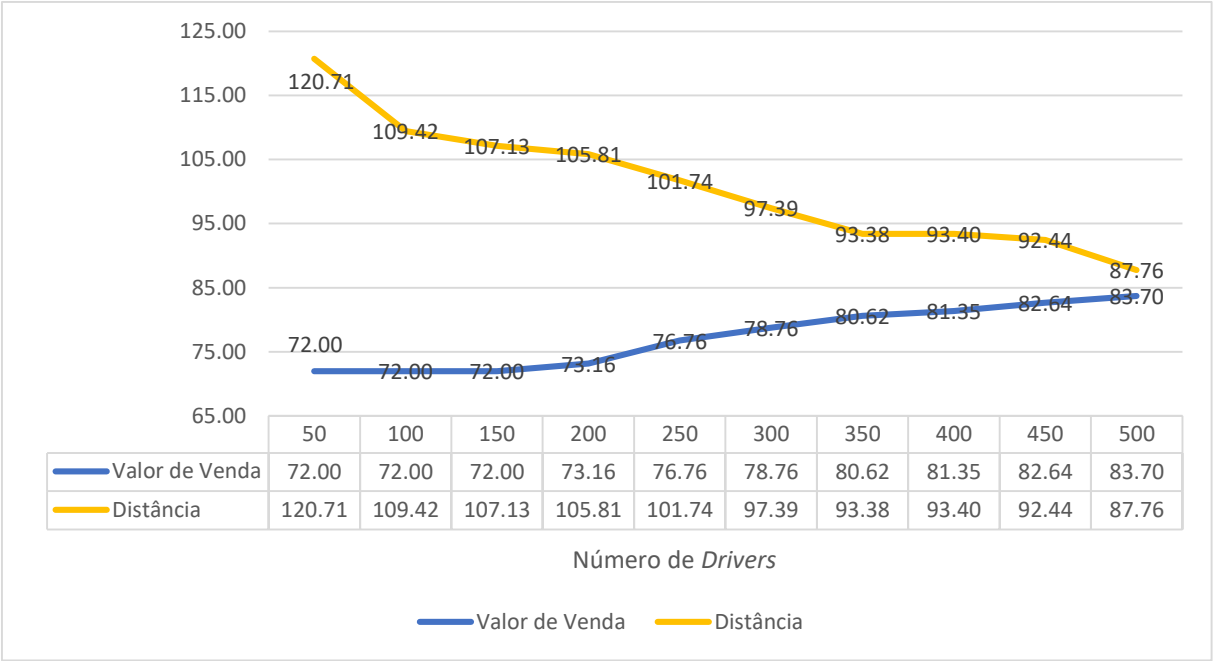
Fonte: Autoria Própria.

A partir da análise dos dados coletados da execução dos cenários de testes, foi possível constatar que o tempo para alocação da vaga de estacionamento tende a crescer conforme o número de *drivers* no sistema, como é possível verificar no Gráfico 1. Este fenômeno acontece devido ao fato de que, quanto maior o número de agentes no sistema mais mensagens serão enviadas para realizar a negociação das vagas. A troca de mensagens influi no tempo de alocação das vagas, pois quanto maior o número de mensagens trocadas para fechar uma negociação, mais demorada é esta negociação. Portanto, o tempo médio para estacionar também cresce conforme o número de *drivers* no sistema, como é possível visualizar no Gráfico 1.

O Gráfico 2 apresenta a variação da distância média entre o ponto onde o agente deseja a vaga e o ponto onde o agente efetivamente estacionou, juntamente com valor médio de venda das vagas em cada cenário. No Gráfico 2, os valores médios de venda das vagas são multiplicados em 12 vezes, para que a escala fique

mais próxima dos valores de distância média e assim fazendo com que a análise fique mais clara.

Gráfico 2 – Distância Média entre o Ponto de Desejo e o Ponto Estacionado e Valor Médio de Venda das Vagas em Relação ao Número de *Drivers*



Fonte: Autoria Própria.

Por meio da análise do Gráfico 2, foi possível constatar que, a distância média entre o ponto onde o agente deseja a vaga e o ponto onde o agente efetivamente estacionou diminui conforme o número de *drivers* no sistema aumenta. Consequentemente, o valor médio de venda das vagas tende a crescer conforme o número de *drivers* cresce, devido ao fato de que, quanto menor a distância entre estes pontos maior será o valor que um *driver* estará disposto a pagar por uma vaga. Este fenômeno acontece pois quanto maior o número de *drivers* no sistema, mais negociações irão acontecer. Portanto, os drivers *buyers* terão mais opções de vagas para negociar e como eles tendem a escolher vagas mais próximas ao ponto onde desejam estacionar, a distância média entre o ponto de desejo e o ponto onde estacionou diminui.

