UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ANGELO BITTENCOURT MARINI FILHO

IMPLEMENTAÇÃO E ANÁLISE DE MECANISMOS DE NEGOCIAÇÃO EM UM SISTEMA MULTIAGENTE APLICADO A ALOCAÇÃO DE VAGAS EM UM ESTACIONAMENTO INTELIGENTE

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2018

ANGELO BITTENCOURT MARINI FILHO

IMPLEMENTAÇÃO E ANÁLISE DE MECANISMOS DE NEGOCIAÇÃO EM UM SISTEMA MULTIAGENTE APLICADO A ALOCAÇÃO DE VAGAS EM 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. André Pinz Borges

PONTA GROSSA

2018

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa
Diretoria de Graduação e Educação Profissional
Departamento Acadêmico de Informática
Bacharelado em Ciência da Computação

TERMO DE APROVAÇÃO

IMPLEMENTAÇÃO E ANÁLISE DE MECANISMOS DE NEGOCIAÇÃO EM UM SISTEMA MULTIAGENTE APLICADO À ALOCAÇÃO DE RECURSOS EM UM ESTACIONAMENTO INTELIGENTE

por

ANGELO BITTENCOURT MARINI FILHO

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 04 de junho de 2018 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.

_____
Professor Dr. André Pinz Borges
Orientador(a)

_____
Professor Dr. Gleifer Vaz Alves
Membro titular

_____
Professor MSc. Rafael dos Passos Canteri
Membro titular

_____
Professora Dra. Helyane Bronoski Borges
Responsável pelo Trabalho de Conclusão de Curso

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

AGRADECIMENTOS

Agradeço ao professor Gleifer Vaz Alves, o qual me guiou desde envolvimento com as pesquisas, sempre disposto e solícito, sendo o principal colaborador para a conclusão deste trabalho.

Ao professor André Pinz Borges o qual me orientou no desenvolvimento das etapas finais do trabalho, tendo contribuição ímpar no resultado apresentado,

Aos colegas do Grupo de Pesquisa em Agentes de Software, em especial os alunos Lucas Castro e Wesley Gonçalves, que sempre estiveram disponíveis para discussões e dúvidas.

E por fim à minha família, minha namorada e aos amigos, que me deram suporte durante todos os desafios da graduação.

RESUMO

MARINI, Angelo. **Implementação e Análise de Mecanismos de Negociação em um Sistema Multiagente Aplicado a Alocação de Vagas em um Estacionamento Inteligente**. 2018. 102f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2018.

Em busca de soluções para os problemas gerados por trânsito parado nas grandes cidades, projetos inovadores desenvolvem estacionamentos inteligentes com o objetivo de economizar o tempo de motoristas e liberar vias movimentadas, por meio da automatização da procura e alocação de vagas. O projeto MAPS pesquisa possíveis soluções, utilizando a abordagem de sistemas multiagentes para criação de um estacionamento que automatize a alocação de vagas à motoristas. Até o momento o MAPS resolve o problema da alocação para as vagas com uma modelagem do estacionamento utilizando agentes que representam os motoristas e o gerenciador do sistema. No desenvolvimento deste trabalho implementou-se mecanismos de negociação utilizados em sistemas multiagentes para encontrar soluções em relação a alocação de vagas no *smart parking*. Foram implementados utilizando o *framework* JaCaMo três mecanismos de negociação (leilão inglês, leilão holandês e o *Contract Net Protocol*), e construídos cenários de teste para a análise do tempo, troca de mensagens e o custo aos motoristas que cada mecanismo de negociação apresentou como resultados. Com a análise dessas abordagens observou-se que cada mecanismo apresenta melhores resultados em diferentes cenários, sendo assim necessário a implementação de um mecanismo de negociação híbrido o qual otimize o processo de alocação do recurso.

Palavras-chave: Sistema Multiagente. Estacionamento Inteligente. Negociação. Leilão. Protocolo Contract Net.

ABSTRACT

MARINI, Angelo. **Implementação and Analysis of Negotiation Mechanisms in Multy-agent Systems Applied to Spots Allocation in Smart Parkings**. 2018. 102s. (Bachelor's Degree in Computer Science) - Federal University of Technology - Paraná. Ponta Grossa, 2018.

Searching for solutions to problems related to slow traffic, innovative projects develop smart parkings willing to save drivers time and free the traffic in busy streets. MAPS project looks for solutions to these problems by the approach of multi-agent systems to create a smart parking that automates the allocation of car spots to drivers. To date, MAPS has solved the problem of spots allocation by modeling agents that represents drivers and the system manager. The development of this work implemented multi-agent system negotiation mechanisms to find solutions related to car spots allocation for smart parking. Three negotiation mechanisms (English auction, Dutch auction and Contract Net Protocol) were implemented using the JaCaMo framework, and test scenarios were constructed for the analysis of time, message exchange and the cost to the drivers that each negotiation mechanism presented as results. With the analysis of these approaches, it was observed that each mechanism presents better results in different scenarios, so it is necessary to implement a hybrid negotiation mechanism which optimizes the process of resource allocation.

Keywords: Multi-Agent System. Smart Parking. Negotiation. Auction. Contract Net Protocol.

LISTA DE ILUSTRAÇÕES

LISTA DE CÓDIGOS

LISTA DE TABELAS

LISTA DE GRÁFICOS

LISTA DE ABREVIATURAS

AI	*Auction Identifier*
BDI	*Belief, Desire and Intention*
CDT	*Credits*
CNP	*Contract Net Protocol*
Cartago	*Common ARTifact infrastructure for AGents Open environments*
DF	*Decrease Factor*
FIPA	Foundation for Intelligent Physical Agents
FPSB	*First-Price Sealed-Bid*
FS	*Free Spot*
GPAS	Grupo de Pesquisa em Agentes de Software
ID	Identificador
IF	*Increase Factor*
LAFS	*Last Auction Free Spots*
JaCaMo	Jason, Cartago e Moise
MAPS	*Multiagent Parking System*
MS	*Messager Sent*
MV	*Max Value*
SL	*Sort List*
SV	*Spot Value*
SMA	Sistema Multiagente
TCFP	*Time For New Call For Proposal*
TR	*Total Received*
TS	*Time to Spend*
TTR	*Time to Request*
TWT	*Total Waited Time*

SUMÁRIO

1 INTRODUÇÃO

Nas próximas décadas a maioria da população deve viver em ambientes urbanos, segundo a Organização das Nações Unidas (ONU) aponta que a população mundial estará próxima dos 10 bilhões em 2050 e dois terços das pessoas viverão em áreas urbanas. Com o intuito de melhorar a qualidade de vida, reduzir desperdícios e melhorar as condições econômicas destes cidadãos, propostas inovadoras projetam cidades inteligentes utilizando tecnologias de informação e comunicação (STIMELL, 2016). Um dos principais setores no qual cidades inteligentes buscam solucionar problemas é o do trânsito. Na cidade de São Paulo são perdidos R$18 bilhões por ano decorrente de congestionamentos (LEITE, 2014), sendo 30% desse congestionamento decorrente de motoristas buscando por vagas para estacionar (SILVA, 2015).

Propondo facilitar e acelerar o estacionamento de veículos, surgem os estacionamentos inteligentes, ou *smart parking*. Estes estacionamentos almejam pelo uso destas tecnologias de informação e comunicação reduzir o desperdício de tempo gasto em longas filas na busca por vagas.

Uma abordagem possível para o desenvolvimento de estacionamentos inteligentes é a utilização de Sistemas Multiagentes (SMA), onde os agentes computacionais autônomos podem simular as ações de participantes do estacionamento (como motoristas e administradores). Um SMA é um sistema computacional, no qual agentes inteligentes são inseridos em um ambiente, sendo capazes de executar ações autônomas nesse ambiente para alcançar o objetivo que lhe foi atribuído (WOOLDRIDGE, 2009).

O Grupo de Pesquisa em Agentes de Software – UTFPR-PG (GPAS) utiliza a abordagem SMA no projeto de pesquisa denominado *MultiAgent Parking System* (MAPS). O projeto MAPS tem como objetivo “utilizar as características de um estacionamento e as informações disponíveis a fim de criar um SMA, capaz de organizar o uso do estacionamento, tentando alocar as vagas de uma forma mais adequada e que incentive o motorista a cooperar com o sistema” (GONÇALVES, ALVES, 2015).

A primeira versão do projeto MAPS apresentada em (CASTRO, 2015) foi implementada utilizando o *framework* JaCaMo (JACAMO, 2011), esse *framework* composto por três módulos que permintem implementar as ações do agente, o

ambiente em que os agentes estão inseridos e as normas de organização do sistema multiagente. Esses recursos possibilitam o desenvolvimento dos agentes baseado no modelo BDI (*Belief, Desire, Intention*), modelo capaz de empregar conceitos do raciocínio humano aos agentes que compões o sistema (BORDINI; HüBNER; WOOLDRIDGE, 2007). Nessa versão a alocação de vagas é feita aos agentes motoristas que fazem requisições por vagas, e se há *drivers* na fila de espera quando uma vaga é liberada a vaga era é alocada ao agente com o maior valor de confiança. O valor confiança é a probabilidade subjetiva de que um agente irá executar a tarefa que lhe foi destinada, normalmente denotada por um valor numérico que indica quão confiável um agente é (GRANATYR et al., 2015).

O objetivo principal deste trabalho é propor uma extensão ao MAPS por meio da implementação de mecanismos de negociação ainda não explorados, aprofundando essa busca pela forma mais adequada de fazer a alocação para os agentes que competem por vagas.

Assim como a negociação está constantemente presente na vida do ser humano mirando concretizar acordos, está presente nos SMA. Geralmente os agentes precisam interagir com outros agentes para alcançar os seus objetivos, assim a negociação proporciona aos agentes chegarem a um consenso em um determinado problema. Os mecanismos de negociação entre agentes são utilizados com a intenção de solucionar conflitos e alocar tarefas e recursos (BASTOS, 1998).

Uma forma tradicional de negociação em ambientes competitivos são os leilões, mecanismos de negociação que seguem um protocolo de interação. Um leilão consiste em um sistema de regras e papéis definidos, onde as regras definem e coordenam as interações no processo de negociação entre o leiloeiro e o participante (JENNINGS et al., 1998). O papel do leiloeiro é iniciar a negociação, oferecendo os recursos ou objetos com a intenção de venda para um ou mais participantes interessados em adquirir. Já os participantes têm como função ofertar propostas ao leiloeiro para receber o produto leiloado. Cada agente, ao exercer um dos papéis, utiliza diferentes estratégias ou modelo de raciocínio para alcançar seu objetivo.

Para o desenvolvimento desse trabalho o leiloeiro é o agente *manager*, gerente do estacionamento, responsável por iniciar a negociação oferecendo vagas aos participantes e também por controlar o S*mart Parking*. Enquanto os participantes

são os agentes *drivers*, que representarão os motoristas, que tem como objetivo adquirir as vagas disponibilizadas pelo *manager*.

A implementação deste trabalho em certas instancias é uma extensão do trabalho desenvolvido por (CASTRO, 2015) e neste documento é denominada MAPS-*Auctions*. Assim, fica clara a distinção das denominações MAPS e MAPS-Auctions ao longo deste documento.

Para o desenvolvimento do MAPS-*Auctions* foram implementados três tipos de negociação:

1. Leilão Inglês: O *manager* oferece a vaga, e espera os lances que os *drivers* enviarão disputando pela vaga. Sempre um novo lance deve ser maior do que os ofertados anteriormente;
2. Leilão Holandês: O *manager* oferece o produto por um preço inicial extremamente alto e reduz o mesmo continuamente até chega a um valor que algum *driver* se disponha a pagar pela vaga;
3. *Contract Net Protocol* (CNP): Mecanismo de negociação utilizado para alocar recursos em sistemas distribuídos e adaptado à SMA é um protocolo de licitação onde o *manager* oferece a vaga aos *drivers*, espera que enviem a proposta, e seleciona a melhor oferta.

A implementação e análise destes mecanismos de negociação, tem como foco contribuir com o projeto MAPS, apresentando as vantagens e desvantagens de cada um deles na distribuição das vagas.

1.1 OBJETIVOS DO TRABALHO

A seguir são descritos o objetivo geral e os objetivos específicos para a realização do trabalho.

1.1.1 Objetivo Geral

Implementar e analisar mecanismos de negociação em um estacionamento inteligente.

1.1.2 Objetivos Específicos

Para atingir o objetivo geral do trabalho, foram gerados objetivos específicos a serem cumpridos. Esses objetivos estão enumerados a seguir:

1. Implementar o mecanismo de negociação *ContractNet Protocol* no contexto do projeto MAPS;
2. Desenvolver o leilão Inglês ao contexto do projeto MAPS;
3. Implantar o leilão Holandês ao contexto do projeto MAPS;
4. Construir cenários de testes para as simulações
5. Análisar os resultados apresentados pelos modelos implementados.
6. Apresentar uma comparação entre os modelos que indiquem melhores maneiras de alocar recursos em SMA nos diferentes cenários.

1.2 JUSTIFICATIVA

O desperdício de recursos com a dificuldade de motoristas encontrarem vagas é notório (LEITE, 2014). O projeto MAPS busca alternativas computacionais, por meio de SMA, para tentar uma melhor organização e gerenciamento de estacionamentos inteligentes.

Existem lacunas a serem preenchidas na implementação inicial do MAPS, e mecanismos a serem testados para alcançar a melhora da organização e gerenciamento. Por exemplo, o formato simples da negociação das vagas.

Este trabalho visa contribuir com a implementação de novas estratégias de negociação das vagas, com o intuito de apresentar novos resultados com relação a distribuição de recursos do SMA.

Os mecanismos de negociação (CNP, leilão inglês e holandês) foram escolhidos para serem implementados, pois possuem bastante relevância na literatura de SMA. Outro motivo é o de as três abordagens funcionarem como negociações centralizadas, com um agente gerenciando o processo, assim como o trabalho proposto utilizará um agente para gerenciar o s*mart parking.*

1.3 ORGANIZAÇÃO DO TRABALHO

O desenvolvimento deste trabalho está organizado em oito capítulos que apresentaram o seguinte conteúdo:

- Capitulo 2: Descreve as principais definições de SMA;
- Capítulo 3: Apresenta o *framework* JaCaMo e suas plataformas;
- Capítulo 4: Definições das negociações em sistemas multiagentes, e descrições de alguns mecanismos de negociação;
- Capítulo 5: Contextualização do trabalho no projeto MAPS;
- Capítulo 6: Desenvolvimento do MAPS-*Auctions;*
- Capítulo 7: Resultados das simulações aplicadas aos mecanismos implementados;
- Capítulo 8: Considerações finais sobre o trabalho.

2 SISTEMAS MULTIAGENTES

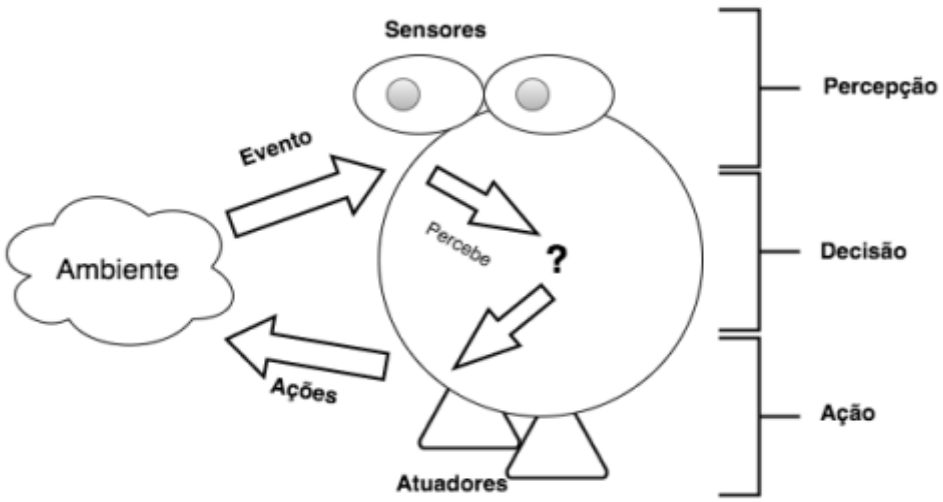
Os Sistemas Multi-Agente (SMA) são compostos por múltiplos agentes, classificados como inteligentes, que possuem comportamento autônomo e interagem com os outros agentes presentes no ambiente e com o próprio ambiente almejando alcançar seus objetivos. Nesta seção são apresentadas as principais definições sobre agentes e SMA.

2.1 AGENTES INTELIGENTES

Um agente é uma entidade real ou virtual, capaz de agir num ambiente e se comunicar com outros agentes, motivado por um conjunto de objetivos. Possui recursos próprios, é capaz de perceber seu ambiente através de sensores, e seu comportamento tende a atingir objetivos com o uso da competência e os recursos que dispõe. Para atingir os objetivos, ele considera os resultados de suas funções de percepções e comunicação (FERBER, GASSER, 1991).

A Figura 1 ilustra a representação gráfica de um agente. O agente recebe uma entrada pelo sensor, executa a estratégia de raciocínio definida, e produz como saída uma ação que afeta o ambiente. Um agente não possui controle completo do ambiente, podendo apenas influenciar uma parte do mesmo.

Figura 1 – Representação de um agente inteligente



Fonte: Adaptado de Wooldridge (2009)

Conforme apresentado em (WOOLDRIDGE, 2009), são três as principais capacidades dos agentes inteligentes:

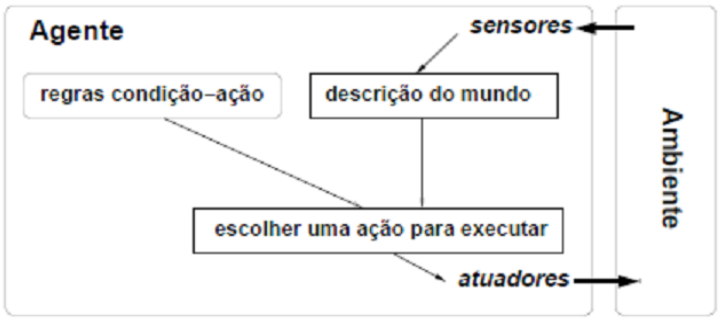
1. Reatividade: são capazes de perceber o ambiente no qual estão inseridos, e responder em tempo hábil as mudanças que ocorrem para alcançar o seu objetivo.
2. Proatividade: são capazes de tomar a iniciativa em uma ação para alcançar seu objetivo.
3. Habilidade Social: são capazes de interagir com outros agentes (e possivelmente humanos), para alcançar seu objetivo.

A partir dessas capacidades, Wooldridge (2009) define três tipos de agentes: reativos, racionais e híbridos. Nas subseções abaixo são apresentadas essas definições.

2.1.1 Agente Reativo

O agente reativo desenvolve conhecimento a partir de interações com o ambiente onde está situado, sem necessitar de um modelo pré-estabelecido. O agente reativo geralmente toma as suas decisões em tempo real, com base num conjunto de informações limitado e regras simples de ações que permitem selecionar um dado comportamento (RUSSEL; NORVIG, 2003). Neste modelo, representado na Figura 2, a informação captada pelos sensores é utilizada diretamente no processo de decisão, não sendo criada uma representação simbólica do ambiente.

Figura 2 – Arquitetura resumida de um agente reativo



Fonte: Russel e Norvig (2003).

O agente reativo possui duas características principais:

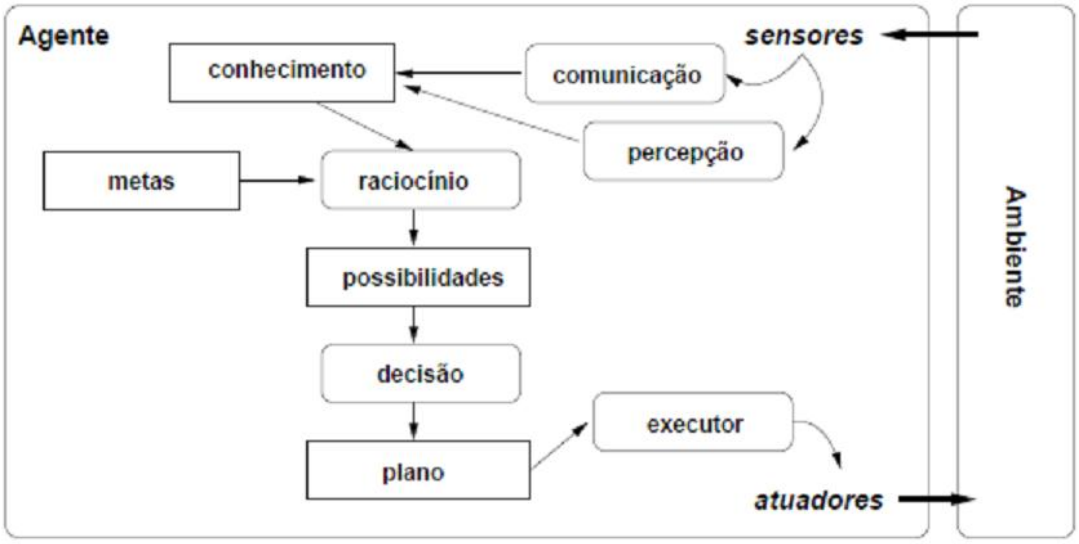
1. A tomada de decisão do agente deve ser de acordo ao conjunto de comportamentos para a execução de tarefas, isto é, cada comportamento deve ser pensado como uma função individual de ação. Logo, o agente, continuamente toma consciência do ambiente e traduz essa percepção para uma ação a ser executada
2. Vários comportamentos podem ser acionados simultaneamente. Assim, deverá existir um mecanismo para selecionar a melhor ação a ser executada em cada momento.