5.3 EXPERIMENTO 2

Segundo mencionado, o objeto de estudo desta seção é a análise da variação da distância máxima aceitável por um *driver*, variável α da Função 1. Para tanto, foram executados 10 cenários de testes, todos com 500 *drivers* e 160 vagas de estacionamento. As configurações das variáveis do modelo de raciocínio em todos os cenários de teste foram definidas sendo λ = 10 e δ = 0,2. A configuração dos parâmetros de cada cenário de teste pode ser visualizada na Tabela 2. A distância máxima aceitável para estacionar é 10 vezes o valor de α, conforme definido no modelo de raciocínio.

Tabela 2 – Configurações dos Cenários do Experimento 2

Cenário	α	Distância Máxima Aceitável
Cenário 1	5	50
Cenário 2	10	100
Cenário 3	15	150
Cenário 4	20	200
Cenário 5	25	250
Cenário 6	30	300
Cenário 7	35	350
Cenário 8	40	400
Cenário 9	45	450
Cenário 10	50	500

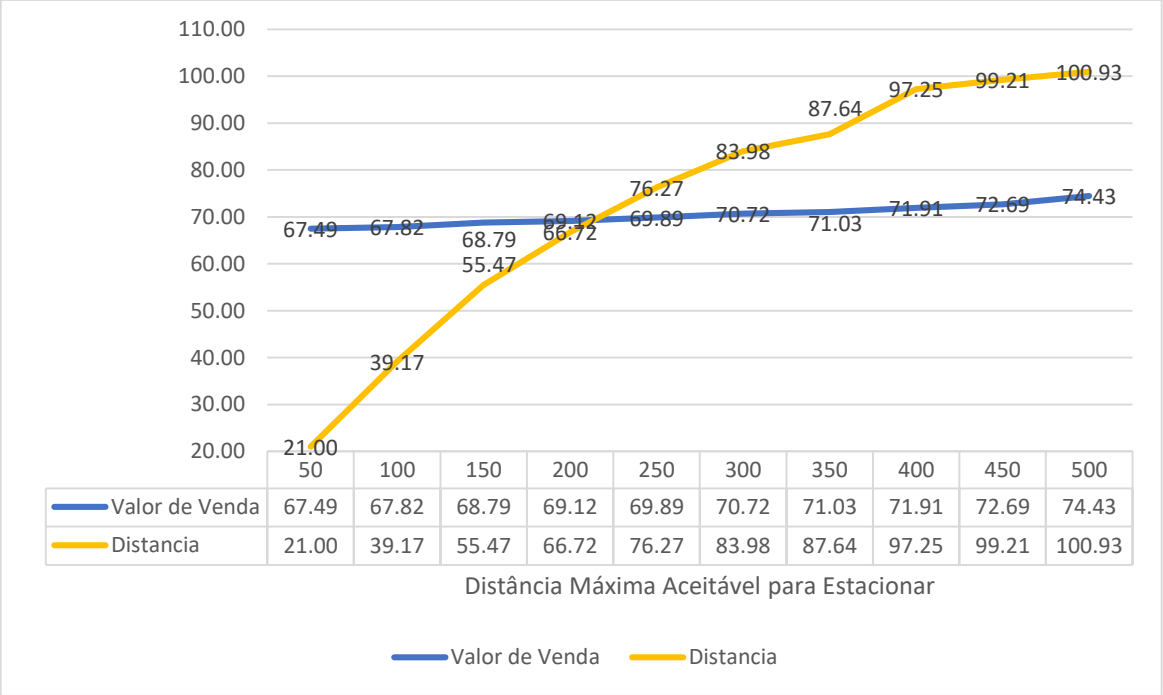
Fonte: Autoria Própria.

O Gráfico 3 apresenta os valores médios de venda das vagas e o valor médio da distância entre o ponto onde o agente deseja a vaga e o ponto onde o agente efetivamente estacionou em cada cenário de teste. Os valores médios de venda das vagas são multiplicados em 7 vezes, para que a escala fique mais próxima dos valores médio da distância entre o ponto onde o agente deseja a vaga e o ponto onde o agente efetivamente estacionou, fazendo assim com que a análise fique mais clara.

A partir da análise dos dados coletados da execução dos cenários de testes, foi possível constatar que, a distância média entre o ponto onde o agente deseja a vaga e o ponto onde o agente efetivamente estacionou tende a aumentar conforme o valor de α aumenta, como é possível observar no Gráfico 3. Este fenômeno acontece pois quanto maior o valor de α, maior será a distância máxima aceitável entre o ponto

onde o agente deseja a vaga e o ponto onde o agente efetivamente estaciona, fazendo com que a média deste valor aumente.

Gráfico 3 – Valor Médio de Venda das Vagas e Distância Média entre o Ponto de Desejo e o Ponto Estacionado em Relação a Distância Máxima Aceitável



Fonte: Autoria Própria.

Também foi possível constatar que, neste caso, quando maior é a distância máxima aceitável maior é o valor médio de venda das vagas, como é possível observar no Gráfico 3. Segundo a Função 1, do modelo de raciocínio apresentado, quanto maior o valor de α maior é o resultado da função, que é justamente o valor pago por uma vaga, o que faz com que a média do valor de venda das vagas cresça conforme o valor de α cresce, como é observado no gráfico.

5.4 EXPERIMENTO 3

Conforme mencionado, o objeto de estudo desta seção é a análise da variação do valor máximo que um driver está disposto a pagar por uma vaga, variável λ da Função 1. Para tanto, foram executados 10 cenários de testes, todos com 500 *drivers* e 160 vagas de estacionamento. As configurações das variáveis do modelo de raciocínio em todos os cenários de teste foram definidas sendo α = 25 e δ = 0,2. A

configuração dos parâmetros de cada cenário de teste pode ser visualizada na Tabela 3.

Tabela 3 – Configurações dos Cenários da Experimento 3

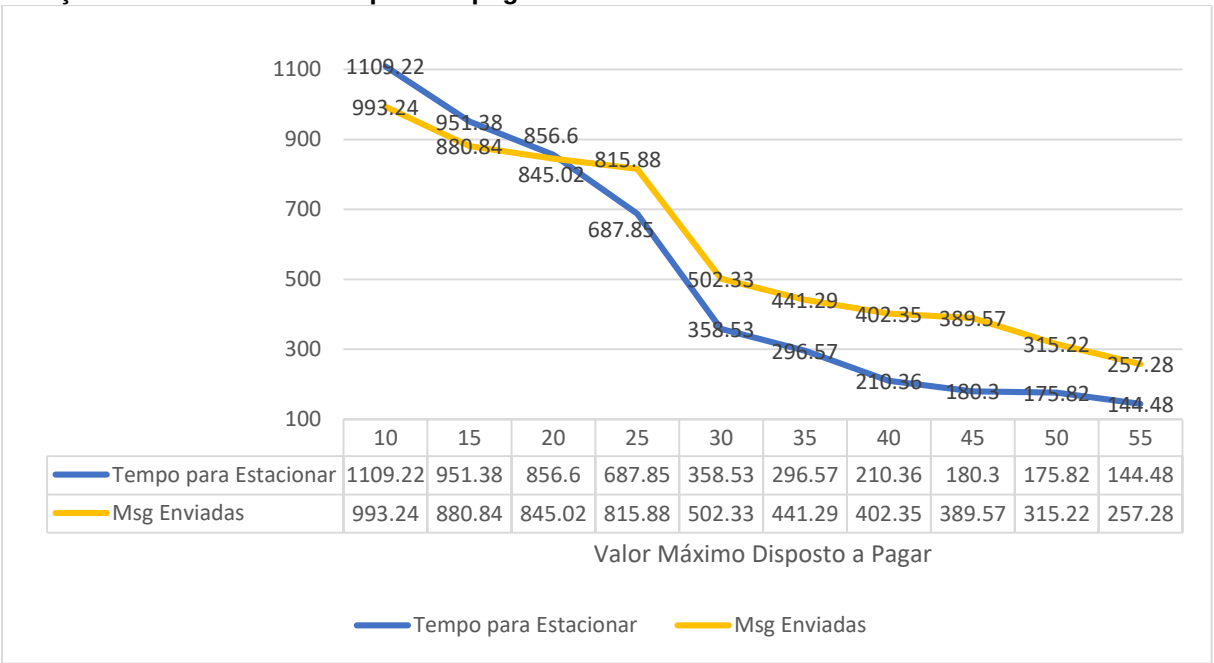
Cenário	Valor Máximo Disposto a Pagar
Cenário 1	10
Cenário 2	15
Cenário 3	20
Cenário 4	25
Cenário 5	30
Cenário 6	35
Cenário 7	40
Cenário 8	45
Cenário 9	50
Cenário 10	55

Fonte: Autoria Própria.

O Gráfico 4 apresenta a variação do número médio de mensagens enviadas e do tempo médio para estacionar de cada perfil de *driver*. Os valores médios do tempo para estacionar são multiplicados em 10 vezes, para que a escala fique mais próxima dos valores médios de mensagens enviadas, fazendo assim com que a análise fique mais clara.