As vantagens do agente reativo são a simplicidade, robustez contra falhas, e ser um sistema mais simples para desenvolver. A principal desvantagem desse agente é a impossibilidade de agir pensando em planos de longo prazo.

2.1.2 Agente Racional

Os agentes racionais seguem a abordagem clássica da Inteligência Artificial, onde os agentes atuam com pouca autonomia e possuem modelos simbólicos dos seus ambientes. O agente possui a representação do ambiente, uma base de conhecimento semelhante a um banco de dados.

Figura 3 – Arquitetura resumida de um agente racional



Fonte: Russel e Norvig (2003).

A base de conhecimento do agente possui as informações sobre o ambiente, que são mapeadas de acordo com as percepções. Após a interpretação da percepção proveniente do ambiente, o agente utiliza esta informação para manter atualizada uma representação simbólica do estado do ambiente. Este estado em conjunto com os objetivos do agente são utilizados como forma de gerar as possíveis ações a serem executadas pelo agente e selecionar aquela que lhe parece mais apropriada.

Esse modelo considera os agentes como parte de um sistema baseado em conhecimento, as decisões dos agentes são realizadas através de raciocínio lógico.

2.1.3 Agente Híbrido

Como o próprio nome sugere, o agente híbrido é uma união dos agentes reativos e racionais. Os agentes reativos têm uma limitação significativa que é a sua dificuldade em implementar comportamento orientado aos planos dos objetivos. Os agentes racionais são baseados em mecanismos de raciocínio simbólico complexo e se tornam incapazes de uma reação imediata a estímulos exteriores. Um agente híbrido combina estas duas características, atuando tanto de forma rápida frente à situações que possam surgir no ambiente, como também possui capacidades de agentes racionais para armazenar estados prévios, e utilizá-los em planos em que informações prévias são necessárias (WOOLDRIDGE, 2009).

2.2 ARQUITETURA BDI

A arquitetura *Belief, Desire and Intention* (BDI), (Crença, Desejo e Intenção), (BRATMAN; ISRAEL; POLLACK, 1988) é inspirada num modelo comportamental humano desenvolvido pelo campo da Filosofia. É uma arquitetura comum para agentes racionais, onde o agente possui crenças, desejos e intenções, e utiliza noções mentais a fim de representar raciocínio prático (WOOLDRIDGE, 2009).

Segundo Reis (2003), as crenças, desejos e intenções são respectivamente entendidos como:

- Crenças: As informações que o agente possui sobre si, outros agentes e o ambiente. Por exemplo: o agente *driver* possui uma crença de quantos

créditos ele possui, enquanto o *manager* possui crenças com informações a respeito de cada vaga.

- Desejos: Objetivo que o agente almeja alcançar. Os objetivos dos agentes resultam de um processo de raciocínio, que consiste numa escolha de um subconjunto dos desejos que são consistentes e atingíveis. O *driver* possui como objetivo receber uma vaga, depois passa a ter como objetivo estacionar, e por fim possui o objetivo de liberar a vaga.
- Intenções: Conjuntos de ações ou tarefas que o agente executará para alcançar o seu objetivo. São os planos executados pelos agentes.

Uma forma de implementar um agente utilizando a arquitetura BDI é com o uso de linguagens de programação de agentes, como a linguagem Jason, usada no *framework* JaCaMo, a ser apresentado no Capítulo 3.

2.3 DEFINIÇÃO DE SMA

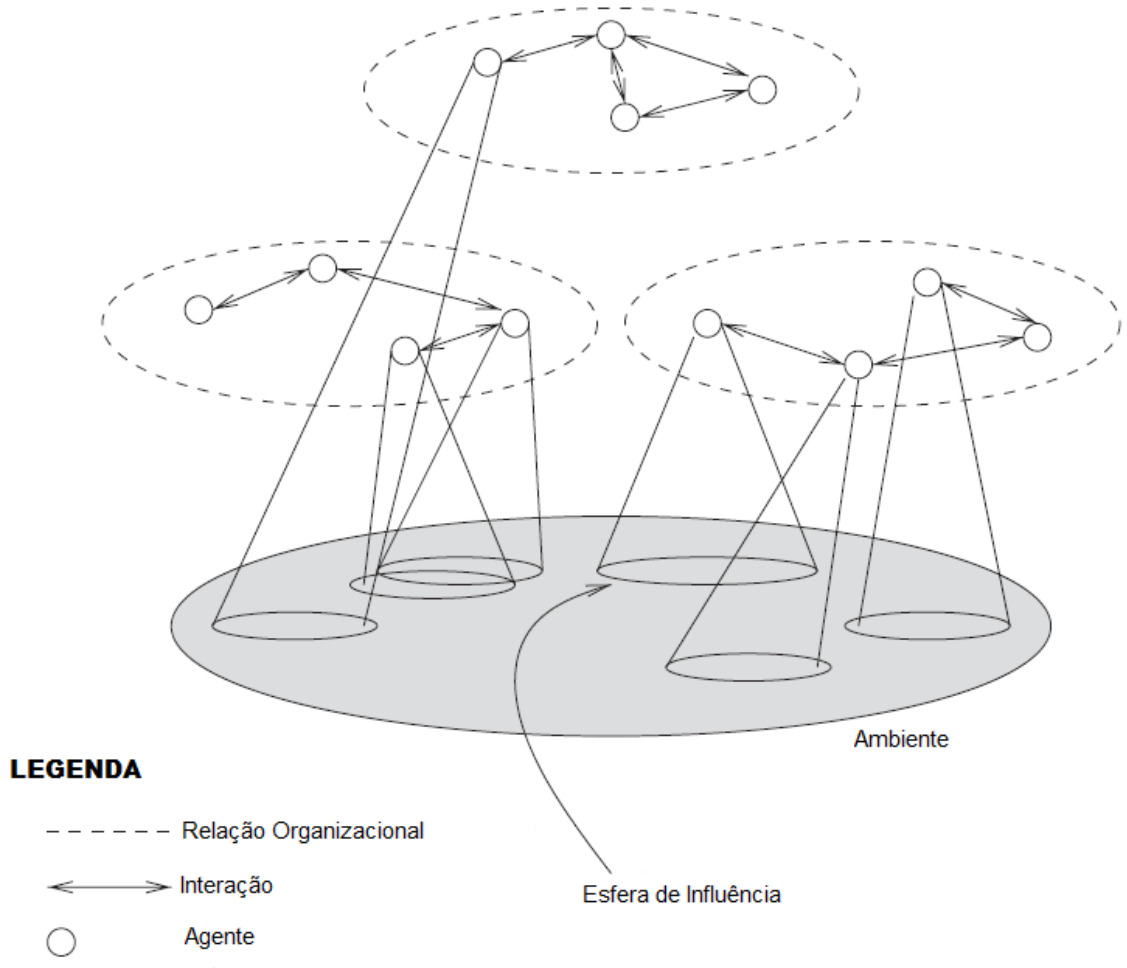
Os SMA podem incluir diversos agentes que interagem entre si, onde cada agente é um elemento capaz de resolução autônoma de problemas e opera assincronamente em relação aos demais agentes. Para que um agente possa agir como parte do sistema, é necessário a existência de uma estrutura que permita a comunicação e/ou interação entre os agentes que compõe o SMA.

SMA abordam atividades de um conjunto de agentes autônomos. O foco das pesquisas residem nos modelos para conceber agentes, suas organizações e interações de modo genérico. Isso para que o modelo genérico de SMA desenvolvido possa ser instalado em um caso particular semelhante quando determinada necessidade surgir (REZENDE, 2003), por exemplo um SMA para o leilão de vagas pode ser adaptado para outros leilões.

Diversas metodologias de coordenação dos SMA foram propostas, dividindo-se em dois grupos de agentes: competitivos e cooperativos. Os agentes competitivos estão preocupados com seu próprio bem e negociam com outros agentes com o intuito de atingir seu próprio objetivo. O oposto ocorre com agentes cooperativos, onde os agentes cooperam entre si a fim de melhorar o bem-estar coletivo (FERBER, 1999).

A Figura 4 apresenta a estrutura de um SMA, mostrando as organizações de agentes (conjuntos de agentes presentes no sistema), e a esfera de influência (parte do ambiente a qual o agente será capaz de modificar).

Figura 4 – Estrutura de um SMA



Fonte: Adaptado de Bordini et al. (2007)

Um SMA normalmente é composto de múltiplos agentes, cada um com diferentes capacidades de percepção e ação a respeito do ambiente. Cada agente terá a sua esfera de influência, sendo assim capaz de influenciar diferentes aspectos do ambiente (JENNINGS, 2000).

2.4 UTILIZAÇÃO DE SMA EM ESTACIONAMENTO INTELIGENTE

Em (DI NAPOLI et al., 2014) é apresentado um SMA que utiliza negociação entre agentes na organização de um S*mart Parking*. A negociação nesse trabalho segue uma adaptação do *Contract Net Protocol* onde os agentes usuários solicitam uma vaga com os atributos que a vaga precisa atender (como preço e localização do estacionamento), e o agente gerenciador observa as vagas disponíveis nos estacionamentos cadastrados. Em seguida o gerenciador faz rodadas de propostas, oferecendo uma vaga a cada rodada, até o acordo ser fechado ou até que o limite de propostas chegue ao fim.

2.5 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentados os principais conceitos acerca de Agentes Inteligentes, Sistemas Multiagentes, os relacionamentos entre os agentes e os tipos de agentes. Por fim, foi apresentado uma utilização de um SMA em um estacionamento inteligente.

No próximo capítulo será apresentado o framework JaCaMo, utilizado para programação e modelagem de um SMA, o qual engloba as seguintes plataformas: Jason, para a programação dos agentes utilizando a arquitetura BDI; Cartago, utilizada para a modelagem dos artefatos e Moise, empregada na normatização do SMA.

3 FRAMEWORK JACAMO

A primeira versão implementada do projeto MAPS (CASTRO, 2015) foi desenvolvida utilizando o *framework* JaCaMo. O presente capítulo apresenta as definições existentes na literatura sobre o *framework.*

O JaCaMo (JACAMO, 2011) combina três plataformas para alcançar uma programação robusta de SMA, sendo elas:

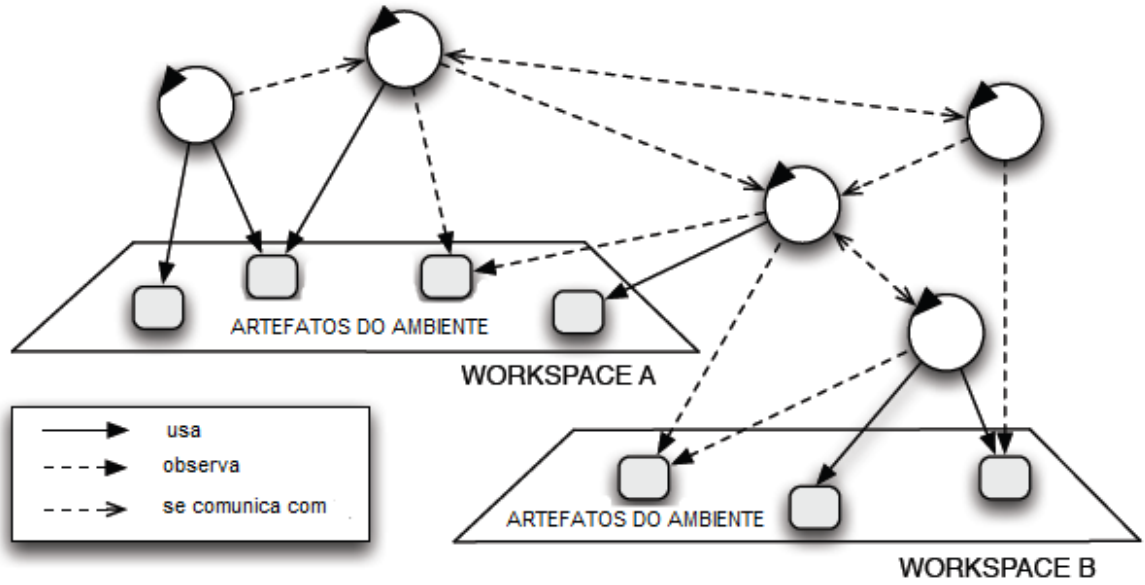
- Jason: Para programação de agentes;
- Cartago: Para a programação dos artefatos do ambiente;
- Moise: Para a organização normativa dos agentes.

Em um SMA implementado em JaCaMo a organização social dos agentes BDI é dada pelo Moise, os agentes são programados em Jason e eles trabalham em um ambiente compartilhado programado em Cartago.

Cada uma das ferramentas que compõem o *framework* JaCaMo possui o seu próprio conjunto de abstrações para a programação, bem como, seu próprio modelo e meta-modelo de programação. Portanto, para o *framework* considerou-se como peça fundamental a definição do modelo global de programação, tornando assim possível a integração de todas as abstrações disponíveis em cada plataforma.

Para a implementação dos mecanismos de negociação foram utilizados apenas os recursos das plataformas Jason e Cartago. Em Jason estão implementados os agentes, com seus planos, crenças e objetivos, enquanto o Cartago foi utilizado para o controle do tempo que os agentes *drivers* levam para receber o recurso. A interação destas duas plataformas é ilustrada na Figura 5.

Figura 5 – Interação Jason e Cartago



Fonte: Adaptado de (JACAMO, 2011).

Na sequencia deste capítulo são apresentadas com mais detalhes as plataformas do JaCaMo.

3.1 JASON

A linguagem de programação Jason, desenvolvida utilizando Java, é uma extensão da linguagem AgentSpeak, e usada para a programação de agentes BDI.

A seguir serão descritas as funções que a linguagem possui apresentadas em (BORDINI; HüBNER; WOOLDRIDGE, 2007), a partir de três categorias (crenças, objetivos e planos).

3.1.1 Crenças

A linguagem Jason possui uma base de crenças inicial, a qual é uma simples coleção de literais, do mesmo modo que uma programação lógica tradicional. Sendo assim, a informação é representada através de predicados, como por exemplo o predicado, *freeSpots(10)* é uma crença, possível ao *manager,* a qual indica que o estacionamento possui dez vagas livres. O agente possuir a crença que

existem dez vagas significa que ele acredita nisso, o que não é necessariamente verdade.

Além das crenças iniciais o agente recebe informações vindas de um agente no ambiente ou do próprio ambiente. A linguagem Jason possui um recurso de anotações, onde é possível identificar a origem de cada informação recebida.

A programação em Jason considera três tipos de fonte de informação:

- Informação perceptiva: O agente adquire certas crenças em consequência da observação e percepção do seu ambiente.
- Comunicação: Como os agentes se comunicam dentro do SMA, eles devem ser capazes de interpretar a informação recebida de outro agente. No comando "***.send(AGENT,tell,waitFor(TWT))***" o agente *manager* envia ao *driver "AGENT"* uma mensagem informando o tempo de espera, então o *driver* passa a ter a crença "*waitFor*(TWT)" onde ***TWT*** é o valor em milissegundos (*Total Waited Time*).
- Notas Mentais: Possui o objetivo de lembrar o agente sobre o passado e até mesmo um lembrete do que deve ser utilizado, exemplo a seleção de um plano.

3.1.2 Objetivos

Objetivos correspondem às características do ambiente que o agente almeja ser verdade, estando disposto a atuar sobre o ambiente para alterá-lo de forma que realmente seja verdade, sendo conhecido como objetivo de realização. Jason possui dois tipos de objetivos:

- Objetivo por realização: é definido pelo operador "!" como em *!printSpots*. Quando o agente tem esse objetivo, ele deverá executar um plano para imprimir na tela as vagas, então o objetivo será alcançado.
- Objetivo de teste: denotado por "?", como em *?credits(CDT)*, significa que o agente vai verificar em suas crenças e buscar pela informação do saldo atual de créditos que ele possui.

3.1.3 Planos

Um plano em Jason é composto em três partes: evento gatilho, o contexto e o corpo. O evento gatilho e o contexto são chamados de cabeça do plano. As três partes são sintaticamente separadas por ":" e "← ". Assim define-se um plano como:

eventoGatilho : contexto ← corpo

onde:

- Evento gatilho: informa para o agente as condições para que a escolha do plano seja realizada.
- Contexto: O contexto é um subconjunto de crenças, quando essas possuem o mesmo valor que as atuais crenças do agente o plano a que elas se referem é escolhido para a execução.
- Corpo: corresponde as ações do agente a serem executadas se o plano for escolhido.

O Código 1 apresenta o plano *+!leaveSpot,* executado pelo *driver* quando estiver saindo do *Smart Parking.*

Código 1 – Plano *requestSpot*

```
1 +!requestSpot[source(AG)] <-
2     .term2string(AG,AGENT);
3     .print("Agent: ",AGENT," has requested a spot!");
4     driverRequestedSpot(AG).
```

Fonte: Autoria Própria

Neste plano o evento gatilho é objetivo *!requestSpot* disparado a partir de uma requisição do *driver* “AG”, o plano está livre de contexto, e o corpo são as linhas 2, 3 e 4 onde respectivamente o plano executa uma função padrão do Jason que gera uma *string* com o nome do agente em “*AGENT*”, mostra na tela que o *driver* requisitou uma vaga, e por fim executa a operação *driverRequestSpot* do artefato *TimeControl* em Cartago que será explicado na Subseção 3.2.2.

3.2 CARTAGO

Common ARTifact infrastructure for AGents Open environments (Cartago) é uma plataforma de desenvolvimento, baseada no modelo de Agentes e Artefatos

para modelagem e *design* de SMA. A plataforma possibilita o desenvolvimento e execução de ambientes baseados em artefatos, estruturado em *workspaces* abertos, nas quais agentes de diferentes plataformas podem se unir para realizar tarefas. Com o Cartago, desenvolvedores de SMA possuem um modelo simples para projetar e programar o ambiente computacional do agente, composto por conjuntos dinâmicos de artefatos (CARTAGO, 2006).

3.2.1 Workspaces

Um ambiente é composto por um ou mais *workspaces*, possivelmente distribuídos em uma rede. Agentes implementados em Jason podem participar de um ou mais *workspaces* simultaneamente, sendo inicialmente designado a um *workspace* padrão. O agente deve obrigatoriamente pertencer a no mínimo um workspace para usufruir de um ambiente.

3.2.2 Repertório de Ações e Artefatos

Conjunto de ações definidas pelos artefatos do ambiente, sendo mapeamento um-para-um entre ações e operações, implicando também que se uma operação tem sucesso ou falha sua ação possuirá o resultado respectivo. Um artefato é definido como sendo uma entidade não-autônoma dentro de um ambiente, pois essa entidade é invocada apenas por agentes.

O Código 2 apresenta uma operação do artefato *TimeControl* invocada pelo agente *manager*, utilizada para registrar o momento em que um *driver* faz a requisição por uma vaga. Nela, o identificador do *driver* é passado por parâmetro é o mesmo é adicionado a um *arrayList* junto com o seu horário de chegada.

Código 2 – Operação driverRequestSpot do artefato TimeControl

```
1 @OPERATION
2 public void driverRequestedSpot(Object idDriver){
3     driver = new Driver(idDriver.toString(), new Date());
4     timeControl.add(driver);
5 }
```

Fonte: Autoria Própria

3.3 MOISE

Moise é um modelo organizacional para sistemas multiagentes baseado em noções como papéis, grupos e missões. Ele permite que um SMA tenha uma especificação explícita de sua organização. Esta especificação deve ser usada tanto pelos agentes para argumentar sobre sua organização e por uma plataforma de organização que impõe que os agentes sigam a especificação (MOISE, 2006).

3.4 CONSIDERAÇÕES FINAIS

Neste capítulo foi apresentado o *framework* utilizado para a implementação do MAPS e do MAPS-*Auctions*, o JaCaMo, sendo ele a união de três plataformas que possibilitam implementar de maneira robusta um SMA. O desenvolvimento do MAPS-*Auctions* utilizando o JaCaMo está apresentado no Capítulo 6.

O próximo capítulo apresentará a negociação entre agentes. Negociação que pode ser implementada utilizando a plataforma Jason do JaCaMo. Esta plataforma é responsável pela programação das estratégias do agente e das funções de comunicação entre os agentes.