A partir da análise dos dados coletados da execução dos cenários de testes, foi possível constatar que o número médio de mensagens enviadas para alocação da vaga de estacionamento tende a diminuir, conforme o valor máximo que um *driver* está disposto a pagar por uma vaga, como é possível observar no Gráfico 4. A troca de mensagens influi no tempo de alocação das vagas devido ao fato de que, quanto menor o número de mensagens trocadas para fechar uma negociação menos tempo levará a negociação. O tempo médio para alocação das vagas também tente a decair conforme o valor máximo que um *driver* está disposto a pagar por uma vaga.

Gráfico 4 – Número de Mensagens Médias Enviadas e Tempo Médio para Estacionar em Relação ao Valor Máximo disposto a pagar



Fonte: Autoria Própria.

Este fenômeno acontece pois, quanto maior o valor que um *buyer* está disposto a pagar por uma vaga de estacionamento, maior será o valor da oferta de compra de vagas, em razão de que quanto maior o valor desta oferta, maior é a chance de um seller aceitar. Fazendo com que o tempo médio de alocação de vagas decaia, conforme o valor que um *driver* está disposto a pagar por uma vaga.

5.5 EXPERIMENTO 4

Como citado, o objeto de estudo desta seção é análise da variação da faixa de fechamento de acordo, variável δ da Função 3. Para tanto, foram executados 10 cenários de testes, todos com 500 drivers e 160 vagas de estacionamento. As configurações das variáveis do modelo de raciocínio em todos os cenários de teste foram definidas sendo $\alpha = 25$ e $\lambda = 10$. A configuração dos parâmetros de cada cenário de teste pode ser visualizada na Tabela 4.

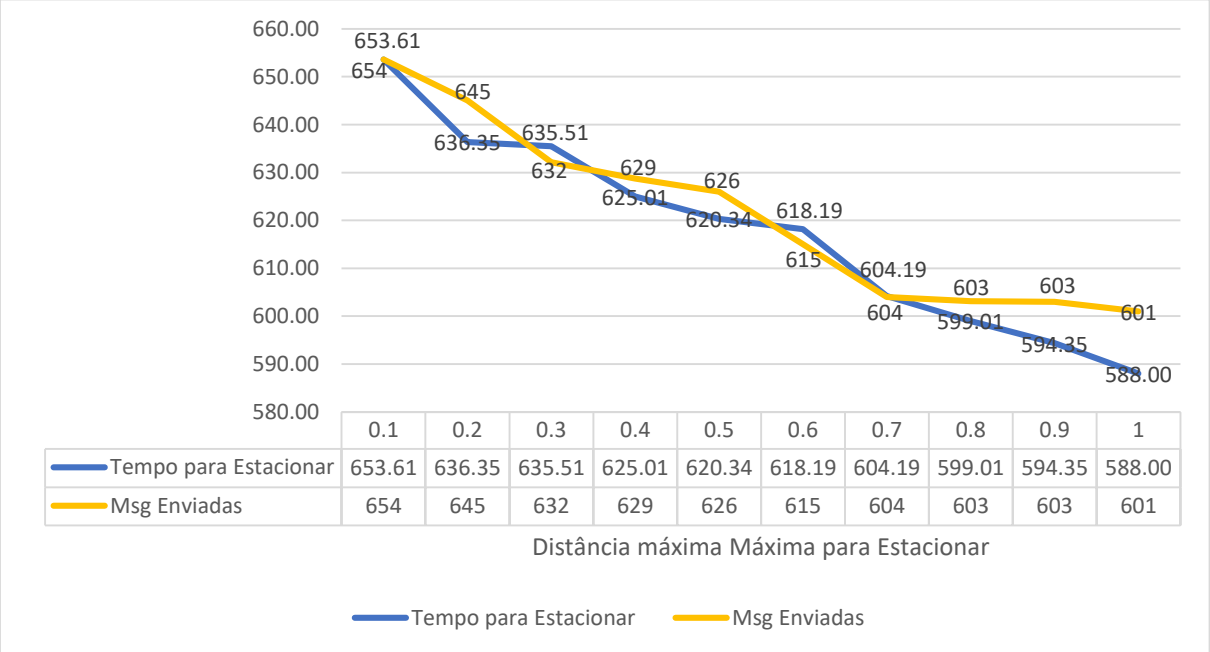
Tabela 4 – Configurações dos Cenários do Experimento 4

Cenário	Valor de δ
Cenário 1	0,1
Cenário 2	0,2
Cenário 3	0,3
Cenário 4	0,4
Cenário 5	0,5
Cenário 6	0,6
Cenário 7	0,7
Cenário 8	0,8
Cenário 9	0,9
Cenário 10	1

Fonte: Autoria Própria.