4 NEGOCIAÇÃO ENTRE AGENTES

Agentes autônomos interagem com outros agentes almejando alcançar os seus objetivos. Segundo (FARATIN, 1998) essas interações têm como objetivo persuadir o outro agente a:

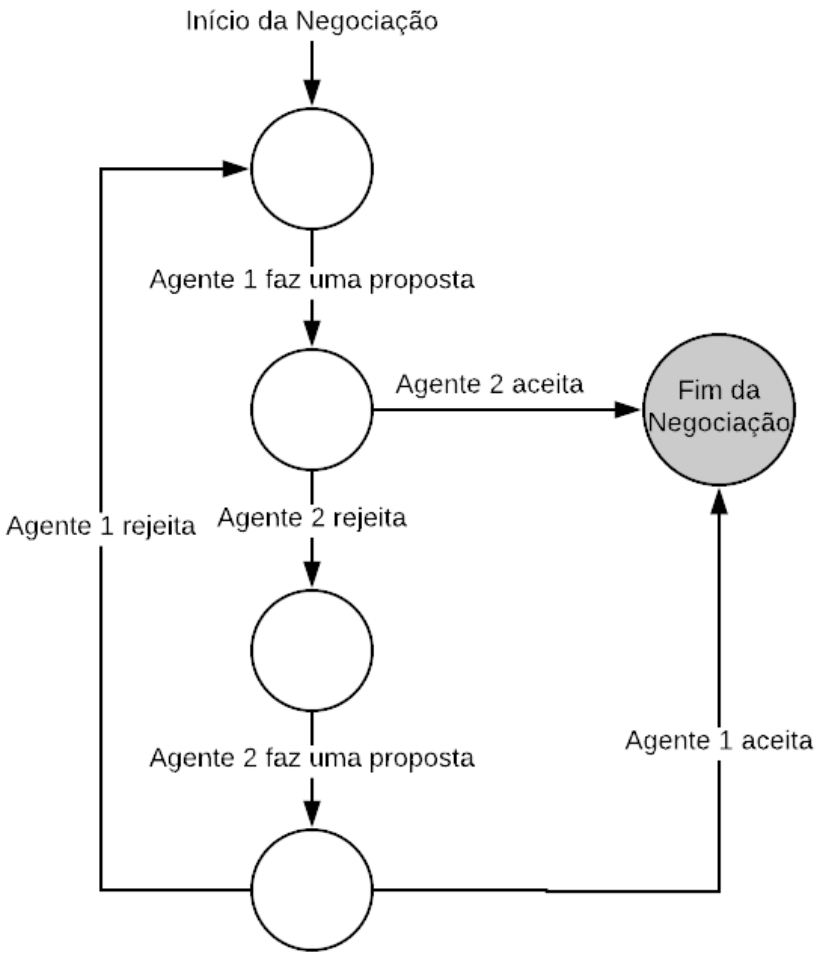
- Executar uma determinada ação;
- Modificar o seu plano de ação;
- Chegar a um comum acordo em uma linha de ação.

Buscando essa persuasão é necessário que ocorra a negociação, troca de mensagens entre os agentes. Segundo Michael Wooldridge (2009) existem quatro componentes presentes em uma negociação:

- O conjunto de negociação que representa o conjunto de propostas possíveis que o agente pode fazer;
- O protocolo onde estão reguladas quais são as propostas que o agente tem permissão para realizar;
- O conjunto de estratégias, onde cada agente possui a sua, a qual determina a proposta que o agente fará;
- A regra que determina quando o acordo é fechado e quais são os termos desse acordo.

Geralmente a negociação ocorrerá com os agentes trocando propostas em uma série de rodadas, até chegarem a um acordo definido na regra de negociação ou interromperem a negociação. As propostas que o agente faz são definidas por sua estratégia, as quais devem estar presentes no conjunto de negociação, e devem estar conforme o protocolo (WOOLDRIDGE, 2009). O autômato ilustrado na Figura 6 representa essas rodadas da negociação.

Figura 6 – Autômato demonstrativo das propostas em uma negociação



Fonte: Adaptado de Wooldridge (2009)

Conforme apresentado por O'Hare e Jennings (1996), são três os principais fatores presentes em uma negociação:

- A comunicação, responsável pela linguagem utilizada para a interação dos agentes, pela semântica utilizada, pelo protocolo que a negociação deve seguir e pela estrutura da negociação;
- A decisão, modelando o raciocínio dos agentes durante a negociação. Algoritmos para definir as estratégias que o agente emprega na negociação a fim de alcançar suas necessidades; e

- O processo, responsável pela visão global da negociação, analisando como um todo o procedimento e o comportamento dos agentes envolvidos.

Os fatores que tornam complexas as negociações entre os agentes são os múltiplos atributos, e a quantidade de agentes envolvidos, pois o crescimento desses fatores gera um problema exponencial (WOOLDRIDGE, 2009).

Um exemplo de negociação com apenas um atributo se dá quando dois agentes estão negociando apenas sobre o preço 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 suba, e o agente comprador quer o menor preço possível.

Um cenário de negociação de múltiplos atributos é a compra de um imóvel, não sendo apenas o preço o atributo de negociação da compra. O preço até pode ser o principal, mas a localização, vizinhança, garagem, distribuição dos cômodos, entre outros diversos atributos também são analisados. Esse tipo de combinação 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 (WOOLDRIDGE, 2009).

Com relação a quantidade de agentes envolvidos, o processo pode ocorrer de três maneiras:

- Um para um: um agente negocia apenas com outro agente. Como acontece na negociação do MAPS (CASTRO, 2015) entre o agente *manager* e o agente *driver* para alocação de vaga.
- Muitos para um: um agente negocia com mais de um agente ao mesmo tempo, como em um leilão.
- Muitos para muitos: neste caso, diversos agentes negociam com diversos agentes ao mesmo tempo. No pior caso, com “n” agentes envolvidos, geraria um negociação com até n(n-1)/2 *threads* de negociação, tornando difícil de controlar.

Na literatura (FARATIN, 1998), (FIPA, 2002) (WOOLDRIDGE, 2009). é possível encontrar mecanismos de negociação em SMA. Nas subseções a seguir serão apresentados mecanismos de leilão e o *Contract Net Protocol*.

4.1 LEILÕES EM SMA

Leilões consistem em uma estrutura de negociação que permita a um vendedor (leiloeiro) ofertar um ou mais itens diferentes para os participantes interessados. Neste mecanismo de negociação leiloeiros e participantes negociam competitivamente utilizando um conjunto de regras que auxiliam os envolvidos a obterem o maior benefício próprio proveniente da negociação (BANASZEWSKI, 2014)**.**

Um leilão em SMA é uma negociação entre um agente leiloeiro e uma coleção de agentes licitantes (Muitos para um), onde o objetivo do leiloeiro é alocar certo recurso para um licitante apenas. Na maioria dos casos o desejo do agente leiloeiro é obter o maior preço possível enquanto o agente licitante deseja o menor preço. O leiloeiro buscará alcançar seu desejo através do projeto apropriado de leilão, enquanto o licitante tentará alcançar o menor preço por meio do uso de uma estratégia de lances eficaz (WOOLDRIGE, 2009).

Existem diversos tipos de leilões, e esses exibem diferentes características, como:

- Negociação Unilateral: Os licitadores são uniformemente do tipo comprador, havendo um só vendedor que aceita as licitações de múltiplos compradores. Ou os licitadores são uniformemente do tipo vendedor, havendo um só comprador que aceita as licitações de múltiplos vendedores;
- Negociação Bilateral: Admitem múltiplos compradores e vendedores ao mesmo tempo;
- Negociação de Item Único: Apenas a alocação de um recurso é negociada;
- Negociação de Múltiplos Itens: São negociados diferentes recursos ao mesmo tempo.

No desenvolvimento deste trabalho serão apresentados apenas leilões unilaterais e de item único, pois estão definidos os agentes *drivers* como participantes e o *manager* como leiloeiro, e o único item a ser leiloado serão as vagas do *Smart Parking*.

As subseções a seguir trazem modelos de leilões apresentados por Wooldridge (2009), sendo eles o leilão Inglês, o Holandês e o FPSB (*First-Price Sealed-Bid*).

4.1.1 Leilão Inglês

É o modelo mais comum de leilão, onde todos os agentes no sistema podem ver os lances feitos, e os lances acontecem de maneira crescente. Funciona da seguinte forma:

1. O agente leiloeiro (vendedor) inicia a negociação definindo um preço inicial para a alocação do recurso.
2. Os agentes licitantes dão lances, que devem necessariamente ter um valor maior do que o maior valor oferecido até então.
3. Quando não há mais agentes dispostos a aumentar a oferta, ou o tempo de negociação chega ao fim, o recurso é alocado ao agente que deu o maior lance.
4. Por fim o agente ganhador deve pagar o valor definido em seu lance para obter o recurso.

Uma estratégia para a forma como os agentes licitantes fazem sua oferta pode ser definindo que eles deem lances com uma porcentagem definida maiores em comparação com o maior lance, até chegar ao valor máximo que está disposto a pagar pelo recurso. Ultrapassando esse valor, o agente desiste da aquisição do recurso.

4.1.2 Leilão Holandês

É um modelo de leilão aberto, onde todos os participantes podem ter acesso aos lances, e seus valores são descendentes. Funciona na desta maneira:

1. O agente leiloeiro inicia a negociação, ofertando o recurso a um preço elevado, acima do valor máximo.
2. O leiloeiro abaixa continuamente o preço do recurso, até que algum licitante aceite o valor atual.

3. O recurso é alocado ao licitante que aceitou arcar com o valor atual do leilão.

4.1.3 Leilão First-Price Sealed-Bid (FPSB)

O FPSB é um leilão fechado, onde apenas o leiloeiro conhece os lances ofertados, e os compradores submetem uma única proposta. Não há mais iterações, sendo o recurso alocado ao licitante que fez a proposta de maior valor pelo recurso, mediante ao pagamento do valor proposto.

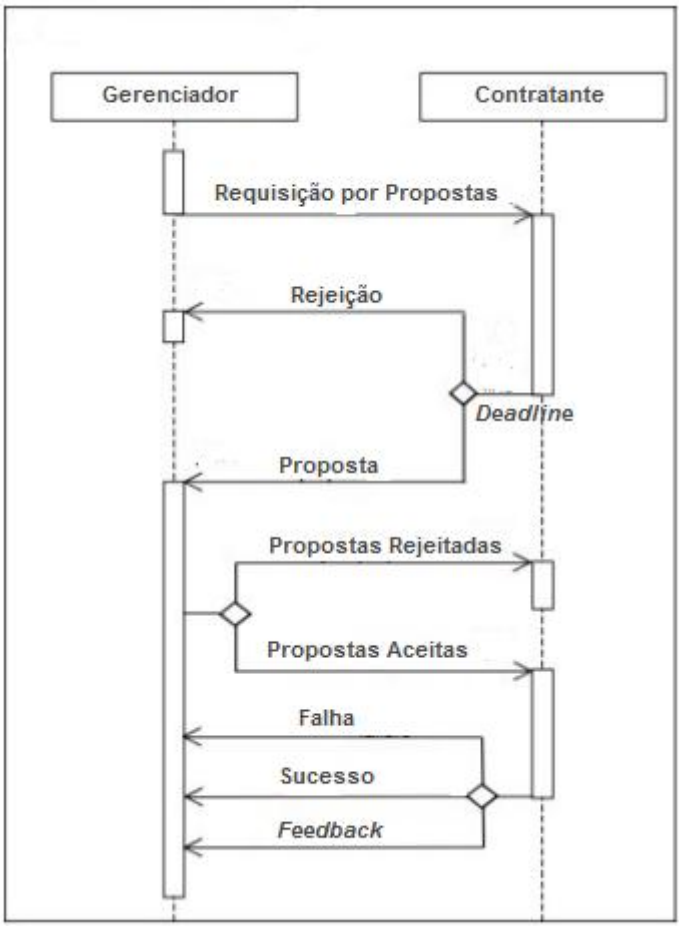
Esse modelo de leilão é usado no *Contract Net Protocol*, apresentado na Subseção 4.2.

4.2 *CONTRACT NET PROTOCOL*

Proposto por Smith (1980) com o objetivo de alocar recursos e como mecanismo de negociação para sistemas distribuídos e padronizado pela *Foundation for Intelligent Physical Agents* (FIPA) em 2002, adaptado à SMA, o *Contract Net* funciona como protocolo de licitação.

A Figura 7 apresenta um diagrama de sequência que ilusta a negociação nesse mecanismo. A negociação é iniciada quanto um agente gerenciador dispara uma requisição para outros agentes buscando propostas para alocar algum recurso, especificando as condições para o negócio. 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, negativa ou confirmando o acordo. Ao fim da negociação o contratante envia uma mensagem ao gerenciador que pode informar o sucesso, trazer um *feedback* sobre, ou informar falha no negócio.

Figura 7 – FIPA- Contract Net Protocol



Fonte: Adaptado de FIPA (2002).

O *Contract Net* é uma negociação de “Muitos para um”, que permite encontrar o agente mais adequado para determinado recurso ou tarefa, comparando propostas apresentadas pelos agentes.

4.3 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentados os conceitos das negociações entre agentes e como funcionam alguns mecanismos de negociação presentes na literatura.

O próximo capítulo mostra as principais mudanças do MAPS para o MAPS-*Auctions*. Estas mudanças possibilitaram a implementação e análise dos mecanismos de negociação apresentados.

5 DIFERENÇAS ENTRE O MAPS E O MAPS-*AUCTIONS*

Este capítulo apresentará com mais detalhes como funciona o projeto MAPS implementado em (CASTRO, 2015), e em seguida as diferenças que o presente trabalho propõe com relação à versão inicial do MAPS.

5.1 ARQUITETURA DO PROJETO MAPS

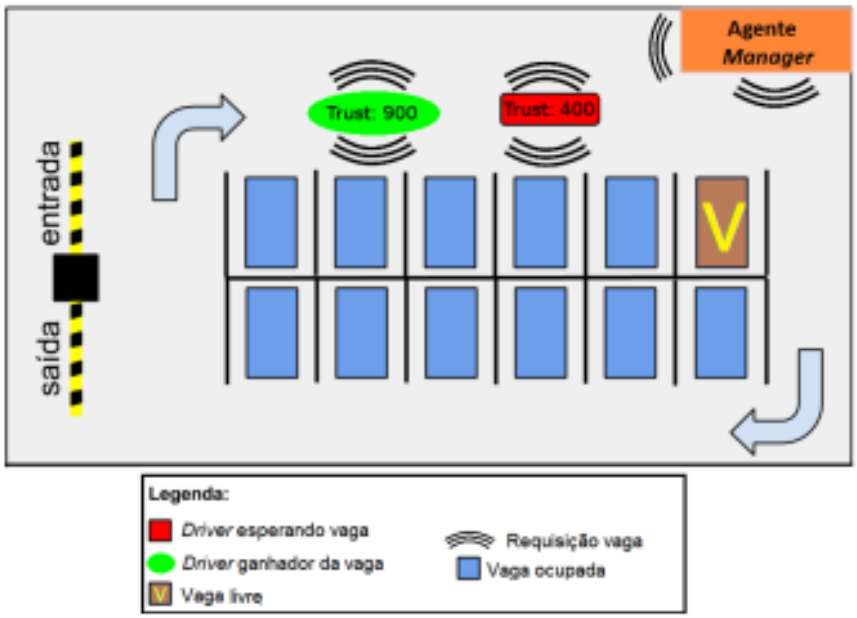
Na versão inicial do projeto MAPS (CASTRO, 2015), desenvolvida utilizando o *framework* JaCaMo tem-se dois tipos de agentes: agentes *drivers* e o agente *manager*. Os *drivers* fazem requisições por vagas, informando seu nome e seu valor de confiança, e aguardam até receberem a vaga. Quando a recebem executam os planos para estacionar e depois para deixar a vaga.

Já o *manager* tem a responsabilidade de gerenciar todo o funcionamento do estacionamento e atribuir as vagas aos agentes *drivers*. Se o *Smart* Parking possuir vagas disponíveis, e o *manager* receber uma requisição de um *driver,* então atribuirá a vaga ao requisitante diretamente. Porém, se o estacionamento está com a capacidade de ocupação máxima e requisições são recebidas, os agentes que requisitaram vagas serão inseridos em uma fila de espera, e assim que liberar uma vaga o *manager* iniciará o processo de escolher qual *driver* da fila receberá a vaga.

Ele possui duas estratégias para alocar o recurso. Uma é pelo tempo de espera, se um *driver* está esperando além do limite de tempo, o mesmo é priorizado. Outra se dá pela utilização do valor de confiança do agente, o *driver* que estiver na fila e possuir o maior grau de confiança recebe a vaga que está sendo disputada.

A Figura 8 ilustra a arquitetura para a alocação de vagas do projeto MAPS:

Figura 8 – Arquitetura do MAPS



Fonte: Castro, et al. (2016)

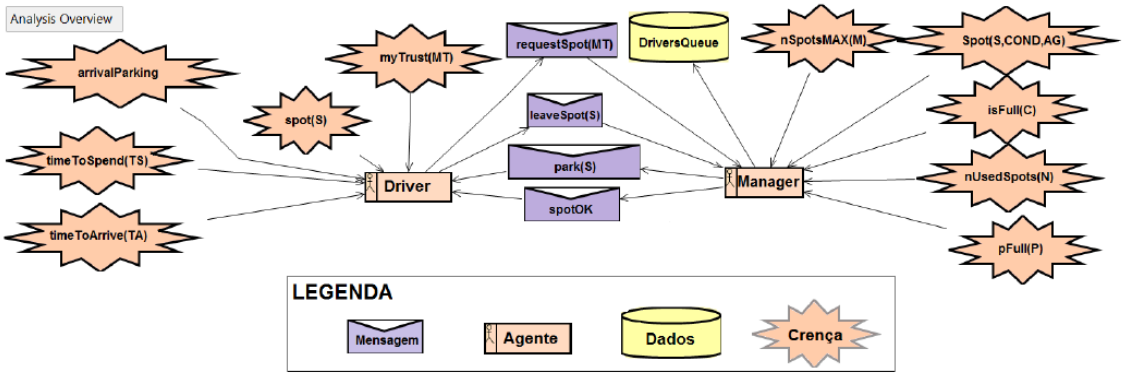
5.2 DIFERENÇAS ENTRE O MAPS *AUCTIONS* E O MAPS

Nesta subseção apresenta-se os diagramas de visão geral do MAPS e do MAPS *Contract Net* a fim de comparação e de apresentar o que é acrescentado neste trabalho.

A Figura 9 apresenta a visão geral utilizando a metodologia *Prometheus*, que ilustra a interação entre os agentes e suas crenças no trabalho desenvolvido em (CASTRO, 2015). São cabíveis aos *drivers* crer sobre: o tempo que ele levará para chegar ao estacionamento; o tempo que ficará estacionado; se ele chegou ao estacionamento; a vaga que ele está e o seu valor de confiança. As crenças do *manager:* número de vagas; status de cada vaga; conhecimento a respeito do estacionamento estar cheio; o número de vagas usadas e a porcentagem de vagas ocupadas.

A interação entre os agentes acontece do *driver* para o *manager*, quando o *driver* quer requisitar uma vaga ou avisar que está saindo do estacionamento. Já na direção contrária acontece quando o *manager* autoriza o *driver* a estacionar em uma determinada vaga, e para avisar a existência de vagas disponíveis.

Figura 9 – Diagrama de visão geral MAPS Original



Fonte: Castro (2015)

Os mecanismos de negociação implementados neste trabalho, são a continuação do trabalho de (CASTRO, 2015). Mantem-se o formato das vagas como crenças do *manager*. Porém funções como o controle de entrada, saída, e requisição por vagas foram adaptadas para focar este trabalho na implementação dos modelos de negociação.

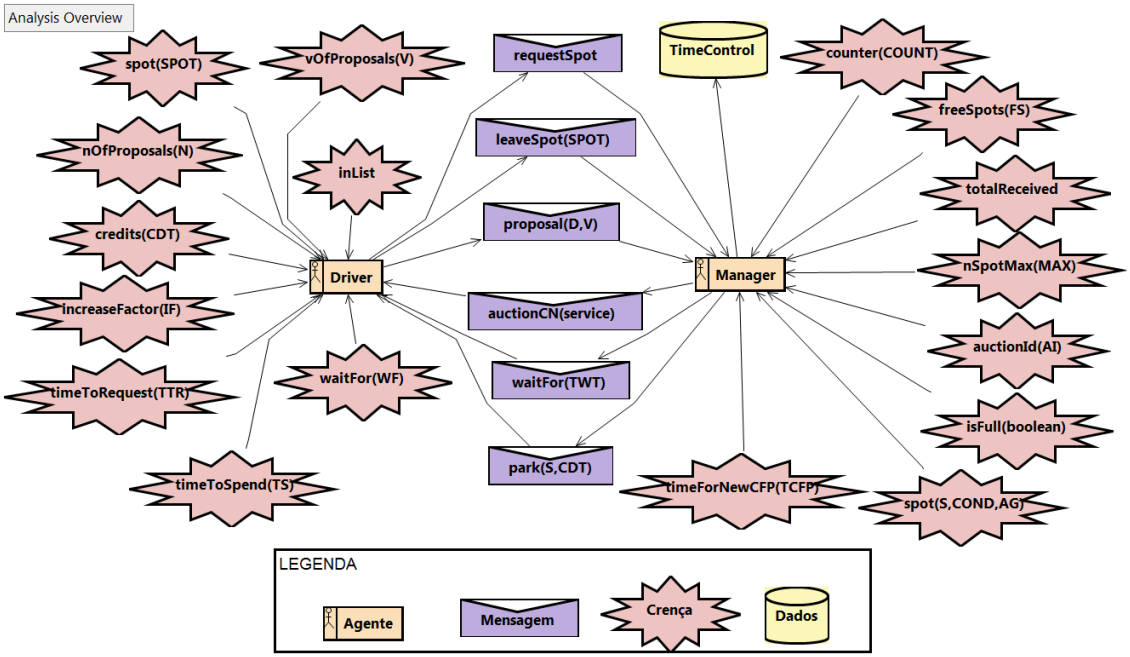
O trabalho MAPS-*Auctions*, não utiliza mais as crenças do valor de confiança e o número de vagas usadas, mas são utilizadas novas crenças que possibilitam realizar a negociação das vagas pelo novo modelo.

Os dados "*DriverQueue*", antes utilizados para decidir qual agente *Driver* receberia o recurso disponível, não são mais utilizados. As decisões de alocação do recurso são tomadas pelo *manager* utilizando funções do Jason, como será apresentado no Capítulo 6 onde será descrito detalhadamente a implementação de cada mecanismo de negociação.

Por fim, o uso de banco de dados e do Cartago no modelo desenvolvido neste trabalho será para o controle do tempo de espera dos *drivers.* Os detalhes do artefato Cartago serão apresentados na Seção 6.5.

A Figura 10 apresenta o diagrama de visão geral do MAPS *Contract Net* a fim de ilustrar as diferenças entre os trabalhos. Nele aparecem outras crenças do *driver* como: a quantidade de créditos que possui, o fator utilizado como estratégia para aumentar seus lances, o número de propostas enviadas, entre outras. O *manager* passa a crer a respeito das vagas livres, o quanto o *Smart Parking* recebeu de créditos, além de outras. Existem três tipos novos de mensagens: o *driver* envia propostas com o valor de créditos que quer pagar pela vaga; o *manager* anuncia o início da disputa por vagas; e o *manager* informa ao *driver* o tempo esperado.

Figura 10 – Diagrama de visão geral MAPS *Contract Net*



Fonte: Autoria Própria

Neste capítulo foi introduzido o MAPS *Contract Net* apenas para compará-lo com o MAPS (CASTRO,2015). O próximo capítulo apresentará detalhadamente a implementação de cada um dos três mecanismos de negociação do MAPS-*Auctions*.

5.3 CONSIDERAÇÕES FINAIS

O objetivo desse capítulo foi apresentar o MAPS e ilustrar as mudanças realizadas para o desenvolvimento deste trabalho. O MAPS-*Auctions* é uma extensão do MAPS.

No Capítulo 6 é apresentado com detalhes como funciona os seguintes mecanismos de negociação, implementados no MAPS-*Auctions*: *Contract Net*, leilão inglês e leilão holandês.

6 DESENVOLVIMENTO DO MAPS-AUCTIONS

Com o objetivo de auxiliar a solucionar alguns problemas do trânsito nas grandes cidades, o projeto MAPS trabalha com o desenvolvimento de um *smart parking* utilizando sistema multiagente. Como colaboração a este projeto do GPAS, foram implementados três mecanismos de negociação a fim de viabilizar a análise de eventuais vantagens e desvantagens deles aplicados ao contexto do MAPS e a alocação de recursos em SMA.

No Subseção 6.1 será apresentado o funcionamento geral do MAPS-*Auctions*, e as funções comuns aos três modelos de negociação como funções de controle do estacionamento para o *manager* e de requisitar, estacionar e sair executadas pelo *driver.* As subseções 6.2, 6.3 e 6.4 apresentarão com detalhes como funcionam as estratégias dos *drivers* e do *manager* em cada negociação. Por fim a Seção 6.5 apresenta o artefato *TimeControl* desenvolvido na plataforma Cartago.

6.1 FUNCIONAMENTO MAPS AUCTIONS

No desenvolvimento deste trabalho o gerenciamento das vagas será realizado por um agente centralizador, denominado gerente (*manager*) do estacionamento. Este agente é responsável por controlar a entrada e saída dos motoristas, oferecer as vagas e aceitar ou não as ofertas recebidas para então alocar o recurso (vaga) aos motoristas. Os outros agentes do sistema são os motoristas (*drivers*), que utilizam e interagem com o *manager* solicitando, pagando e utilizando as vagas do estacionamento.

As Subseções 6.1.1 e 6.1.2 explicam as funções genéricas dos *drivers* e do *manager*, respectivamente, para todos os mecanismos de negociação desenvolvidos.

6.1.1 Agente Driver

O objetivo de cada agente *driver* é o de competir com outros agentes a fim de obter o recurso (a vaga) em menos tempo. Ele inicia sua participação fazendo a

requisição por uma vaga, e depois de feita ele passa a crer que está em uma lista de espera. Quando ele estiver na lista ele estará apto a participar das negociações. Um dos parâmetros analisados neste trabalho é o tempo que leva para os *drivers* receberem as vagas, sendo assim os *drivers* participarão das negociações até adquirirem o recurso.

Depois de receberem o recurso, serão executados os planos para que o agente estacione na vaga determinada, ocupe a vaga por determinado tempo e em seguida execute o plano para deixar a vaga.

Como o foco deste trabalho é analisar a negociação das vagas, os *drivers* utilizados nos testes realizarão a requisição por vagas ao mesmo tempo, e utilizarão a vagas também por tempo igual.

As funcionalidades dos *drivers* comuns em todos os mecanismos de negociação apresentados são:

1. **Requisitar vaga**: Como plano inicial, o *driver* envia uma mensagem ao *manager* solicitando o recurso e passa a crer que está na lista de espera;
2. **Participar da negociação das vagas**: Sabendo que está na lista de espera por uma vaga, o *driver* é informado pelo *manager* que há vagas disponíveis e, se sua estratégia permitir, responde com uma oferta pela vaga;
3. **Receber uma vaga, estacionar e pagar**: Recebe do *manager* uma mensagem contendo a vaga para estacionar e o valor que deverá ser pago. Com essas informações ele passa a crer que está na vaga determinada e o valor a ser pago é descontado do total de créditos que possui;
4. **Deixar a vaga**: Depois de ocupar a vaga pelo tempo pré-determinado, o agente envia uma mensagem ao *manager* informando que está saindo da vaga.

Nos três modelos de negociação os *drivers* estão separados em três perfis, os quais serão explicados na Seção 7.1 e possuem diferentes estratégias a fim de alcançar o objetivo de obter uma vaga

Para os testes que serão apresentados no Cenário 3 das simulações, os créditos dos agentes serão definidos de maneira aleatória dentro do intervalo [1, 100].

6.1.1.1 Implementação do *driver* em Jason

Esta subseção apresenta os códigos utilizados para desenvolver as crenças, os objetivos e os planos genéricos para o funcionamento do *driver* no MAPS-*Auctions*.

O agente *driver* possui crenças e objetivos iniciais para começar a interagir no SMA, como apresentado no Código 3. Este exemplo expõe as seguintes crenças e objetivo respectivamente:

- Tempo até requisitar a vaga (*timeToRequest*): Valor inteiro correspondente a um segundo;
- Tempo que pretende ficar no estacionamento (*timeToSpend*): Valor inteiro correspondente a um segundo;
- Créditos iniciais (“*credits”*): Valor inteiro correspondendo a 100 créditos;
- Disposição a pagar pela vaga (“*willingnessToPay”*): Fator de multiplicação utilizado para determinar a porcentagem que o *driver* está disposto a pagar por uma vaga a partir dos créditos possui.

Código 3 – Crenças e objetivos iniciais do agente driver

```
/*Initials Beliefs*/
timeToRequest(6000).
timeToSpend(6000).
credits(100).
willingnessToPay(1).


/*Initials Goals*/
!requestParking.
```

Fonte: Autoria Própria

A estrutura de um plano é composta por três elementos: evento gatilho, contexto e corpo de ações. A partir do objetivo inicial, o agente passa a executar planos para que seus objetivos, que surgem durante a execução do programa, sejam alcançados. O *driver* possui três planos para alcançar o objetivo inicial, o objetivo de estacionar e o de deixar o *Smart Parking.* Esses planos serão apresentados a seguir:

1. ***requestParking***: Este é o plano executado a partir do objetivo inicial do agente *driver*, o qual requisita uma vaga do *Smart Parking*. Conforme apresentado no Código 4, para que o plano seja executado, o contexto é

que o *driver* possua a crença "*timeToRequest(TTR)*", onde *"TTR"* é valor inteiro em milissegundos que representa o tempo que levará ao *driver* requisitar a vaga depois de iniciado o plano. Para o agente aguardar este tempo, na segunda linha foi utilizada uma função do Jason que quando executada faz o agente esperar pelo tempo determinado até executar o próximo comando. Depois de o *driver* aguardar o tempo determinado, ele executa a linha 3, uma função do Jason de envio de mensagens, na qual o *driver* envia uma mensagem ao *manager* solicitando o recurso que fará o *manager* executar o plano "*!driverRequestedSpot"*. Por fim na última linha do plano o agente depois de enviar o requerimento passa a crer que está na lista de espera por vagas quando adiciona a crença "*inList"* a sua base de crenças.

Código 4 – Plano requestParking

```
1 +!requestParking : timeToRequest(TTR)<-
2         .wait(TTR);
3         .send(manager,achieve,driverRequestedSpot);
4         +inList.
```

Fonte: Autoria própria

2. ***park*:** Conforme apresentado no Código 5, o plano é executado pelo *driver* quando o *manager* envia ao *driver* uma mensagem para ele estacionar enviando como parâmetros a vaga ("*SPOT")* em que o *driver* deve estacionar e os créditos ("*CDT")* que serão pagos pelo recurso. Para o plano ser executado o *driver* deve crer que está na lista de espera (*inList)*, possuir a crença de quanto tempo ficará no estacionamento ("*timeToSpend(TS"),* onde "*TS"* é um valor em milissengundos), e por fim possuir a crença "*credits(PREVCDT)"*, o valor "*PREVCDT"* é o valor de créditos que o *driver* possui antes de pagar pela vaga e estacionar. O corpo de execução começa na linha quatro onde a crença que o *driver* está na lista de espera é excluída "*(-inList)"*, na sequência o agente utiliza uma função do Jason para imprimir no console que está estacionando na vaga *SPOT,* na linha 7 é adicionada a crença *spot* ao *driver* com o identificador da vaga que ele se encontra. Nas linhas 8 e 9 o *driver* mostra o valor pago pela vaga, e depois é atualizada a crença "*credits()"* subtraindo os créditos que o *driver* possuía pelos créditos pagos pela

vaga. Na linha 10 a função *".wait(TS)"* faz com que o *driver* permaneça pelo tempo *TS* estacionado, e quando a execução dessa função Jason chega ao fim, o *driver* passa a linha 11, onde passa a ter objetivo para deixar o estacionamento *"!leaveSpot"*.

Código 5 – Plano park

```
1 +!park(SPOT,CDT)[source(AGENT)] : inList &
2                                   timeToSpend(TS) &
3                                   credits(PREVCDT)
4     <-
5         -inList;
6         .print("Parking at the spot: ",SPOT);
7         +spot(SPOT);
8         .print("I paid ", CDT, "credits for the spot.");
9         -credits(PREVCDT); +credits(PREVCDT-CDT);
10        .wait(TS);
11        !leaveSpot.
```

Fonte: Autoria Própria

3. ***leaveSpot***: Plano executado quando o *driver* possui o objetivo de deixar o estacionamento. Conforme será apresentado no Código 6, o agente imprime a mensagem de saída, envia para o *manager* uma mensagem informando que está deixando o estacionamento e em qual vaga estava e por fim ele retira a crença de estar na vaga.

Código 6 – Plano leaveSpot

```
1 +!leaveSpot : spot(SPOT) <-
2     .print("Leaving the parking...");
3     .send(manager,achieve,driverLeftSpot(SPOT));
4     -spot(SPOT).
```

Fonte: Autoria Própria

Todos esses são planos genéricos à todas as implementações para que os *drivers* solicitem o recurso, ocupem e liberem a vaga. Os planos específicos de cada mecanismo de negociação serão apresentados nas Suseções 6.2, 6.3 e 6.4.

6.1.2 Agente M*anager*

O agente *manager* é o agente centralizador do SMA, responsável por controlar o status do *Smart Parking* e fazer a comunicação com os *drivers* para receber solicitações, negociar as vagas, e alocar o recurso. Existe apenas um *manager* no sistema.

Os *drivers* possuem as seguintes funcionalidades e responsabilidades dentro do sistema:

1. **Perceber requisições dos *drivers* por vagas:** A qualquer momento da execução do programa, o agente *manager* está apto a receber requisições provindas dos *drivers* por vagas;
2. **Gerenciar a negociação das vagas**: O agente gerencia a negociação das vagas, recebendo e enviando propostas com os valores e disponibilidade;
3. **Conhecer e controlar o estacionamento**: Controla as informações de todas as vagas, como qual agente está em cada uma ou se ela está livre. Controla as vagas específicas que serão atribuídas a cada *driver* (considera-se que os *drivers* sempre estacionaram na vaga correta). Está continuamente atento as saídas dos agentes *drivers*, e conhece a porcentagem da lotação atual do estacionamento;
4. **Calcular os ganhos**: A cada nova alocação de vaga o *manager* atualiza o valor total recebido como pagamento pelas vagas até o momento;
5. **Controlar o tempo de espera dos *drivers***: Conta o tempo que leva desde receber a requisição de um *driver*, até alocar a vaga a este *drivers.*

6.1.2.1 Implementação do Agente *Manager* em Jason

A partir das crenças do *manager* será realizado o controle do sistema. O Código 7, a seguir, apresenta um exemplo das crenças iniciais e do objetivo inicial do agente *Manager.*

Código 7 – Crenças e objetivos iniciais do agente manager

```
1 nSpotsMAX(2).
2 freeSpots(2).
3 isFull(false).
4 auctionId(0).
5 totalReceived(0).
6 timeForNewAuction(6000).
7 counter(1).
8 spot(0,0, "EMPTY").
9 spot(1,0, "EMPTY").
10
11 /* Initial goals */
12 !setupParking.
```

Fonte: Autoria Própria

No código estão apresentadas, respectivamente, as crenças:

- Número total de vagas do estacionamento (“*nSpotsMAX”*): valor inteiro correspondente a 3 vagas;
- Número de vagas disponíveis no momento (“*freeSpots”)*: Valor inteiro iniciado com o mesmo número do total de vagas do estacionamento;
- Crença booleana a respeito de o estacionamento estar com a lotação máxima (“*isFull”*);
- Crença com a quantidade de vezes que o *manager* executou o plano para ofertar vagas (“*auctionId”*): Valor inteiro identificador de cada negociação;
- Valor total recebido pelo estacionamento (“*totalReceived”*): Valor inteiro iniciado em 0;
- Valor inteiro que representa o tempo que o *manager* aguarda para iniciar uma nova negociação (“*TimeForNewAuction”*);
- A crença “*counter”* é um valor inteiro que funciona como uma variável auxiliar, utilizada para selecionar as propostas vencedoras em uma rodada de negociação;
- As vagas do estacionamento baseada em uma tupla composta por três valores: ID da vaga, estado da vaga e nome do agente que a ocupa.

O objetivo inicial *setupParking,* faz a chamada ao plano que inicia o controle do estacionamento inteligente. Este plano e todos os demais planos do agente *manager* serão apresentados a seguir:

1. ***setupParking***: A execução do plano *setupParking* cria uma instância do artefato Cartago *TimeControl*, apresentada em maiores detalhes na Subseção 6.5. Em seguida executa o plano “*startNegotiation”*. O código

do “*setupParking”* utilizado na implementação do *Contract Net Protocol*, o qual é semelhante nos outros mecanismos alterando apenas o nome do pacote do artefato Cartago e o nome do plano para iniciar a negociação, está apresentado no Código 8.

Código 8 – Plano setupParking

```
1 +!setupParking
2         <-
3
4          makeArtifact("t_Control", "mAPS_CartagoCNP.TimeControl", ["Starting Time Control"], ArtId);
5          focus(ArtId);
6
7          !startNegotiations.
```

Fonte: Autoria Própria

2. ***driverRequestedSpot***: Executado quando um *driver* envia uma mensagem ao *manager* solicitando uma vaga, quando recebida o *manager* registra o agente fonte da mensagem como “*AG”* em “*source(AG)”*. Na linha 2 do Código 9 é utilizada uma função do Jason para transformar o nome do *driver* em uma *string*, a linha 3 imprime o registro do requerimento. Por fim na linha 4 é executado uma operação do artefato *TimeControl* a qual inicia a contagem do tempo de espera do agente até receber a vaga. O Código 9, a seguir, apresenta o plano.

Código 9 – Plano driverRequestedSpot

```
1 +!driverRequestedSpot[source(AG)] <-
2     .term2string(AG,AGENT);
3     .print("Agent: ",AGENT," has requested a spot!");
4     driverRequestedSpot(AG).
```

Fonte: Autoria Própria

3. ***allocateSpot***:

   O plano apresentado no Código 10 é responsável pela alocação de vagas aos *drivers*, o qual recebe por parâmetro para execução o agente “*AGENT”* e o valor que será pago pela vaga “*CREDITS”*.

Código 10 – Plano allocateSpot

```
1 @allocateSpot[atomic]
2 +!allocateSpot(AGENT, CREDITS) :
3                     nSpotsMAX(MAX) &
4                     isFull(COND) &
5                     totalReceived(TR) &
6                     freeSpots(FS) &
7                     COND=false
8       <-
9           +~find;
10
11    for(spot(S,C,A)){
12        if(A = "EMPTY" & ~find & (COND = false)){
13            -spot(S,C,A); +spot(S,1,AGENT);
14            .print("Spot(",S,")has allocated for the agent ",AGENT," for ",CREDITS,"credits.");
15
16            -+totalReceived(TR+CREDITS);
17            .print("Total received untill now: ",TR+CREDITS);
18
19            totalWaitingTime(AGENT,TWT);
20            .send(AGENT,tell,waitFor(TWT));
21            .send(AGENT,achieve,park(S,CREDITS));
22
23            -freeSpots(FS);
24            +freeSpots(FS-1);
25
26            if((FS-1) = 0){
27                -isFull(COND);
28                +isFull(true);
29                .print("Parking lot FULL!");
30            };
31
32            .print("Parking available: ",((FS-1) * 100) / MAX,"%");
33
34            -~find;
35        }
36    };
37    +~find.
```

Fonte: Autoria Própria

Na linha 11 do Código 10 é iniciado um laço em busca de vagas desocupadas. Quando essa vaga for encontrada pela operação condicional da linha 12, o *manager* atualiza sua crença com ela ocupada na linha 13, e imprime qual vaga foi alocada a qual *driver* e por quantos créditos na linha 14. Na sequência incrementa a crença "*totalReceived(TR+CREDITS)"*, valor total recebido pelo *Smart Parking,* onde "*TR"* é o valor recebido até essa operação ser executada e "*CREDITS"* o valor pago pelo *driver* pela vaga em questão e imprime esse valor atualizado. Na linha 19 é executada uma operação do artefato *TimeControl* a qual retorna "*TWT",* o tempo esperado pelo *driver*. Na linha 20 o *manager* informa ao *driver* o tempo de espera. Na linha 21 o *manager* envia uma mensagem para o *driver* executar o plano "*park"* informando a vaga "*S*" e o valor pago. Em seguida são atualizadas as crenças do número de vagas livres ("*FS")* e se o estacionamento está

cheio. E por fim é impresso a porcentagem de vagas que permanecem disponíveis no estacionamento.

4. ***printSpots***: A execução do Código 11 faz com que o *manager* execute um laço do Jason que passa por todos as suas crenças *spot,* o qual todas as vagas, mostrando suas informações, onde "*V"* é o identificador da vaga, "*Z"* um valor booleano que representa se a vaga está ocupada ou livre, e por fim AG é o nome do agente *driver* que está ocupando a vaga ou "*EMPTY"* se a vaga estiver vazia.