O Gráfico 5 apresenta a variação do número médio de mensagens enviadas e do tempo médio para estacionar em cada cenário. Os valores médios do tempo para estacionar são multiplicados em 7 vezes, para que a escala fique mais próxima dos valores médios de mensagens enviadas e assim fazendo com que a análise fique mais clara.

Gráfico 5 – Número de Mensagens Médias Enviadas e Tempo Médio para Estacionar em Relação ao Valor de δ



Fonte: Autoria Própria.

A partir da análise dos dados coletados da execução dos cenários de testes, foi possível constatar que o número médio de mensagens enviadas para alocação da vaga de estacionamento tende a diminuir conforme o valor de δ cresce, como é possível observar no Gráfico 5. Como constatado nas análises anteriores, a troca de mensagens influi no tempo de alocação das vagas, fazendo com que o tempo médio para alocação das vagas também tenda a decair, conforme o valor de δ.

Este decrescimento do tempo médio de alocação das vagas acontece pois, quanto maior é o valor de δ, maior é a faixa de contra propostas aceitas pelo agente *buyer*. Desse modo, as contra propostas tendem a ser aceitas mais rapidamente, diminuindo o tempo médio da negociação para alocação das vagas.

6 CONCLUSÃO

O desenvolvimento de projetos relacionados com o tema de Cidades Inteligentes é importante, melhorando a qualidade de vida das pessoas e contribuindo no desenvolvimento sustentável e inteligente das cidades. Estacionamentos Inteligentes podem contribuir neste aspecto, buscando alternativas para automatizar o gerenciamento e organização de vagas de estacionamento.

O Projeto MAPS desenvolve soluções para Estacionamentos Inteligentes, utilizando técnicas de Sistemas Multiagentes, com o intuito de auxiliar no desenvolvimento de soluções de mobilidade urbana para Cidades Inteligentes. O trabalho desenvolvido apresenta o desenvolvimento de um modelo de raciocínio e protocolo de negociação descentralizado, utilizado na implementação de um novo mecanismo de negociação descentralizado para o Projeto MAPS também apresentado neste trabalho, o qual possibilitou uma negociação com independência de um módulo central e gerou uma abordagem alternativa de alocação de vagas para o MAPS. Possibilitando a visualização de informações do sistema, por meio do desenvolvimento de uma interface gráfica e da conexão com o simulador de trânsito SUMO.

O *framework* JaCaMo foi utilizado pois se trata de uma plataforma robusta e flexível, que suporta o desenvolvimento dos agentes, do ambiente onde os agentes se encontram e da organização social do SMA, estes três níveis do *framework* foram importantes para o desenvolvimento do trabalho. O JaCaMo também facilitou a criação da interface gráfica e da interligação com o SUMO, devido ao fato de que a camada *Cartago* utiliza a linguagem Java, proporcionando um alto nível de abstração e o uso de bibliotecas no desenvolvimento dos recursos. A camada do *Moise* facilitou a implementação dos papeis no sistema, já que oferece suporte para a definição dos papeis e para utilização pelos agentes.

Com o propósito de validar o funcionamento do SMA e aperfeiçoar o modelo de raciocínio e protocolo de negociação, foram desenvolvidos testes com o Sistema Multiagente, que resultaram nos experimentos apresentadas no Capítulo 7. Conforme apresentado no capítulo anterior, foram feitos experimentos para analisar o funcionamento do sistema, aqui destacam-se as seguintes características: i) existe uma curva de tendência de crescimento de mensagens trocadas entre os agentes conforme aumenta o número de agentes no sistema, ocasionando um maior tempo

para a alocação das vagas de estacionamento; ii) conforme o número de agentes cresce existe uma tendência de que os agentes estacionem em uma vaga mais próxima do ponto de desejo; e iii) os agentes que estão dispostos a pagar mais por uma vaga tendem a estacionar mais rapidamente.

O desenvolvimento do SMA possibilitou a implantação de um mecanismo de negociação que aloca vagas levando em consideração características dos *drivers* (distância máxima para estacionar, valor máximo a pagar por uma vaga e faixa de fechamento de acordo), onde, no Capítulo 7, são apresentadas as consequências das alterações das características dos agentes na alocação de vagas de estacionamento.