Código 11 – Plano printSpots

```
1 +!printSpots <-
2     for(spot(V,Z,AG)){
3                 .print("Spot: ",V," - Condition: ",Z, " - Agent: ",AG);
4         }.
```

Fonte: Autoria Própria

5. ***driverLeftSpot:*** Conforme apresentado no Código 6, esse plano é executado quando um *driver* informa que está deixando a vaga. O Código 12 apresenta o plano "*driverLeftSpot" onde* a linha 4 executa uma operação do Jason que mata o *driver* no sistema. Na linha 6 o *manager* registra que o *driver* está deixando a vaga. Nas linhas 8 e 9 remove a crença que o agente estava na vaga "*S*" e registra que a vaga está vazia. Nas linhas 11 e 12 atualiza o valor "*FS*" da crença de vagas disponíveis. Na linha 14 o *manager* imprime a porcentagem do *smart parking* que está disponível. E por fim é atualizado a crença "*isFull()".*

Código 12 – Plano driverLeftSpot

```
@leaveSpot[atomic]
+!driverLeftSpot(SPOT)[source(AG)] : nSpotsMAX(MAX)
                                   & freeSpots(FS)
     <- .kill_agent(AG);
        .term2string(AG,AGENT);
        .print(AGENT," leaving the spot: ",SPOT);

        -spot(S,1,AG);
        +spot(S,0,"EMPTY");

        -freeSpots(FS);
        +freeSpots(FS+1);

        .print("Parking available: ",((FS+1) * 100) / MAX,"%");

        -isFull(COND);
        +isFull(false).
```

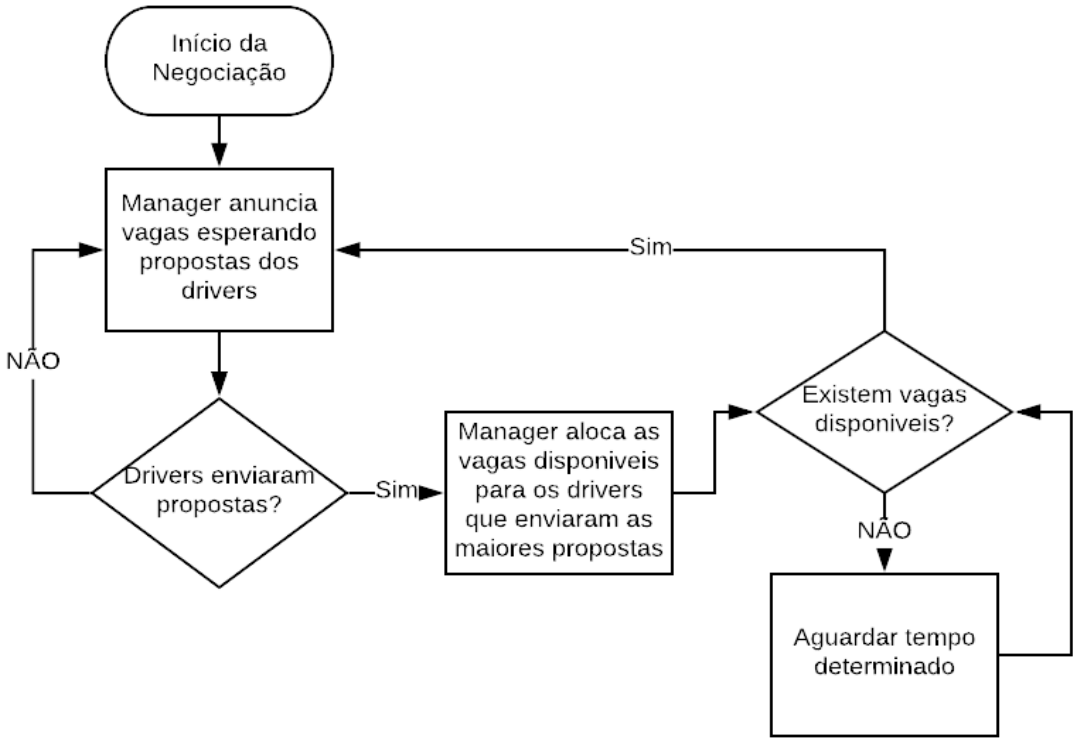
Fonte: Autoria Própria

Estes planos são comuns a todos os mecanismos de negociação. Nas subseções a seguir serão apresentados os planos executados para a negociação das vagas em cada abordagem.

6.2 DESENVOLVIMENTO MAPS CONTRACT NET

Conforme apresentado um dos modos de negociação realizados segue o protocolo *Contract Net* descrito na Seção 2 do Capítulo 4. A negociação das vagas nesse modelo, gerenciada pelo *manager*, é iniciada quando há vagas disponíveis. O primeiro passo é enviar uma mensagem aos *drivers* informando a disponibilidade de vagas e o requerimento propostas pelo recurso. Depois de aguardar um tempo determinado para receber essas propostas, o *manager* selecionará n *drivers* ganhadores em função de n vagas disponíveis no momento, e executará o plano para alocar a vaga a esses *drivers*. A Figura 11 apresenta o fluxograma desta negociação entre os agentes.

Figura 11 – Fluxograma da negociação no MAPS *Contract Net*



Fonte: Autoria Própria

Os *drivers* que possuem a crença (*inList*) de estarem na lista de espera por vagas, quando recebem do *manager* a mensagem de que o serviço para alocação de vagas foi aberto, enviam uma proposta iniciada com o valor mínimo randômico entre 1 e 2 créditos. O valor da proposta é incrementado a cada nova oferta, em função de um fator de aumento estipulado de acordo com o perfil do agente, até atingir o valor máximo de créditos que o agente está disposto a pagar pela vaga. Quando esse valor é atingido o *driver* repete o valor na proposta a cada novo serviço do *manager* até conseguir o recurso.

6.2.1 Implementação do MAPS *Contract Net* em Jason

A negociação é iniciada quando o plano "*startNegotiations"* é executado pelo *manager*, um plano executado recursivamente até o sistema ser desligado. A execução do plano, na linha 6 do Código 13, analisa se existem vagas livres ("*FS>0")*, se não houver ele volta ao início do plano executando a linha 18, caso contrário a linha 8 exibe os status da vaga, a linha 10 envia utiliza uma função Jason

que envia uma mensagem a todos os agentes do sistema para informar que existem vagas disponíveis e que a negociação *“AI” (*identificador da negociação) está acontecendo. Depois de ter anunciado o serviço na linha 10, a linha 11 faz com que o *manager* espere por meio segundo até executar o plano *“+!decide()”* que será apresentado no Código 14. Na linha 12 a crença *“auctionId()”* é atualizada com um número a mais para gerar um valor identificador para o próximo serviço de negociação. Nas linhas 13 e 14 a crença *“counter()”* é atualizada para o valor 1, essa crença é utilizada como uma variável auxiliar na execução do plano *“decide()”*. Por fim o plano executa a função *“.wait(TCFP)”* onde *“TCFP”* é o tempo determinado para que o *manager* aguarde para executar novamente o plano.

Código 13 – Plano startNegotiation manager CNP

```
1 +!startNegotiations : auctionId(AI) &
2                       timeForNewCFP(TCFP) &
3                       freeSpots(FS) & counter(COUNT)
4         <-
5
6         if (FS>0){
7
8             !printSpots;
9             .print("Call for Proposal");
10            .broadcast(tell, auctionCN(service, spot_available("Auction: ",AI)));
11            .at("now + 0.5 seconds", {+!decide(spot_available("Auction: ",AI))});
12            -auctionId(AI); +auctionId(AI+1);
13            -counter(COUNT);
14            +counter(1);
15            .wait(TCFP);
16        }
17
18        !startNegotiations.
```

Fonte: Autoria Própria

O plano *“decide()”* lista todas as ofertas recebidas por um serviço e escolhe, em função de quantas vagas livres existem, os vencedores da vaga. Os vencedores são os agentes *drivers* que ofereceram o maior número de créditos pelo recurso. O Código 14 mostra a implementação deste plano.

Na linha três do Código 14 é utilizada uma função do Jason, a qual gera a lista *“L”* com todas as propostas enviadas por um determinado serviço, com o *driver* *“A”* que a enviou e o valor *“CREDITS”* que o agente se propôs a pagar pelo recurso. Na linha 6 está um condicional para verificar se a crença de vagas livres *“FS”* é maior que zero, se for na linha 7 número de vagas é impresso no *console*, a linha 8 executa uma função do Jason que ordenará as propostas em função do valor da proposta e armazenará a nova lista ordenada em *“SL”*. Na linha 8 mais uma função nativa do Jason é utilizada para calcular quantas propostas foram enviadas pelo

serviço e a armazena em "*PROPOSALSLENGTH".* Se não houverem propostas o *manager* executa a linha 20, senão ele verifica se o número de propostas é maior do que o valor "*COUNT*" o qual é utilizado pelo *manager* como variável auxiliar para selecionar em "*SL*" os agentes vencedores. Se o número de propostas for maior do que o valor de "*COUNT"* é executado o comando a linha 12 onde é impresso no console o lista "*SL"*.

A execução da linha 13 do Código 14 seleciona o n-ésimo *driver* de "*SL"* o qual receberá a vaga, sendo o enésimo o valor de "*PROPOSALSLENGTH"* subtraído por "*COUNT*". O valor de *"COUNT"* é incrementado em 1 na linha 16. Assim a cada iteração, enquanto ainda houverem vagas disponíveis, o *manager* seleciona a próxima melhor proposta da lista (a posição *"PROPOSALSLENGHT–1"* de "*SL"* é onde está a melhor oferta). Com o enésimo *driver*, denominado *"W"*¸ selecionado, o plano "*allocateSpot()"* é executado, utilizando como parâmetros para a execução o *driver "W"* e o valor "*CREDITS"* que será pago pela vaga.

Código 14 – Plano decide Manager CNP

```
1 @decide[atomic]
2 +!decide(Service)     // receives proposals and checks for new winner
3      :  .findall(b(CREDITS,A),proposal(Service,CREDITS)[source(A)],L) &
4         freeSpots(FS) & auctionId(AI) & counter(COUNT)
5      <-
6         if(FS>0){
7             .print("FREE SPOTS: ",FS);
8             .sort(L,SL);
9             .length(SL,PROPOSALSLENGTH);
10            if(PROPOSALSLENGTH>0){
11                if(PROPOSALSLENGTH-COUNT>=0){
12                .print("SORT: ", SL);
13                .nth(PROPOSALSLENGTH-COUNT, SL, b(CREDITS,W));
14                .print("Winner for ",Service,"  is ",W," with ",CREDITS);
15                -counter(COUNT);
16                +counter(COUNT+1);
17                !allocateSpot(W,CREDITS);
18                }
19            } else {
20                .print("There are no proposal");
21            }
22        }
23        if(COUNT<PROPOSALSLENGTH){
24            if(FS-1>0 & PROPOSALSLENGTH>0){
25                !decide(Service);
26            }
27        }
28        if(FS=0 ){
29            .print(" There are no more freeSpots ");
30        }.
```

Fonte: Autoria Própria

Depois de escolhido e alocado a vaga para o *driver* que enviou a n-ésima proposta, o *manager* executa a linha 23 do plano onde verifica se o contador "*COUNT"* não é maior que as ofertas enviadas, pois se for então não existem mais *drivers* requisitantes no serviço.

Nas linhas 23 e 24 o *manager* verifica se ainda existem mais *drivers* que enviaram propostas além dos que já foram alocados, e se existe vaga disponível para ser alocada ao *driver* da próxima proposta. Se as duas condições forem satisfeitas o plano "*decide()*" é executado novamente pelo mesmo serviço.

A participação dos *drivers* nas negociações acontece quando percebem (recebem a mensagem do *manager*) que um serviço de disputa de vagas está aberto, e possuem a crença de estar na lista de espera. Então calculam um valor da proposta e enviam ao *manager* uma mensagem com esse valor. A implementação do plano "*auctionCN()"* do agente *driver* aparece no Código 15.

Código 15 – Plano auctionCN

```
1 +auctionCN(service, D)[source(A)] : inList &
2                                     nOfProposals(N) &
3                                     vOfProposal(V) &
4                                     credits(CDT) &
5                                     increaseFactor(IF)
6                                     & willingnessToPay(WTP)
7     <-
8         if(V<=(CDT*WTP)){
9             .send(A,tell,proposal(D,V));
10            .print("I'm willingness paying: ",V);
11            -vOfProposal(V);
12            +vOfProposal(V*IF);
13            -nOfProposals(N);
14            +nOfProposals(N+1)}
15        else{
16            .send(A,tell,proposal(D,(CDT*WTP)));
17            .print("I'm willingness paying: ",CDT*WTP);
18            -nOfProposals(N);
19            +nOfProposals(N+1);
20        }.
```

Fonte: Autoria Própria

Para um *driver* responder ao serviço com uma proposta ele deve estar na lista de espera por vagas ("*inList")*, conhecer o número de propostas enviadas "*N",* o valor da proposta "*V*", os créditos "*CDT"* que possui, o seu fator de incremento das propostas "*IF"* e o valor "*WTP"* que representa a porcentagem dos créditos que o agente possui e está disposto a desembolsar pela vaga.

Na linha 8 do Código 15 está um condicional que verifica se o valor da proposta é menor ou igual ao valor em créditos que o *driver* está disposto a pagar

pela vaga. Se essa condição for satisfeita, o *driver* envia ao agente "*A"* (*manager)* uma proposta com o valor "*V".* Com isso feito o *driver* ao executar as linhas 11 e 12 atualiza o atualiza o valor da proposta multiplicando-o pelo fator de incremento para uma possível necessidade de enviar uma próxima proposta, e nas linhas 13 e 14 atualiza o número de propostas enviadas. Se a condição não for satisfeita o *manager* envia uma proposta com o valor máximo que está disposto a pagar pela vaga e atualiza a crença com o número de propostas enviadas.

6.3 DESENVOLVIMENTO MAPS LEILÃO INGLÊS

O leilão inglês, apresentado na Seção 4.1.1, é um modelo aberto de leilão onde todos os participantes conhecem todos os lances ofertados, disputando diretamente o recurso. No contexto do MAPS, o agente *manager* iniciará o leilão, ofertando o número de vagas que serão leiloadas naquele serviço, depois de um determinado tempo ele termina o leilão e seleciona os vencedores (se existirem). Já o *driver,* enquanto o serviço de leilão estiver aberto, se ele não estiver entre os ganhadores, deve enviar lances que superem o maior lance até que chegue ao limite de créditos que está disposto a pagar pela vaga.

6.3.1 MAPS Leilão Inglês em Jason

O desenvolvimento deste leilão teve como base o desenvolvimento do *Contract Net* para as funções do *manager.* O plano "*decide()*" utilizado para decidir quais agentes foram os ganhadores é o mesmo apresentado no Código 14. Já o plano *"startNegotiations()"* possui algumas diferenças e será apresentado no código.

Código 16 – Plano startNegotiations manager leilão inglês

```
1 +!startNegotiations : auctionId(AI) & timeForNewAuction(TFNA) & freeSpots(FS)
2                       & counter(COUNT) & lastAuctionFreeSpots(LAFS)
3         <-
4         if (FS>0){
5             !printSpots;
6             .broadcast(untell,freeSpots(LAFS));
7             .broadcast(tell,freeSpots(FS));
8             -lastAuctionFreeSpots(LAFS); +lastAuctionFreeSpots(FS);
9             .print("Starting auction ",AI);
10            .broadcast(tell, englishAuction(service, spot_available("Auction: ",AI)));
11            .at("now + 0.5 seconds", {+!decide(spot_available("Auction: ",AI))});
12            -auctionId(AI); +auctionId(AI+1);
13            -counter(COUNT);
14            +counter(1);
15            .wait(TFNA);
16        }
17        !startNegotiations.
```

Fonte: Autoria Própria

No Código 16, assim como no Código 13, o plano faz com que seja recursivamente executado o plano *"startNegotiations()"* até o sistema ser desligado. Para a execução do leilão é preciso que o condicional da linha 4 seja satisfeito, onde vagas livres "*FS"* deve ser maior que zero. Se satisfeito, a linha 5 mostra as vagas e os seus status. As linhas 6 e 7 são utilizadas para alterar a crença de vagas livres que os *drivers* possuem. A linha 6 é enviado uma mensagem para todos os agentes a qual contradiz "*freeSpots(LAFS)"* onde "*LAFS"* vem de *Last Auction Free Spots*. A informação de vagas livres do leilão anterior é apagada da crença dos *drivers* para na linha 7 ser atualizada com o valor atual de vagas disponíveis, que serão leiloadas no presente serviço de leilão. Na linha 10 o *manager* inicia o serviço de leilão informando a todos os agentes via *broadcast*, e depois de meio segundo ele executa o plano decide do Código 14 para selecionar os vencedores.

O interessante neste mecanismo é a estratégia utilizada pelos *drivers* na competição pelas vagas. O Código 17 apresenta o plano "*englishAuction()"* que é executado quando o *driver* recebe uma mensagem do *manager* informando que há vagas a serem disputadas.

Código 17 – Plano englishAuction disparado pelo manager

```
1  +englishAuction(Service, D)[source(AG)] : increaseFactor(IF) & vInitialBid(INIBID)
2                                             & nOfAuctions(N) & inList
3                                             & credits(CDT) & auctionPosition(AP) &
4                                             willingnessToPay(WTP) & nOfBids(NOB)
5     <-
6          -nOfAuctions(N); +nOfAuctions(N+1);
7           .wait(600*math.random);
8           .findall(b(CREDITS,A),bid(D,CREDITS)[source(A)],L);
9           .length(L,BIDSLENGTH);
10          if(BIDSLENGTH>0){
11               .max(L,b(MV,A));
12               if((MV*IF)<(CDT*WTP)){
13                     .broadcast(tell,bid(D, MV*IF));
14                     .broadcast(achieve,newBid);
15                     -auctionPosition(AP); +auctionPosition(1);
16                     -nOfBids(NOB); +nOfBids(NOB+1);
17                     .print("Auction",N+1 ,".  Bid ",MV*IF);
18                     !englishAuction(Service,D);
19               }
20           }else{
21             if(INIBID<(CDT*WTP)){
22                 .broadcast(tell,bid(D, INIBID));
23                 -auctionPosition(AP); +auctionPosition(1);
24                 -+nOfBids(NOB+1);
25                 .print("Initial bid ",INIBID);
26             } else {
27                 .broadcast(tell,bid(D, CDT*WTP));
28                 -auctionPosition(AP); +auctionPosition(1);
29                 -+nOfBids(NOB+1);
30                 .print("Initial bid ",CDT*WTP);
31             }
32             !englishAuction(Service,D);
33           }.
```

Fonte: Autoria Própria

Quando o plano é iniciado, a linha 6 atualiza o número de leilões participado “*N”* da crença *“nOfAuctions()”*, em seguida aguarda um tempo entre 0 a 600 milissegundos na linha 7. Na linha oito o *driver* executa a função “*.findall”* do Jason que listará todos os lances feitos naquele serviço em *“L”*, e na linha seguinte a função “*.length()”* faz com que o número de lances dados naquele leilão seja armazenado em *“ BIDSLENGTH”.* Se algum lance já houver sido ofertado neste serviço de leilão, o *driver* utiliza a função Jason “*.max()”* para selecionar o maior lance de “*L”* denominado no código “*MV”*. Se o valor de “*MV”* multiplicado pelo “*IF”* *(*fator de aumento) for menor que o máximo de créditos que o *driver* está disposto a pagar pela vaga, o *driver* envia por *broadcast* o valor de *(“MV * IF”)* como lance naquele serviço. Na linha 14 o *driver* envia uma mensagem para que todos os *drivers* executem o plano *newBid()* o qual fará com que eles atualizem sua crença da posição que possuem no leilão para uma a menos. Na linha 15, o agente atualiza a sua posição “*AP*” para 1 por ser o primeiro, na 16 atualiza o número de lances

“nOfBids(NOB)” enviados e na linha 18 executa o plano “*englishAuction()”* que não tem como fonte o *manager*, que o mantém na disputa para efetuar novos lances naquele serviço de leilão.

Se o valor de *“BIDSLENGTH”* for zero significa que ainda não foi enviado nenhum lance naquele leilão. Então o *driver* envia um lance mínimo que será “*INIBID”* (um valor aleatório entre 1 e 2), ou se esse valor foi maior do que o máximo que o *driver* está disposto a pegar pela vaga esse valor máximo será oferecido como lance. Por fim, depois de enviar o lance e fazer atualização nas crença o plano “*englishAuction()”* disparado pelo próprio *driver* é executado. O Código 17 apresenta este plano.

Código 18 – Plano englishAuction disparado pelo driver

```
1  +!englishAuction(Service,D) : increaseFactor(IF) & credits(CDT) & inList &
2                                nOfAuctions(N) & auctionPosition(AP) &
3                                freeSpots(FS) & willingnessToPay(WTP)
4                                & nOfBids(NOB)
5    <- .findall(b(CREDITS,A),bid(D,CREDITS)[source(A)],L);
6       .max(L,b(MV,A));
7       if(AP<=FS){
8             .print("I'm winning auction", N);
9             !englishAuction(Service,D);
10      } else {
11        if ((MV*IF)<(CDT*WTP)){
12            .broadcast(tell,bid(D, MV*IF));
13            .broadcast(achieve,newBid);
14            -auctionPosition(AP); +auctionPosition(1);
15            -+nOfBids(NOB+1);
16            .print("Auction",N,".  Bid ",MV*IF);
17            !englishAuction(Service,D);
18        }
19        else{
20            .print("I can not increase my bid.");
21        }
22      }.
```

Fonte: Autoria Própria

Este plano é executado para que o *driver* possa fazer ofertas que superem a sua primeira. Na linha 5 do Código 18 o *driver* lista em *“L”* todas as propostas e na linha seguinte grava em “*MV”* o valor do lance que está em primeiro no leilão. A linha 7 executa um condicional que analisa se a posição *(“AP”)* do *driver* no leilão é menor ou igual ao número de vagas livres *(“FS”)*. Se for significa que o *driver* com sua proposta anterior está recebendo a vaga e não precisa aumenta-la, e ele volta a executar o plano.

Se ele não estiver entre os ganhadores na linha 11 é analisado se a sua disposição para pagar permite que ele cubra o maior lance ofertado. Se for possível

ele envia o lance com o antigo maior valor multiplicado por seu "*IF*", atualiza suas crenças como acontece no Código 17 e executa novamente o leilão. Senão ele imprime no console, linha 20, que não pode aumentar o lance. O plano "*englishAuction()"* é executado até o *driver* não poder mais aumentar a proposta, ou até o *manager* fechar o serviço por aquele leilão.