6.1 TRABALHOS FUTUROS

O projeto MAPS continua em desenvolvimento e os seguintes trabalhos futuros podem ser elencados:

- Realizar mais experimentos com o MAPS-OPEN, como por exemplo experimentos variando o número de agentes e das variáveis do modelo de raciocínio simultaneamente, realizar os experimentos em outra configuração de *hardware*, realizar experimentos com dados reais de um estacionamento.
- Implementar mecanismos de negociação alternativos, utilizando outros protocolos de negociações e modelos raciocínios. A fim de elaborar comparação entre os diferentes mecanismos de negociação, tentando verificar quais têm melhor desempenho e determinar em quais contextos possuem este desempenho;
- Implementar um novo mecanismo de alocação de vagas inicial, quando as vagas não foram ocupadas por nenhum agente *driver*, pois no trabalho atual esta alocação acontece de maneira simples;
- Realizar comparações entre a versão inicial do MAPS, com mecanismo de negociação centralizado com a implementação atual, que utiliza um mecanismo de negociação descentralizado;
- Embarcar o Sistema Multiagente desenvolvido em uma plataforma de *hardware*;
- Desenvolver um aplicativo onde os agentes *drivers* podem requisitar vagas, informando suas preferências.

6.2 PUBLICAÇÕES

Este trabalho teve início com o desenvolvimento do artigo “Modelo de Raciocínio e Protocolo de Negociação para um Estacionamento Inteligente com Mecanismo de Negociação Descentralizado” (DUCHEIKO; ALVES, 2018), o qual foi apresentado no *Workshop*-Escola de Sistemas de Agentes, seus Ambientes e Aplicações (WESAAC). Neste trabalho foi apresentada uma versão inicial do modelo de raciocínio e protocolo de negociação utilizados no presente trabalho.

Utilizando a versão inicial do modelo de raciocínio e protocolo de negociação foi realizado a implementação de um Sistema Multiagente, o qual foi apresentado no trabalho “Implementação de Modelo de Raciocínio e Protocolo de Negociação para um Estacionamento Inteligente com Mecanismo de Negociação Descentralizado”, na Revista Junior de Iniciação Científica em Ciências Exatas e Engenharia (ICCEEG) (DUCHEIKO; BORGES; ALVES, 2018).

REFERÊNCIAS

ALONSO, F. et al. Measuring the Social Ability of Software Agents. In: Software Engineering Research, Management and Applications. Prague,. 2008.

AN, B.; GATTI, N.; LESSER, V. Alternating-offers bargaining in one-to-many and many-to-many settings. **Annals of Mathematics and Artificial Intelligence**, v. 77, n. 1, p. 67–103, 1 jun. 2016.

BATTY, M. et al. Smart Cities of the Future. 2012.

BLANGER, L.; JUNIOR, V.; PANISSON, A. Computer on the Beach. **Uma Aplicação para Gerenciamento de Motoristas Autônomos: Usufruindo da Escalabilidade Oferecida por Sistemas Multiagentes**, 2017.

BOISSIER, O. et al. Multi-agent oriented programming with JaCaMo. p. 747–761, 2011.

BORDINI, R. H.; HÜBNER, J. F.; WOOLDRIDGE, M. **Programming multi-agent systems in AgentSpeak using Jason**. John Wiley & Sons, 2007.

BRATMAN, M. **Intention, plans, and practical reason**. Cambridge, MA: Harvard University Press, 1987.

CÂMARA, Á. et al. European Simulation and Modelling Conference. **Comparing Centralized and Decentralized Multi-Agent Approaches to Air Traffic Control**, n. 28, 2014.

CARTAGO. **CArtAgO**. Disponível em: <http://cartago.sourceforge.net/>. Acesso em: 5 jan. 2012.

CASTRO, L. F. S. DE; ALVES, G. V.; BORGES, A. P. Utilização de grau de confiança entre agentes para alocação de vagas em um Smart Parking. 10º Workshop-Escola de Sistemas de Agentes, seus Ambientes e aplicações. Maceió - AL. 2016a.

CASTRO, L. F. S. DE; ALVES, G. V.; BORGES, A. P. `. 2016b.

DUCHEIKO, F.; ALVES, G. Modelo de Raciocínio e Protocolo de Negociação para um Estacionamento Inteligente com Mecanismo de Negociação Descentralizado. **WESAAC: Workshop-Escola de Sistemas de Agentes, seus Ambientes e apliCações**, n. 12, p. 250–256, 2018.

DUCHEIKO, F. F.; ALVES, G. V. Desenvolvimento de Interface Gráfica para Gerenciamento de um Smart Parking. 1º Workshop de Pesquisa em Computação dos Campos Gerais. Ponta Grossa - PR. Set. 2016. 2016.

DUCHEIKO, F. F.; BORGES, A. P.; ALVES, G. V. Implementação de Modelo de Raciocínio e Protocolo de Negociação para um Estacionamento Inteligente com Mecanismo de Negociação Descentralizado. **Revista Junior de Iniciação Científica em Ciências Exatas e Engenharia**, v. 1, n. 19, p. 25–32, 2018.