6.4 DESENVOLVIMENTO MAPS LEILÃO HOLANDÊS

Conforme apresentado na Seção 4.1.2, o leilão holandês funciona com o leiloeiro iniciando o leilão oferecendo a vaga por um valor alto, e decresce este valor até encontrar um *driver* que esteja disposto a pagar aquele valor pelo recurso. Os *drivers* não disputam por lances, apenas dizem se aceitam pagar o valor que lhes é ofertado.

6.4.1 MAPS Leilão Holandês em Jason

O leilão é iniciado pela execução do plano "*parkingOffer()"* pelo *manager*, plano apresentado no Código 19. Na execução desse plano, que é semelhante aos planos "*startNegotiation()"* dos outros mecanismos, o *manager* utiliza o plano ".*wait"* para esperar o tempo "*TFNA"*, definido nas crenças iniciais. Depois desse tempo ele verifica se existem vagas disponíveis (na linha 4), se houver o *manager* imprime no console a mensagem que existe vagas disponíveis e o valor que ele deseja vende-las é "*SV" (Spot Value)*. Na linha 8 o *manager* envia a todos do sistema uma mensagem informando a abertura do serviço de leilão, e o valor em créditos pelo qual ele deseja vender a vaga. Espera meio segundo até executar o plano para decidir os ganhadores, e depois atualiza a crença identificadora dos leilões e a crença do contador auxiliar. Se não existirem vagas disponíveis o plano "*parkingOffer"* é executado recursivamente.

Código 19 – Plano parkingOffer

```
+!parkingOffer :spotValue(SV) & auctionId(AI) & timeForNewAuction(TFNA)
    <-
        .wait(TFNA);
        if (FS>0){

            !printSpots;
            .print("Spot available, price: ",SV);
            .broadcast(tell, dutchAuction(service,spot_available("Auction: ",AI),SV));
            .at("now + 0.5 seconds", {+!decide(spot_available("Auction: ",AI))});
            -auctionId(AI); +auctionId(AI+1);
            -counter(COUNT);
            +counter(1);
        }else{
            !parkingOffer;
        }.
```

Fonte: Autoria Própria

A execução do plano "*decide()",* apresentado no Código 20, faz a seleção dos vencedores das vagas se houverem. A lista "*L"* é onde estão armazenadas as mensagens dos agentes que aceitarão pagar o valor oferecido pelo *manager* naquele serviço, e "*ACCEPTLENGTH"* é o valor inteiro correspondente a quantas mensagens estão em "*L"*.

Código 20 – Plano decide leilão holandês

```
@decide[atomic]
+!decide(Service)
     :  .findall(b(A),accept(Service)[source(A)],L) & freeSpots(FS) & counter(COUNT) & spotValue(SV)
     <-
        if(FS>0){
            .print("FREE SPOTS: ",FS);
            .length(L,ACCEPTLENGTH);
            .print("List:",L);
            if(ACCEPTLENGTH>0){
                if(ACCEPTLENGTH-COUNT>=0){
                    .nth(ACCEPTLENGTH-COUNT, L, b(W));
                    .print("Winner for ",Service,"  is ",W,". He accept paying ", SV, " for the spot.");
                    -counter(COUNT);
                    +counter(COUNT+1);
                    !allocateSpot(W,SV);
                } else {
                    !refreshSpotValue;
                }
            } else {
                .print("No one accept paying ", SV);
                !refreshSpotValue;
            }
        }
        if(COUNT<=ACCEPTLENGTH){
            if(FS-1>0){
                !decide(Service);
            } else {
                .print(" There are no more freeSpots ");
                !parkingOffer;
            }
        }.
```

Fonte: Autoria Própria

Na linha 9 do plano "*decide()"* é executado uma operação condicional para verificar se algum *driver* aceitou pagar o valor "*SV"* do serviço, se ninguém aceitou o plano *"refreshSpotValue"* (que será apresentado no Código 21) é executado. Se

algum *driver* aceitar pagar "*SV"* a linha 10 é executada para verificar se "*COUNT"* não está maior que significaria que já foram alocadas vagas a todos os agentes que aceitaram pagar o valor "*SV"*, então seria executado a linha 17. Se "*ACCEPTLENGTH"* for maior que *"COUNT"*, então as linhas de 11 a 15 são executadas onde o *manager* selecionará o enésimo ganhador da vaga, o agente "*W"*, imprimirá no console o nome do vencedor e quanto pagou pela vaga, atualizará a crença do contador e alocará a vaga para o *driver "W".*

Na linha 24 do Código 20, se o contador for menor que o número de aceitações a linha 25 é executada, onde o *manager* analisa se ainda existem vagas disponíveis para outro ganhador, se houver o plano "*decide"* é executado novamente, senão o plano *"parkingOffer"* é executado sem ser diminuído o valor da vaga, pois ainda existem agentes dispostos a pagar o valor "*SV*" atual.

O Código 21 apresenta o plano "*refreshSpotValue"* utilizado pelo *manager* para atualizar o valor da vaga quando não existem *drivers* que aceitem pagar o valor "*SV".* Na linha 3 é excluída a crença "*spotValue(SV)"* e na linha 4 o a crença é adicionada com o valor de *"SV"* subtraído pela porcentagem definida em "*DF".* Depois de atualizar o valor das vagas, o plano "*parkingOffer"* é executado novamente.

Código 21 – Plano refreshSpotValue

```
1 +!refreshSpotValue: spotValue(SV) & decreaseFactor(DF)
2     <-
3     -spotValue(SV);
4     +spotValue(SV-(SV*DF));
5     !parkingOffer.
```

Fonte: Autoria Própria

Este mecanismo de negociação é o que o agente *driver* possui menor participação. O papel do *driver* se resume a avisar o *driver* que aceita pagar o valor que lhe foi proposto. O plano "*dutchAuction()"* está no Código 22 e apresenta o comportamento do *driver* na negociação.

Código 22 – Plano dutchAuction

```
1 +dutchAuction(Service,D,SV)[source(AG)] : credits(CDT) &
2                                           willingnessToPay(WTP) &
3                                           inList & messagesSent(MS)
4     <-
5         if(SV<(CDT*WTP)){
6             .send(AG,tell,accept(D));
7             -messagesSent(MS); + messagesSent(MS+1);
8         }
9         else {
10            .print("I will not pay ",SV);
11        }.
```

Fonte: Autoria Própria

Quando o *driver* percebe que o leilão foi aberto, (recebe a mensagem do *manager)* se estiver na lista de espera (possuir a crença "*inList")* ele executa o plano "*dutchAuction".* Na linha 5 é executado uma operação condicional a qual verifica se o valor "*SV"* da vaga oferecida pelo *manager* é menor que o valor em créditos que o *driver* está disposto a pagar. Se for menor, o *driver* envia uma mensagem para o agente "*AG"* (*manager)* informando que aceita pagar o valor requisitado no serviço de leilão "*D*" e atualiza a crença "*messageSent(MS)"* acrescentando mais um. Se for maior, é impresso no console a mensagem que não pagará o valor solicitado pelo *manager.*

6.5 DESENVOLVIMENTO ARTEFATO TIMECONTROL

Para o desenvolvimento do MAPS *ContractNet* foi implementado apenas um artefato em Cartago, o *TimeControl*. Este artefato é responsável por prover ações para que o agente *manager* pudesse controlar o tempo de espera dos *drivers.* O Código 23, apresenta a implementação do artefato.

Código 23 – Artefato TimeControl

```
1 public class TimeControl extends Artifact {
2
3     public LinkedList <Driver> timeControl;
4     Driver driver;
5
6     void init(String msg) {
7         timeControl = new LinkedList<Driver>();
8         System.out.println("System message: " + msg );
9     }
10
11
12
13    @OPERATION
14    public void driverRequestedSpot(Object idDriver){
15        driver = new Driver(idDriver.toString(), new Date());
16        timeControl.add(driver);
17    }
18
19    @OPERATION
20    public void totalWaitingTime(Object idDriver, OpFeedbackParam<Integer> wtDriver){
21        Driver d = findId(idDriver.toString());
22        wtDriver.set((int) (new Date().getTime() - d.getArrivalTime().getTime()));
23    }
24
25    public Driver findId(String s){
26
27        Driver d1 = new Driver();
28        for(Driver d : timeControl){
29            if(s == d.getId())
30                d1 = d;
31        }
32        return d1;
33    }
34 }
```

Fonte: Autoria Própria

Quando um *driver* faz a requisição por uma vaga, o *manager* executa a operação *driverRequestedSpot()*, a qual tem como parâmetro de entrada o nome do *Driver*. Com o nome é criado um novo objeto do tipo *Driver* e este é armazenado na *LinkedList timeControl.*

6.6 CONSIDERAÇÕES FINAIS

Este capítulo apresentou o desenvolvimento do MAPS-*Auctions.* Foi apresentada a funcionalidades dos agentes *drivers* e *manager* dentro do SMA, e como foi feita a implementação utilizando o JaCaMo.

O Capítulo 7 apresentará os cenários criados para testar os mecanismos implementados e os resultados que cada mecanismo apresentou com relação ao: tempo de espera dos *drivers*; valor pago pelas vagas; e o número de mensagens trocadas em cada negociação.

7 RESULTADOS

Os mecanismos de negociação apresentados no capítulo anterior foram implementados com o intuito de apresentar novos resultados com relação à distribuição de recursos em SMA. Contudo,Não é o foco do trabalho a análise o funcionamento mecânico do estacionamento (como o abrir e fechar de cancelas) e não são consideradas possíveis corrupções dos agentes, como estacionar em outra vaga que não a destinada ou não realizar o pagamento.

Para análise da implementação do leilão inglês, holandês e do *Contract Net* ao contexto do projeto MAPS, são três os parâmetros de avaliação:

- O valor em créditos recebido pelo *manager* ao final de cada teste;
- O tempo que levará desde o *driver* requisitar a vaga, até o momento em que ele executa o plano para estacionar;
- A quantidade de mensagens e ofertas, que o *driver* envia para o *manager* durante a negociação.

Na sequência deste capítulo estão os cenários utilizados para testar os SMA, os resultados obtidos a partir dos cenários em cada mecanismo de negociação, e por fim será apresentado uma seção de comparação entre os mecanismos.

7.1 CENÁRIOS DE TESTE

A fim de observar o funcionamento dos SMA desenvolvido, os comportamentos de 9 *drivers* foram analisados durante as negociações em 4 cenários diferentes, em cada um dos três mecanismos de negociação.

A Tabela 1 apresenta as configurações dos cenários de testes, onde são descritos o nome do cenário, o número de agentes *drivers* e o número de vagas em cada cenário.

Tabela 1 – Configuração dos cenários

Cenário	Número de *Drivers*	Número de Vagas
0	9	20
1	9	9
2	9	1
3	90	5

Fonte: Autoria própria

Os números de agentes e vagas dos cenários foram escolhidos para observarmos o funcionamento do estacionamento quando:

- **Cenário 0**: Há recursos de sobra para os solicitantes;
- **Cenário 1**: O número de *drivers* é igual ao de vagas disponíveis;
- **Cenário 2**: Existe apenas uma vaga, para vários motoristas que querem estacionar;
- **Cenário 3**: Há pouco recurso para muitos solicitantes.

A Tabela 2 apresenta as diferentes configurações dos agentes usados nos experimentos. São mostrados o tempo para requisitar uma vaga (TTR- *time to request*), o tempo que cada agente permaneceu na vaga (TS – *time to spend*) e o número de créditos que cada agente possui inicialmente(CDT - *credits*).

Tabela 2 – Configuração dos agentes

Agente	timeToRequest(TTR)	timeToSpend(TS)	Gasto Máximo	Credits(CDT)
Arriscado_1	6s	6s	100%	95
Arriscado_2	6s	6s	100%	50
Arriscado_3	6s	6s	100%	5
Moderado_1	6s	6s	50%	95
Moderado_2	6s	6s	50%	50
Moderado_3	6s	6s	50%	5
Conservador_1	6s	6s	25%	95
Conservador_2	6s	6s	25%	50
Conservador_3	6s	6s	25%	5

Fonte: Autoria Própria

Com o objetivo de analisar como diferentes comportamentos dos *drivers* em cada cenário, foram definidos três perfis aos agentes:

- Agentes arriscados (Arriscado_X): os agentes pertencentes a esse grupo possuem grande necessidade da vaga, e estão dispostos a gastar até 100% de seus créditos por uma vaga.
- Agentes moderados (Moderado_X): estão no meio termo entre os grupos arriscados e conservadores. Possuem certa necessidade pela vaga, mas não estão dispostos a despender todos os seus créditos. Esses agentes podem desembolsar até 50% dos créditos que possuem pela vaga;
- Agentes conservadores (Conservador_X): simulam motoristas que não possuem grande necessidade de conseguir a vaga no momento, e estão pouco dispostos a pagar altos valores por ela. Sendo assim, eles estão dispostos a gastar até 25% de seus créditos pela vaga;

Para os testes efetuados serão analisados em cada perfil um agente com um elevado número de créditos, um com médio e outro com poucos créditos disponíveis, para cada perfil.

No Cenário 3, além dos nove *drivers* que aparecem na Tabela 2, ainda serão utilizados mais vinte e sete agentes de cada perfil. Esses agentes terão o valor de créditos aleatório dentro do intervalo [1-100], assim pode-se observar possíveis mudanças ao executar o experimento repetidas vezes.

7.2 RESULTADOS MAPS *CONTRACT NET*

Durante a negociação do *Contract Net*, os *drivers* não possuem acesso às propostas enviadas por outros *drivers*. Sendo assim eles incrementam a sua proposta com base na própria oferta anterior. Além das características de cada perfil definidas neste capítulo, os *drivers* de cada perfil possuem estratégias diferentes, definidas empiricamente, para incrementarem as propostas, sendo elas:

- Agentes conservadores: Aumentam sua proposta em 10% a cada nova proposta enviada;
- Agentes moderados: Incrementam em 30% a proposta a cada nova oferta;
- Agentes arriscados: Incrementam em 50% a proposta a cada nova oferta.

Se os agentes necessitarem incrementar até alcançarem o valor máximo que estão dispostos a pagar pela vaga, eles continuarão enviando propostas com o valor máximo que ele está disposto a pagar.

O valor inicial das propostas é um valor aleatório entre [1-2], gerado por uma função do Jason quando o agente é iniciado, desde que não passe o valor máximo que o *driver* está disposto a pagar pela vaga. Se passar, o valor inicial será o valor máximo que ele está disposto a pagar.

A seguir serão analisados os cenários construídos.

7.2.1 Contract Net Protocol Cenário 0

Neste cenário foram utilizados os nove *drivers* apresentados na Tabela 2, disputando um total de vinte vagas.

Foram realizadas duas simulações neste cenário a fim de obter clareza sobre possíveis mudanças. Os valores obtidos nas simulações são mostrados nas Tabelas 3 e 4. Estas simulações mostram que com vagas de sobra os *drivers* recebem o recurso praticamente ao mesmo tempo, com uma diferença de milissegundos gerada pela demora do *manager* a executar o plano de decisão e enviar as mensagens.

Tabela 3 – Resultados simulação CNP 0.1

Nome do agente	Créditos(CDT)	Valor pago	Tempo(s)	Propostas enviadas	Ordem de aquisição
arriscado_1	95	1,28	7,56	1	7
arriscado_2	50	1,18	7,8	1	8
arriscado_3	5	1,11	7,98	1	9
moderado_1	95	1,23	7,8	1	6
moderado_2	50	1,7	7,14	1	1
moderado_3	5	1,52	7,38	1	3
conservador_1	95	1,63	7,32	1	2
conservador_2	50	1,49	7,44	1	4
conservador_3	5	1,25	7,62	1	5
Total		12,39	68,04	9	
Média		1,38	7,56	1	

Fonte: Autoria própria

Este cenário foi simulado duas vezes para analisar a consistência da negociação, e observar o impacto dos valores iniciais aleatórios no cenário.

Tabela 4 – Resultados simulação CNP 0.2

Nome do agente	Créditos(CDT)	Valor pago	Tempo(s)	Propostas enviadas	Ordem de aquisição
arriscado_1	95	1,24	7,38	1	6
arriscado_2	50	1,62	7,32	1	2
arriscado_3	5	1,67	7,26	1	1
moderado_1	95	1,07	7,44	1	9
moderado_2	50	1,17	7,38	1	7
moderado_3	5	1,16	7,44	1	8
conservador_1	95	1,27	7,32	1	4
conservador_2	50	1,56	7,32	1	3
conservador_3	5	1,25	7,38	1	5
Total		12,01	66,24	9	
Média		1,33	7,36	1	

Fonte: Autoria própria

Além do agente *conservador_3*, todos pagam o valor inicial aleatório que sempre será dentro do intervalo [1,2]. O *conservador_3* paga o seu teto pela vaga nas duas ocasiões, porque o valor aleatório gerado ultrapassa os 1,25 que ele está disposto a pagar.

Fica claro que, com este mecanismo, quando há mais vagas que requisitantes, não é importante o quanto os *drivers* estão dispostos a pagar, nem os créditos que possuem, pois todos pagarão o valor da primeira proposta ofertada, e logo receberão a vaga.

7.2.2 Contract Net Protocol Cenário 1

A execução deste cenário apresenta resultados semelhantes aos do cenário 0, onde a maior oferta leva primeiro a vaga, porém a diferença para os *drivers* receberem o recurso continua na casa dos milissegundos.

A Tabela 5 apresenta os resultados da execução do cenário 1.

Como o recurso disponível é igual ao número de requisitantes, a proposta inicial é suficiente para que todos os *drivers* consigam adquirir o recurso. A diferença entre o primeiro e o último a receber é de apenas 0,6 segundos.

Tabela 5 – Resultados simulação CNP 1

Nome do agente	Créditos(CDT)	Valor pago	Tempo(s)	Propostas enviadas	Ordem de aquisição
arriscado_1	95	1,74	7,26	1	2
arriscado_2	50	1,02	7,56	1	9
arriscado_3	5	1,64	7,26	1	4
moderado_1	95	1,55	7,44	1	5
moderado_2	50	1,72	7,32	1	3
moderado_3	5	1,16	7,5	1	8
conservador_1	95	1,54	7,44	1	6
conservador_2	50	1,88	6,96	1	1
conservador_3	5	1,25	7,44	1	7
Total		13,5	66,18	9	
Média		1,5	7,35	1	

Fonte: Autoria própria

7.2.3 Contract Net Protocol Cenário 2

Este é o primeiro cenário executado no CNP onde os *drivers* disputaram vagas, pois há mais requisitantes do que vagas disponíveis. Neste experimento os *drivers* passam a utilizar sua estratégia de incremento das propostas, que nos cenários 0 e 1 não foram necessárias pois enviavam apenas uma proposta para adquirir a vaga.

A Tabela 6 apresenta os valores da execução do teste nesse cenário.

Tabela 6 – Resultados simulação CNP 2

Nome do agente	Créditos(CDT)	Valor pago	Tempo(s)	Propostas enviadas	Ordem de aquisição
arriscado_1	95	2,58	13,38	2	2
arriscado_2	50	3,92	25,5	4	4
arriscado_3	5	3,43	19,38	3	3
moderado_1	95	3,4	31,5	5	5
moderado_2	50	4,14	37,68	6	6
moderado_3	5	1,91	7,14	1	1
conservador_1	95	2,87	43,68	7	7
conservador_2	50	2,22	49,74	8	8
conservador_3	5	1,25	55,74	9	9
Total		2,86	283,74	45	
Média		25,72	31,53	5	

Fonte: Autoria própria

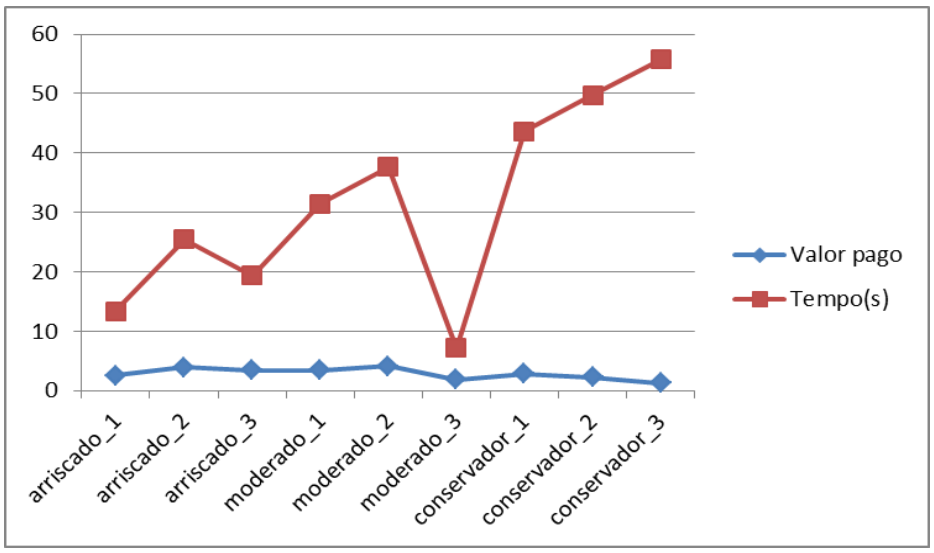
Nesta disputa de nove *drivers* por uma vaga, a ordem que recebem a vagas é o mesmo tanto de propostas que eles enviam para recebê-la, ou seja enviam uma mensagem a cada anuncio do *manager* até receber o recurso. O Intervalo de tempo entre os *drivers* é quase constante pelo fato de todos utilizarem o estacionamento por apenas 6 segundos, assim que liberam a vaga é alocada ao próximo *driver*.

O valor total recebido pelo *Smart Parking* quase dobrou comparado com aos experimentos onde não houve a disputa. O *driver* que mais pagou pela vaga foi o *moderado_2*, 4,14 créditos, incrementando em seis oportunidades sua proposta inicial até chegar a esse valor.

É interessante observar que os agentes *moderado_1* e *moderado_2*, desembolsam valores semelhantes aos de perfil arriscado, porém demoram mais para receber as vagas. O motivo é a estratégia utilizada pelos moderados. Como eles não possuem conhecimento a respeito das propostas de outros *drivers*¸ o aumento de 30% a cada proposta faz com que eles cheguem neste cenário a valores próximos aos pagos pelos arriscados, que têm o fator de incremento em 50%, mais tarde.

O único *driver* que paga com relação aos créditos que possui outra vez é o conservador_3, que atinge o seu valor máximo de disposição pela vaga. Exemplo disso é o fato de o arriscado_1, que possui mais créditos que os outros dois agentes do mesmo perfil, é o que menos pagou pela vaga dos três.

Gráfico 1 – Tempo e valores da simulação CNP2



Fonte: Autoria própria

Com o Gráfico 1, que ilustra os valores pagos pelos agentes e o tempo que levou para cada um deles receber a vaga, observa-se que além do moderado_3 que teve o maior valor aleatório na primeira chamada por propostas do *manager,* primeiro os arriscados receberam, depois os moderados e por fim os conservadores. Isso acontece por causa do fator de incremento utilizado por cada perfil.

7.2.4 Contract Net Protocol Cenário 3

No Cenário 3 noventa agentes, trinta de cada perfil, disputaram cinco vagas. A Tabela 7 apresenta os valores do teste e o Gráfico 2 ilustra os resultados de tempo e do valor pago de cada um dos nove agentes conhecidos, no primeiro teste executado. A ordem de aquisição das tabelas diz respeito apenas aos agentes observados.

Com mais agentes rodando, passou a ser mais relevante o limite de créditos disponíveis para a vaga. Dos nove agentes, os três que possuem o menor número de créditos foram os que mais esperaram para receber a vaga, mesmo utilizando o limite de pagamento.

Apesar disso, o padrão para os outros agentes foi mantido, onde os arriscados receberam o recurso primeiro, depois os moderados e por fim os conservadores. Porém a diferença entre o maior preço pago pelo *arriscado_1* e o *moderado_2* (Os que pagaram mais dos dois perfis) chegou a 6,29 créditos, diferença essa apenas 0,57 créditos menor que a média paga pelos nove *drivers*.

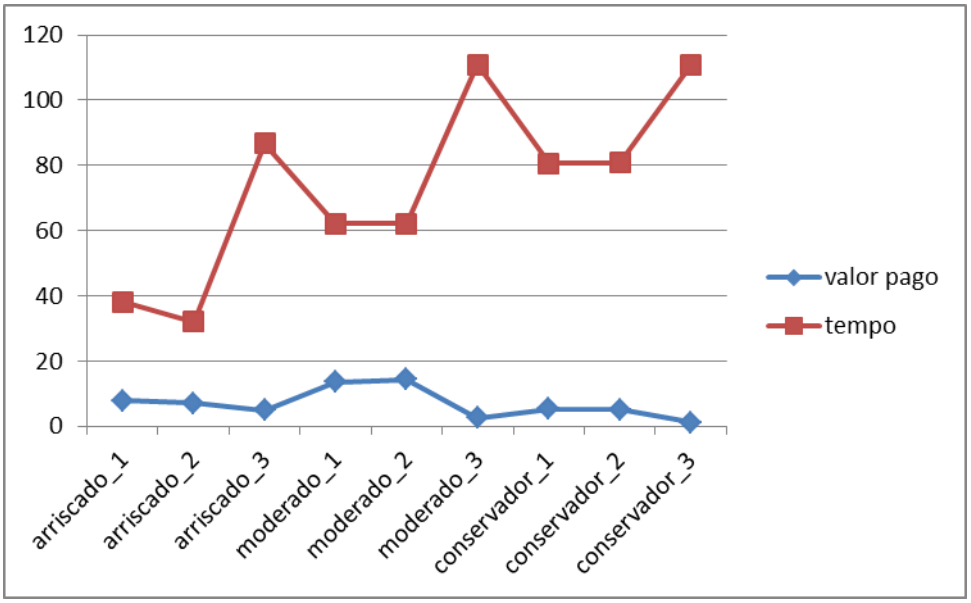
Tabela 7 – Resultados simulação CNP 3.1

Nome do agente	Créditos(CDT)	Valor pago	Tempo(s)	Propostas enviadas	Ordem de aquisição
arriscado_1	95	7,78	38,16	6	2
arriscado_2	50	6,98	32,04	5	1
arriscado_3	5	5	86,7	14	7
moderado_1	95	13,55	62,22	10	3
moderado_2	50	14,31	62,1	10	4
moderado_3	5	2,5	110,82	18	8
conservador_1	95	5,3	80,58	13	5
conservador_2	50	5,13	80,76	13	6
conservador_3	5	1,25	110,82	18	9
Total		61,8	664,2	107	
Média		6,87	73,8	11,89	

Fonte: Autoria própria

Essa grande diferença acontece porque mesmo depois que os conservadores já conseguiram vagas, os moderados ainda disputam entre si. A cada nova iteração o incremento da proposta é maior.

Gráfico 2 – Tempo e valores da simulação CNP3.1



Fonte: Autoria própria

Na Tabela 8 são mostrados os dados do segundo teste realizado deste cenário no CNP. Este segundo teste é importante para observarmos o impacto da aleatoriedade na primeira proposta.

O agente *moderado_2* que na primeira simulação, dentre os nove agentes analisados, foi o que mais pagou pela vaga esperou 62,1 segundos e foi apenas o quarto a receber a vaga, nesta simulação ele foi o segundo que menos pagou e o

que menos esperou, apenas 7,68 segundos, pois ofereceu a primeira proposta com um valor elevado.

Tabela 8 – Resultados simulação CNP 3.2

Nome do agente	Créditos(CDT)	Valor pago	Tempo(s)	Propostas enviadas	Ordem de aquisição
arriscado_1	95	7,87	38,1	6	3
arriscado_2	50	2,74	13,8	2	2
arriscado_3	5	5	98,88	16	7
moderado_1	95	14,16	68,28	11	4
moderado_2	50	1,92	7,68	1	1
moderado_3	5	2,5	110,7	18	8
conservador_1	95	5,16	74,46	12	5
conservador_2	50	6,07	98,7	16	6
conservador_3	5	1,25	110,94	18	9
Total		5,18	621,54	100	
Média		46,67	69,06	11,11	

Fonte: Autoria própria

Esta aleatoriedade inicial também tem um importante impacto nos lucros do *Smart Parking*. Contando os noventa agentes da simulação, a diferença do total recebido entre as simulações é 8,96% (614,49 recebidos na primeira e 563,94 na segunda).

7.2.5 Análise dos Resultados *Contract Net*

Os cenários 0 e 1 aplicados ao CNP tiveram resultados semelhantes. Na realização das simulações os agentes não competiram entre si e obtiveram as vagas depois de uma curta espera por um baixo preço.

Já nos cenários 2 e 3, como houve a disputa pelo recurso foi possível observar características importantes a respeito do comportamento dos agentes, como:

- *Drivers* arriscados possuem a melhor estratégia neste mecanismo. Isso acontece pelo fato de aumentarem rapidamente a oferta, passando na frente dos outros agentes sem precisarem fazer grandes incrementos na proposta depois de vários envios.
- Os conservadores em geral ficam por último, porém nem sempre pagam menos. Exemplo disso aparece na simulação CNP2, onde o agente

conservador_1 paga 0,29 créditos a mais que o agente arriscado_1, porém aguarda 30,3 segundos a mais para receber o recurso.

- A pior estratégia parece ser a dos moderados. Como os incrementos das propostas funcionam como juros compostos, a cada nova iteração o valor de 30% incrementado sobe bastante. No final da simulação eles pagam mais pelas vagas, mas geralmente eles esperem os arriscados receberem as vagas primeiro. Na simulação CNP3.2 pode-se ver isso claramente quando o *moderado_1* precisa enviar 11 propostas e pagar 14,16 créditos pela vaga, enquanto o *arriscado_1* envia 6 e paga 7,87.

Outro ponto de avaliação desse trabalho diz respeito a análise do lucro do *Smart Parking.* As simulações do cenário 3 mostraram que o custo médio para adquirir as nove vagas foi de 54,23 créditos, enquanto no cenário 2 os *drivers* desembolsaram 25,72 para todos estacionarem. Como no cenário 3 a proporção é de o dobro de agentes por vaga em relação ao cenário 2, pode-se deduzir que a proporção *drivers* por vaga está diretamente relacionada ao total recebido pelo estacionamento.

Apesar de dobrar o valor recebido quando a proporção *drivers* por vaga dobra, o mecanismo não consegue extrair, para o estacionamento, o máximo dos *drivers.* Por exemplo o agente *arriscado_1*, que estava disposto a pagar 95 créditos em uma vaga, pagou no máximo 7,87 nas simulações apresentadas.

7.3 RESULTADOS MAPS LEILÃO INGLÊS

Com o objetivo de conseguir vagas, neste mecanismo de negociação os *drivers* competem entre si desde que o *manager* anunciar a disponibilidade de vagas. Neste mecanismo os valores do fator de incremento são mantidos como o do *Contract Net Protocol* para cada perfil (arriscados 50%, moderados 30% e conservadores 10%). A diferença entre o *Contract Net* é que os *drivers* utilizam este fator para ofertarem lances maiores que o maior lance ofertado por outros *drivers* que estão na disputa, não mais a respeito da própria oferta anterior.

Todos os agentes presentes no SMA percebem todos os lances já ofertados, e possuem conhecimento (fornecido pelo *manager* a cada novo leilão) de quantas vagas estão disponíveis, sendo assim não ofertam novos lances se estiverem entre

os ganhadores mesmo que existam ofertas maiores que a sua. É importante salientar que os *drivers* só podem dar lances maiores que o maior lance ofertado até o momento naquele leilão.

A seguir serão apresentadas simulações dos quatro cenários utilizando o Leilão Inglês.

7.3.1 Leilão Inglês Cenário 0

Neste cenário existem vinte vagas disponíveis para nove *drivers* requisitando pelo recurso. A Tabela 9 apresenta os valores da simulação do cenário no MAPS Leilão Inglês.

Tabela 9 – Resultados simulação leilão inglês 0

Nome do agente	Créditos(CDT)	Valor pago	Tempo(s)	Ordem de aquisição	Lances	Leilões
arriscado_1	95	1,71	5,52	6	1	1
arriscado_2	50	4,78	5,34	2	1	1
arriscado_3	5	2,21	14,28	7	1	2
moderado_1	95	3,19	5,34	3	1	1
moderado_2	50	2,23	5,52	5	1	1
moderado_3	5	1,47	14,28	8	1	2
conservador_1	95	5,26	5,28	1	1	1
conservador_2	50	2,45	5,4	4	1	1
conservador_3	5	1,13	23,34	9	1	3
Total		24,43	84,3		9	
Média		2,71	9,37		1	1,44

Fonte: Autoria própria

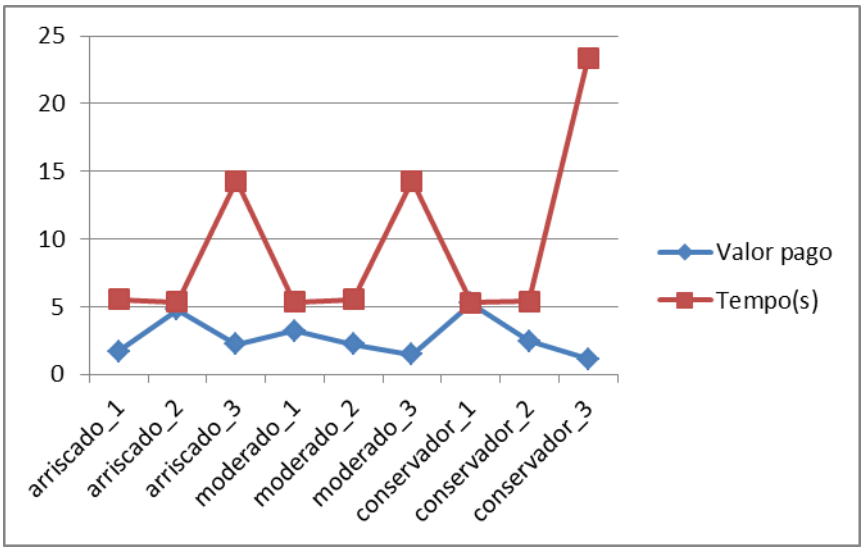
Nesta negociação, apenas o primeiro lance é aleatório (valor no intervalo [1,2]). Os lances a seguir serão maiores que o maior lance da negociação, em função do fator de incremento de cada agente. Sendo assim sempre que houver mais do que um agente requisitando vagas, haverá competição.

Observamos que o lance inicial foi ofertado pelo *driver arriscado_1* em 1,71 créditos. A partir desse lance, o *driver moderado_2* ofertou 30% a mais por uma vaga, o *conservador_2* ofertou 10% a mais que o *moderado_2*, o *moderado_1* ofertou 30% a mais que o *conservador_2*, o *arriscado_2* ofertou 50% a mais que o *conservador_2* e por fim o maior lance veio do agente *conservador_1*, o qual ofertou 5,26 créditos, 10% a mais que o *arriscado_2*.

Neste primeiro leilão apenas seis dos nove agentes que estavam participando do leilão conseguiram vagas. Isso aconteceu porque no momento em que os agentes que possuem 5 créditos foram dar os seu lances, o valor do maior lance era maior do que o que eles estavam dispostos a pagar pelo recurso impossibilitando que o lance fosse efetuado, fazendo com que eles esperassem até o próximo leilão.

O Gráfico 3 ilustra este processo. Pode-se ver os agentes ganhadores no primeiro leilão recebendo vagas na casa de cinco segundos, os agentes *arriscado_3* e *moderado_3* recebendo com pouco mais de 14 segundos, e o *conservador_3* que estava disposto a pagar apenas 1,25 pela vaga teve que participar de mais um leilão esperando mais de 23 segundos para estacionar.

Gráfico 3 – Tempo e valores da simulação leilão inglês 0



Fonte: Autoria própria

Ainda existem vagas disponíveis no estacionamento, mas a alocação não acontece logo em seguida para os outros leilões, porque o *manager* depois de realizar um leilão, aguarda 9 segundos para iniciar o próximo.

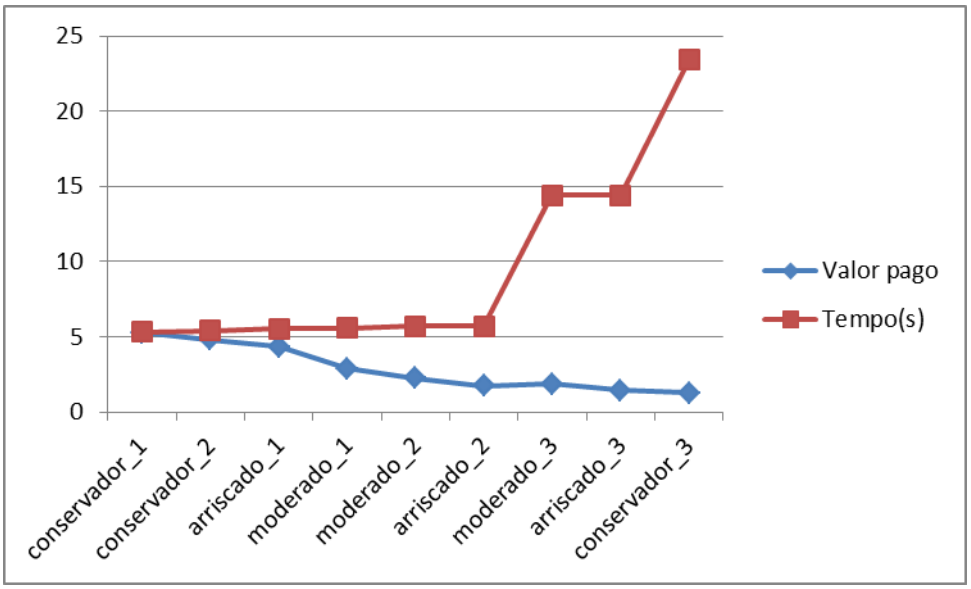
Observa-se que o mecanismo leilão inglês passa a explorar mais a disputa e disposição dos *drivers* em adquirir a vaga. O *driver conservador_1* pagou três vezes mais pela vaga do que o *arriscado_1*, mesmo os dois recebendo o recurso em um mesmo leilão.

7.3.2 Leilão Inglês Cenário 1

Os resultados da simulação do cenário 1 no leilão Inglês foram parecidos com a do cenário 0, onde os *drivers* precisaram enviar apenas um lance para receberem as vagas, e os *drivers* com menos créditos precisaram participar de mais de um leilão quando não conseguem dar um lance antes de chegar no seu limite.

O Gráfico 4 apresenta os valores do tempo e de créditos gastos pelos agentes ordenados pela sequência que receberam as vagas.

Gráfico 4 – Tempo e valores da simulação leilão inglês 1.1



Fonte: Autoria própria

Como no cenário 0, existe uma grande diferença entre o *driver* que pagou mais e o que menos pagou e saíram vencedores de um mesmo leilão, mas como os conservadores deram os lances por último quando os preços estiveram elevados não foram tão incrementados.

Neste cenário, onde existem mais vagas que requisitantes, é importante ao *driver* ser o mais rápido para fazer a oferta, porque os primeiros a darem lances são os que pagam menos pela vaga.

A Tabela 10 apresenta os resultados da simulação 1.2 do leilão inglês, com os motoristas ordenados também por ordem que receberam as vagas. Nela é possível ver que novamente cada *driver* oferta apenas um lance para conseguir a vaga, apesar de alguns participarem de dois leilões.

Nesta simulação é possível observar uma diferença ainda maior entre os valores pagos em um mesmo leilão, pois os conservadores dão lances primeiro, depois os arriscados e por fim os moderados.

Tabela 10 – Resultados simulação leilão inglês 1.2

Nome do agente	Créditos(CDT)	Valor pago	Tempo(s)	Ordem de aquisição	Lances	Leilões
moderado_1	95	8,06	5,1	1	1	1
moderado_2	50	6,2	5,1	2	1	1
arriscado_1	95	4,77	5,16	3	1	1
arriscado_2	50	3,18	5,16	4	1	1
arriscado_3	5	2,12	5,22	5	1	1
conservador_2	50	1,41	5,22	6	1	1
conservador_1	95	1,28	5,22	7	1	1
moderado_3	5	1,43	14,1	8	1	2
conservador_3	5	1,09	14,16	9	1	2
Total		29,543,28	64,44		9	
Média		3,28	7,16		1	1,22

Fonte: Autoria própria

No leilão inglês, a aleatoriedade da oferta inicial possui menor impacto no total recebido pelo *Smart Parking*, em relação à sequência dos lances pelos *drivers* por causa dos seus perfis. Na simulação 1.1 a oferta inicial do primeiro leilão foi alta, 1,72 créditos ofertados pelo agente *arriscado_2*, e ao final da simulação o estacionamento recebeu 25,87 créditos dos nove *drivers*. Já na simulação 1.2 mesmo a primeira oferta sendo 1,28, como os perfis mais agressivos deram lances posteriormente no primeiro leilão, o estacionamento faturou 29,54 créditos.

7.3.3 Leilão Inglês Cenário 2

Nas simulações deste cenário como os nove *drivers* disputarão por apenas uma vaga, serão realizados nove leilões, a cada realização um *driver* receberá o recurso.