ENDRISS, U. **Monotonic Concession Protocols for Multilateral Negotiation**. Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems. **Anais**...: AAMAS '06.New York, NY, USA: ACM, 2006Disponível em: <http://doi.acm.org/10.1145/1160633.1160702>

FARATIN, P.; SIERRA, C.; JENNINGS, N. R. Robotics and Autonomous Systems. **Negotiation Decision Function for Autonomous Agents**, v. 24. North-Holland, p. 159–182, 1998.

FERBER, J. **Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence**. 1st. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

FIPA. FIPA Contract Net Interaction Protocol Specification. 2002.

HEIJMEIJER, A.; ALVES, G. V. **Desenvolvimento de um Middleware entre a ferramenta SUMO e o framework JaCaMo**. Anais do XI Workshop-School on Agents, Environments, and Applications. **Anais**... In: WESAAC. São Paulo: 2017Disponível em: <http://wesaac2017.c3.furg.br>

HÜBNER, J. F.; SICHMAN, J. S.; BOISSIER, O. Developing organised multi-agent systems using the moise+ model: Programming issues at the system and agent levels. **International Journal of Agent-Oriented Software Engineering**, 2007.

HUHNS, M. N.; STEPHENS, L. M. Multiagent systems and societies of agents. In: WEISS, G. (Ed.). . Cambridge, MA, USA: MIT Press, 1999. p. 79–120.

JACAMO. **The JaCaMo approach.** Disponível em: <http://jacamo.sourceforge.net/?page_id=40>.

JASON. **Jason Book Website**. Disponível em: <http://jason.sourceforge.net/jBook/jBook/Home.html>. Acesso em: 5 jan. 2012.

KOSTER, A.; KOCH, F.; BAZZAN, A. L. C. Incentivising Crowdsourced Parking Solutions. **Citizen in Sensor Networks: Second International Workshop, CitiSens 2013, Barcelona, Spain, September 19, 2013, Revised Selected Papers**, p. 36–43, 2013a.

KOSTER, A.; KOCH, F.; BAZZAN, A. L. C. Incentivising Crowdsourced Parking Solutions. 2013b.

KRAJZEWICZ, D. et al. Recent Development and Applications of SUMO - Simulation of Urban MObility. **International Journal On Advances in Systems and Measurements**, v. 5, n. 3&4, p. 128–138, dez. 2012.

MARTINS, R.; MENEGUZZI, F. International Conference on Industrial Informatics. **A smart home model using JaCaMo framework**, n. 12. Porto Alegre-RS, 2014.

MOISE. **Moise organisational model**. Disponível em: <http://moise.sourceforge.net/>. Acesso em: 5 jan. 2012.

NAPOLI, C. D.; NOCERA, D. D.; ROSSI, S. Agent negotiation for different needs in smart parking allocation. Springer, 2014.

NOCERA, D. D.; NAPOLI, C. D.; ROSSI, S. A Social-Aware Smart Parking Application. CEUR Workshop Proceedings. v. 1260, 2014.

O'HARE, G. M. P.; JENNINGS, N. R. (EDS.). **Foundations of Distributed Artificial Intelligence**. New York, NY, USA: John Wiley & Sons, Inc., 1996.

OLIVER, J. R. A Machine-Learning Approach to Automated Negotiation and Prospects for Electronic Commerce. **Journal of Management Information Systems**, v. 13, n. 3, p. 83–112, 1996.

PNUELI, A. In Information Processing. **Specification and development of reactive systems**, v. 86, p. 845–858, Elsevier- Amsterdam, 1986.

REHMAN, M.; SHAH, M. A. International Conference on Automation & Computing. **A Smart Parking System to Minimize Searching Time, Fuel Consumption and CO2 Emission**, n. 23. University of Huddersfield, 2017.

REVATHI, G.; DHULIPALA, V. R. S. Smart Parking Systems and Sensors: A Survey. 2012.

RIZVI, S. R.; ZEHRA, S.; OLARIU, S. ASPIRE: An Agent-Oriented Smart Parking Recommendation System for Smart Cities. **IEEE Intelligent Transportation Systems Magazine**, p. 1–1, 2018.

ROLOFF, M. L. et al. **A multi-agent system for the production control of printed circuit boards using JaCaMo and Prometheus AEOlus**. 12th IEEE International Conference on Industrial Informatics, INDIN 2014, Porto Alegre, RS, Brazil, July 27-30, 2014. **Anais**...2014Disponível em: <https://doi.org/10.1109/INDIN.2014.6945514>

RUSSEL, S. J.; NORVIG, P. **Artificial Intelligence - A Modern Approach**. New Jersey: Pearson Education, 2010.

SHEHORY, O. Architectural Properties of Multi-Agent Systems. The Robotics Institute Carnegie Mellon University. Pittsburgh - Pennsylvania. 1998.

SMITH, R. G. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. p. 357–366, 1988.

VALKANAS, G.; NATSIAVAS, P.; BASSILIADES, N. **A Collision Detection and Resolution Multi Agent Approach Using Utility Functions**. 2009 Fourth Balkan Conference in Informatics, BCI 2009, Thessaloniki, Greece, 17-19 September 2009. **Anais**...2009Disponível em: <https://doi.org/10.1109/BCI.2009.30>

WINIKOFF, M.; PADGHAM, L. The Prometheus Methodology. In: BERGENTI, F.; GLEIZES, M.-P.; ZAMBONELLI, F. (Eds.). . **Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook**. Boston, MA: Springer US, 2004. p. 217–234.

WOLLKIND, S.; VALASEK, J.; IOERGER, T. R. **Automated conflict resolution for air traffic management using cooperative multiagent negotiation**. In: AIAA Guidance, Navigation, and Control Conference. **Anais**...2004

WOOLDRIDGE, M. Multiagent Systems. In: WEISS, G. (Ed.). . Cambridge, MA, USA: MIT Press, 1999. p. 27–77.

WOOLDRIDGE, M. **An Introduction to MultiAgent Systems**. 2nd. ed. New York: J. Wiley, 2009.

XIE, A.; LI, Y.; SUN, W. Review on the Theory of MultiAttribute E-Auction. v. 9, n. 4, p. 621–643, 2004.

APÊNDICE A - Código especificação organizacional do sistema

CÓDIGO DA ESPECIFICAÇÃO ORGANIZACIONAL DO SISTEMA

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <?xml-stylesheet  href="http://moise.sourceforge.net/xml/os.xsl"
   type="text/xsl" ?>
4
5  <organisational-specification
6      id="organization"
7      os-version="0.8"
8
9      xmlns='http://moise.sourceforge.net/os'
10     xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
11     xsi:schemaLocation='http://moise.sourceforge.net/os
                           http://moise.sourceforge.net/xml/os.xsd'
12 >
13
14 <structural-specification>
15
16 <role-definitions>
17     <role id="driver" />
18     <role id="buyer">
19     <extends role="driver"/>
20     </role>
21     <role id="seller">
22     <extends role="driver"/>
23     </role>
24 </role-definitions>
25
26 <group-specification id="parkingLot">
27
28     <roles>
29         <role id="buyer"/>
30         <role id="seller"/>
31         <role id="parkingSpotController"/>
32     </roles>
33
```

```
34      <links>
           <link   from="buyer"   to="seller"   type="communication"
35  scope="intra-group" bi-dir="true"/>
           <link   from="seller"   to="buyer"   type="communication"
36  scope="intra-group" bi-dir="true"/>
           <link       from="buyer"       to="parkingSpotController"
37  type="authority"  scope="intra-group" bi-dir="false"/>
           <link       from="seller"       to="parkingSpotController"
38  type="authority"  scope="intra-group" bi-dir="false"/>
39      </links>
40
41      <formation-constraints>
42         <compatibility from="buyer" to="seller" />
43         <compatibility from="seller" to="buyer" />
44         <compatibility from="parkingSpotController" to="seller" />
45         <compatibility from="parkingSpotController" to="buyer" />
46      </formation-constraints>
47  </group-specification>
48
49  </structural-specification>
50
51  <functional-specification>
52      <scheme id="parkingLotScheme">
53
54          <goal id="ParkingNegotiation">
55
56              <plan operator="sequence">
57
58                  <goal id="arriveParking">
59                     <plan operator="sequence">
60                        <goal id="answerStatusParkingSpot"></goal>
61                     </plan>
62                  </goal>
63
64                  <goal id="startNegotiation">
65                      <plan operator="sequence">
66                         <goal id="offerSpot"></goal>
```

```
                        <goal id="receiveOffer"></goal>
                        <goal id="analyzeOffer"></goal>
                        <goal id="generateCounterOffer"></goal>
                        <goal id="ultimatum"></goal>
                    </plan>
                </goal>

                <goal id="leaveParking"/>

            </plan>

        </goal>

        <mission id="arrival">
            <goal id="arriveParking"/>
        </mission>

        <mission id="negotiate">
            <goal id="startNegotiation"/>
        </mission>

        <mission id="departure">
            <goal id="leaveParking"/>
        </mission>


    </scheme>
</functional-specification>

<normative-specification>
    <norm id="norm1" type="obligation" role="buyer" mission="arrival"/>

    <norm id="norm2" type="permission" role="driver" mission="negotiate"/>

```

```
        <norm      id="norm3"      type="permission"      role="seller"
101 mission="departure"/>
102 </normative-specification>
103
104 </organisational-specification>
```