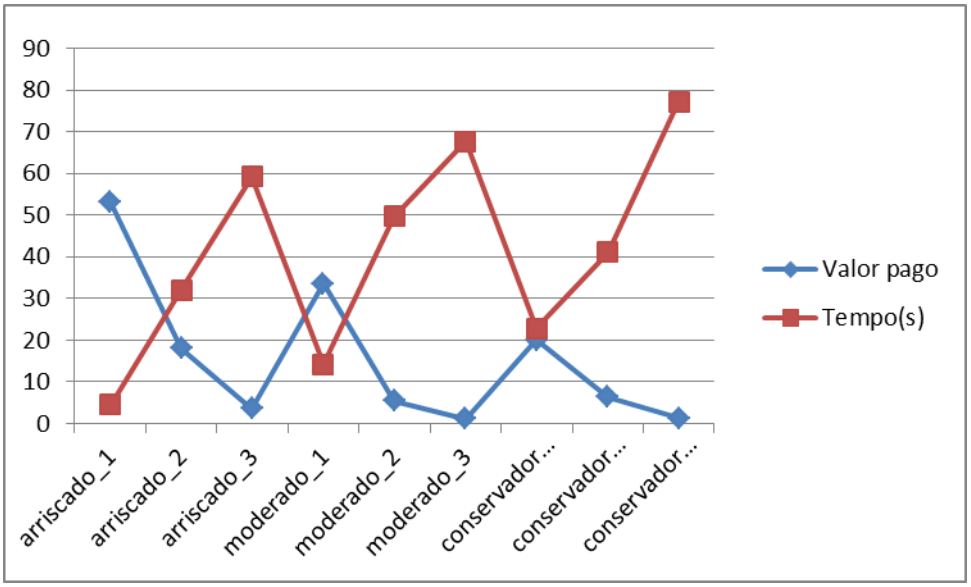
A Tabela 11 mostra os valores gerados pela simulação 2.1 do leilão inglês, e o Gráfico 5 ilustra essa simulação.

Tabela 11 – Resultados simulação leilão inglês 2.1

Nome do agente	Créditos(CDT)	Valor pago	Tempo(s)	Ordem de aquisição	Lances	Leilões
arriscado_1	95	53,08	4,5	1	3	1
arriscado_2	50	18,12	32,04	4	12	4
arriscado_3	5	3,65	59,16	7	10	7
moderado_1	95	33,53	14,22	2	5	2
moderado_2	50	5,52	49,74	6	10	6
moderado_3	5	1,22	67,74	8	6	8
conservador_1	95	20,01	22,86	3	9	3
conservador_2	50	6,33	41,16	5	44	5
conservador_3	5	1,25	77,22	9	4	9
Total		142,71	368,64		103	
Média		15,86	40,96		11,44	5

Fonte: Autoria própria

Gráfico 5 – Tempo e valores da simulação leilão inglês 2.1



Fonte: Autoria própria

A quantidade de créditos que o *driver* possui é importante para esse mecanismo de negociação. Comparando com os outros do mesmo perfil, quem mais possui créditos recebeu a vaga antes. Observa-se que nesta simulação os três *drivers* que possuem mais créditos foram os que mais pagaram pelas vagas e foram os primeiros dos nove a receberem as vagas.

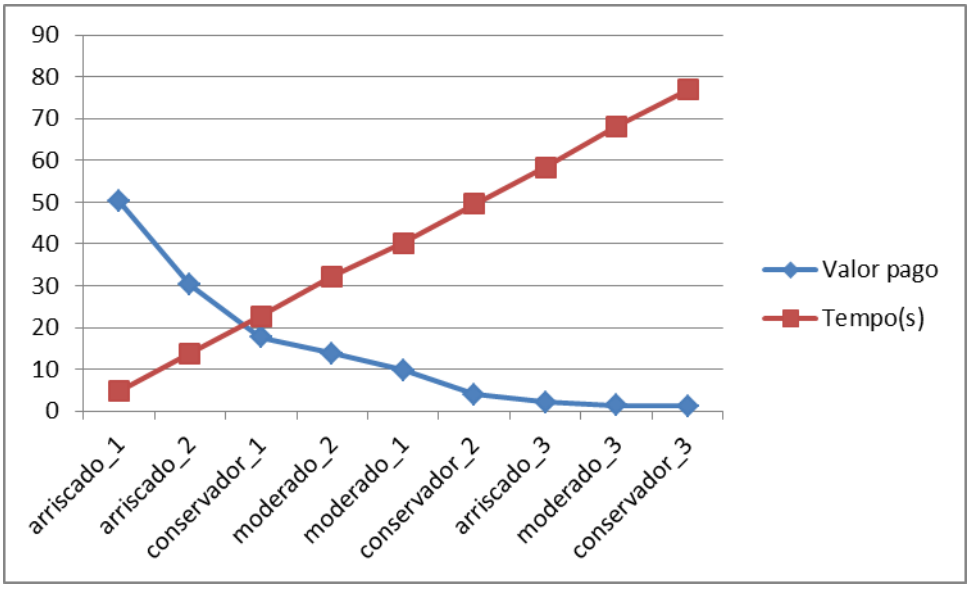
No terceiro leilão realizado, que foi vencido pelo *conservador_1*, o *arriscado_2* teria créditos para superar o lance e ganhar a vaga antes. Isso não aconteceu porque o tempo do leilão, definido pelo *manager*, esgotou antes que o

arriscado_2 cobrisse o lance. Isso acontece no quinto leilão também, que poderia ser vencido pelo *moderado_2*, mas quem venceu foi o *conservador_2*.

O número de lances subiu bastante com relação ao cenários 0 e 1. O arriscado_1 precisou enviar três lances para que sua oferta não fosse mais superada no primeiro leilão, enquanto conservador_2 enviou 44 lances para ganhar a vaga.

O Gráfico 6 mostra uma segunda simulação deste cenário, ordenada por ordem de recebimento da vaga, onde fica claro que a cada novo leilão o preço cai, principalmente entre os três primeiros leilões.

Gráfico 6 – Tempo e valores da simulação leilão inglês 2.2



Fonte: Autoria própria

O *driver arriscado_1* foi vencedor nas duas oportunidades, pois a quantidade grande de créditos junto à porcentagem de incremento a cada lance o faz conseguir superar qualquer oferta dos outros agentes. O contrário acontece com o *conservador_3*, ficando sempre por último, pagando pouco pelo recurso.

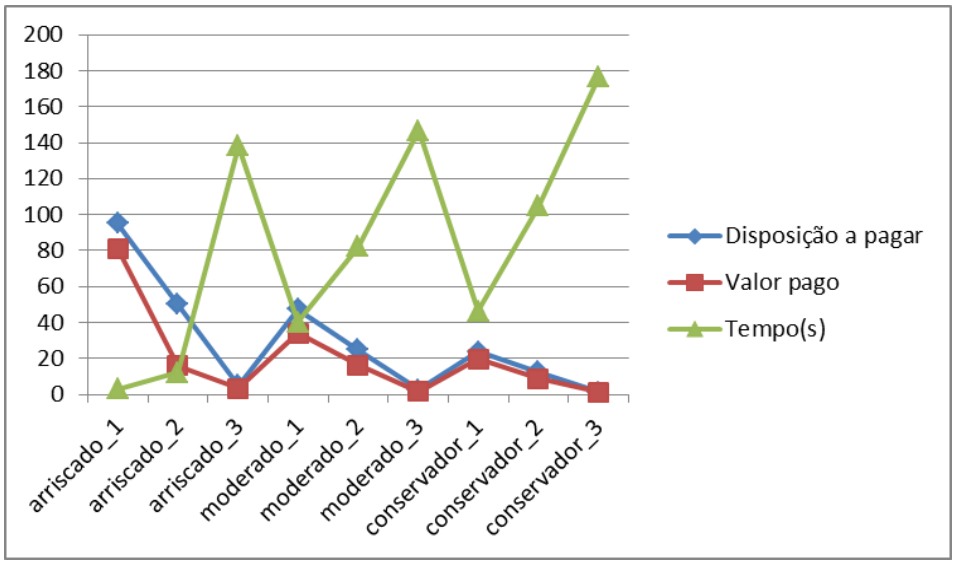
Nas duas simulações o tempo médio ficou na casa dos 40 segundos, assim infere-se a consistência no processo de alocação das vagas. Porém, apesar de a diferença entre o total recebido nas duas simulações ser de apenas 12,3 créditos, a mudança na ordem em que os agentes enviam lances e o tempo até enviarem novos lances possibilitam grandes variações a cada simulação, dificultando estimativas de receitas.

7.3.4 Leilão Inglês Cenário 3

Nas simulações deste cenário mais de uma vaga é alocada a cada leilão. E além dos nove *drivers* conhecidos, são adicionados mais 81 agentes com créditos aleatórios nas simulações.

A simulação 3.1 do leilão inglês apresenta uma exploração da disponibilidade que os *drivers* possuem em empregar seus créditos para adquirir a vaga. O Gráfico 7 ilustra e a Tabela 12 apresenta os resultados da simulação:

Gráfico 7 – Tempo, valores e disposição da simulação leilão inglês 3.1



Fonte: Autoria própria

Tabela 12 – Resultados simulação leilão inglês 3.1

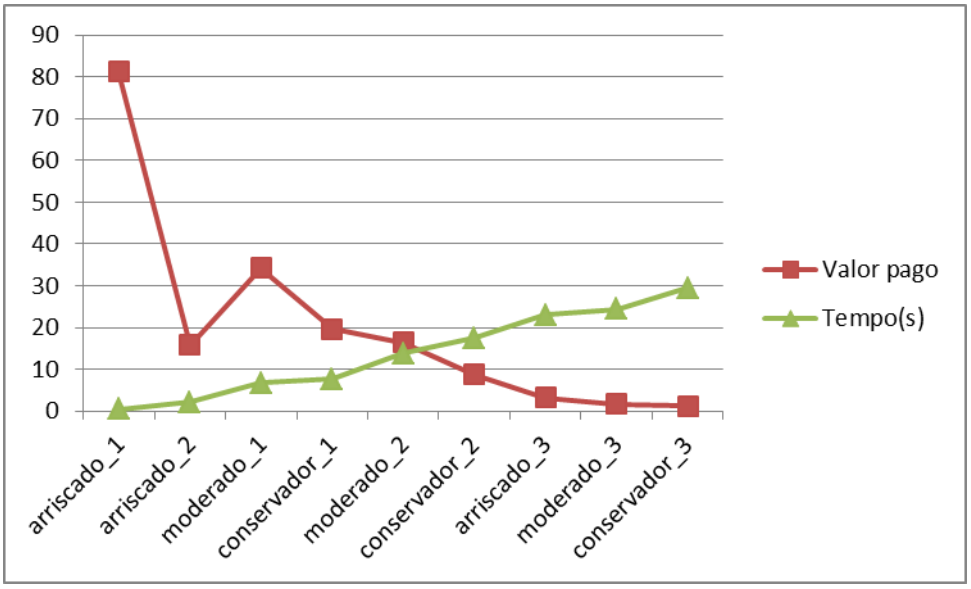
Nome do agente	Créditos(CDT)	Valor pago	Tempo(s)	Ordem de aquisição	Lances	Leilões
arriscado_1	95	81,2	3	1	1	1
arriscado_2	50	15,82	12,24	2	1	2
arriscado_3	5	3,24	138,24	7	7	16
moderado_1	95	34,21	40,32	3	2	5
moderado_2	50	16,42	82,32	5	5	10
moderado_3	5	1,69	146,7	8	1	17
conservador_1	95	19,58	46,2	4	5	6
conservador_2	50	8,66	104,7	6	3	12
conservador_3	5	1,17	176,64	9	3	20
Total		181,99	750,36		28	
Média		20,22	83,37		3,11	9,89

Fonte: Autoria própria

O *driver arriscado_2* chama a atenção na simulação, por não ter chegado tão perto do seu limite de gasto e por ter conseguido a vaga em menos tempo que outros três agentes que pagaram mais do que ele (foi o único agente na simulação que ficou fora da sequência onde quem pagou mais recebeu a vaga antes). Isso aconteceu por ele conseguir ficar com a última vaga do leilão dois, quando outros agentes superaram a sua oferta antes que os agentes *moderado_1, moderado_2* e *conservador_1* pudessem o fazer.

Para ilustrar essa análise, o Gráfico 8 organiza os *drivers* pela ordem que receberam a vaga, mostrando o *arriscado_2* como um ponto fora da curva.

Gráfico 8 – Tempo e valores da simulação leilão inglês 3.1



Fonte: Autoria própria

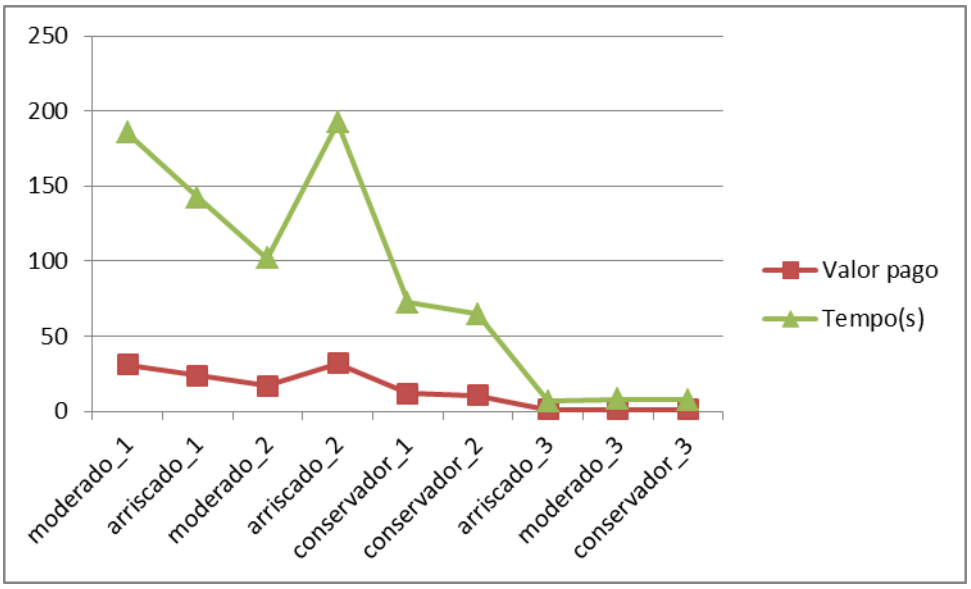
Este comportamento inesperado, motivado pela ordem em que os agentes enviam lances e o tempo até os enviarem, pode fazer com que o ponto fora da curva no valor pago seja para cima.

No Gráfico 9, o qual representa a simulação 3.2 do leilão inglês, está ilustrado esse comportamento onde novamente o agente *arriscado_2* aparece como um ponto fora da curva do valor pago, pagando mais que outros agentes que receberam o recurso anteriormente a ele.

Os valores recebidos pelo *manager* dos nove *drivers* na simulação 3.2 foram consideravelmente menores que os da simulação 3.1 (181,99 contra 130,18), pois o *driver arriscado_1* que pagou 81,2 créditos no primeiro leilão da simulação 3.1 não

conseguiu superar as ofertas do primeiro leilão da simulação 3.2 e acabou conseguindo a vaga no segundo leilão pagando apenas 23,71 créditos por ela.

Gráfico 9 – Tempo e valores da simulação leilão inglês 3.2



Fonte: Autoria própria

A diferença entre o tempo médio nas duas simulações foi de 1,02 segundos, reforçando o que a consistência no processo de alocação das vagas inferido na subseção anterior.

7.3.5 Análise dos resultados Leilão Inglês

Os resultados apresentados nos cenários 0 e 1, mostram que quando o estacionamento possui vagas suficientes para todos os requisitantes, todos agentes enviarão apenas um lance para conseguir o recurso. O importante nesses casos é ser rápido em enviar a oferta, recebendo junto com os demais, pagando menos por isso. Fazer os lances primeiro nesses cenários é mais importante do que possuir muitos créditos ou a estratégia do incremento dos lances.

Nos cenários 3 e 4 onde o recurso é escasso, e os agentes muitas vezes enviam mais de um lance para consegui-lo, três fatores são relevantes a negociação:

- O tempo de envio dos lances;
- O fator de incremento:
- O valor de créditos disponível para a aquisição do recurso.

O tempo de envio dos lances é importante pela necessidade de superar outros lances vencedores antes que o tempo determinado pelo *driver* para o leilão se esgote, como na simulação 2.1 o quinto leilão ser vencido pelo *moderado_2* se houvesse tempo para ele cobrir o lance, mas quem venceu foi o *conservador_2*.

O fator de incremento tem papel importante novamente no comportamento dos agentes. No caso dos agentes arriscados o alto fator (50%) pode rapidamente elevar os preços fazendo que agentes menos dispostos se desinteressem pela disputa, mas também podem atrasar a aquisição quando possuem créditos disponíveis para superar a oferta, mas o fator faz com que não seja possível por a o valor da oferta a ser feita ultrapassar o valor disponível, assim acabam enviando poucos lances.

O contrário acontece aos conservadores, que superam por pouco a oferta vencedora abrindo espaço para serem superados antes que possam recobrir o lance vencedor, porém podem chegar mais próximos ao seu limite. Exemplo disso ocorreu na simulação 2.1 onde o *driver* arriscado ofertou 12 lances e recebeu a vaga no quarto leilão, enquanto o *conservador_2* enviou 44 lances adquirindo o recurso no leilão seguinte.

Por fim os créditos disponíveis, que aparentam ser o critério mais importante visto que em três das quatro simulações dos dois últimos cenários o primeiro a receber a vaga foi o *arriscado_1*. A disponibilidade desse agente por ser pelo menos o dobro maior que dos outros analisados elevou bastante a média recebida, mas observou-se que de maneira geral nas simulações apresentadas a disponibilidade dos agentes foi explorada neste mecanismo de negociação.

A alocação das vagas no mecanismo mostrou-se consistente. Isso é confirmado pela proximidade no tempo médio para os agentes receberem o recurso entre as simulações do mesmo cenário. Este fato também é reforçado quando comparamos entre os dois cenários, onde a média das duas simulações do cenário 3, 82,8 segundos, é aproximadamente o dobro da média das simulações do cenário 2, 40,8 segundos, (cenário 3 a proporção é de o dobro de agentes por vaga em relação ao cenário 2).

7.4 RESULTADOS LEILÃO HOLANDÊS

O leilão holandês funciona de maneira decrescente. O *manager* começa oferecendo aos *drivers* requisitantes a vaga por 100 créditos e a cada nova iteração ele abaixa o preço em 20% até conseguir alocar vagas a todos os requisitantes. Se durante uma oferta, mais agentes aceitarem aquela oferta do que existem vagas disponíveis, o agente repetirá aquele valor na oferta seguinte.

Para esse mecanismo de negociação os *drivers* não possuem um fator de incremento porque não são eles que enviam os valores das propostas. A mensagem que eles enviam para o *manager* é para informar que aceitam pagar o valor que o *manager* propõe pela vaga quando a proposta está dentro da disposição em pagar do *driver.*

7.4.1 Leilão Holandês Cenários 0 e 1

A aplicação do leilão holandês para os cenários 0 e 1, obteve resultados iguais em relação aos valores pagos e muito próximos em relação ao tempo de alocação, como podemos observas nas Tabelas 13 e 14.

Tabela 13 – Resultados simulação leilão holandês 0

Nome do agente	Créditos	Valor pago	Tempo(s)	Propostas aceitas
arriscado_1	95	80	0,96	1
arriscado_2	50	40,96	13,2	1
arriscado_3	5	4,4	53,58	1
moderado_1	95	40,96	13,14	1
moderado_2	50	20,97	25,38	1
moderado_3	5	2,25	65,7	1
conservador_1	95	20,97	25,38	1
conservador_2	50	10,73	37,44	1
conservador_3	5	1,15	77,7	1
Total		222,39	312,48	9
Média		24,71	34,72	1

Fonte: Autoria própria

Os resultados são praticamente os mesmos pelo fato de o *manager* abaixar sempre em 20% a cada nova oferta. Assim como a disposição dos *drivers* e o seu

valor de créditos são estáticos nas simulações, os valores pagos pela vaga também são mantidos.

Tabela 14 – Resultados simulação leilão holandês 1

Nome do agente	Créditos	Valor pago	Tempo(s)	Propostas aceitas
arriscado_1	95	80	0,96	1
arriscado_2	50	40,96	13,08	1
arriscado_3	5	4,4	53,28	1
moderado_1	95	40,96	13,08	1
moderado_2	50	20,97	25,14	1
moderado_3	5	2,25	65,34	1
conservador_1	95	20,97	25,2	1
conservador_2	50	10,73	37,32	1
conservador_3	5	1,15	77,4	1
Total		222,39	310,8	9
Média		24,71	34,53	1

Fonte: Autoria própria

O tempo que leva para os *drivers* receberem a vaga, por existirem vagas sobrando é alto. Isso acontece pelo tempo utilizado pelo *manager* para fazer as operações de oferta, esperar por respostas, alocar possíveis vagas e posteriormente voltar a repetir o ciclo.

A disposição dos *drivers* em pagar foi muito explorada, mesmo sem existir escassez de vagas, a receita total foi de 222,39 créditos. O envio de mensagens pelos *drivers* foi o menor possível, apenas uma para aceitar quando o valor estava dentro da possibilidade.

7.4.2 Leilão Holandês Cenário 2

No cenário onde existe apenas uma vaga para os nove agentes, os valores pagos nos cenários são os mesmo, porém o tempo de espera aumenta considerávelmente. A Tabela 15 apresenta os resultados da simulação do leilão holandês no cenário dois.

Os agentes *arriscado_2* e o *conservador_1* precisam enviar duas vezes a mensagem de confirmação pelo valor da vaga. Isso acontece pois outro agente aceitou pagar o mesmo valor e acabou levando a vaga na primeira tentativa. Então o *manager* como percebe que existem *drivers* que não receberam o recurso, mas

estão disponíveis a pagar aquele valor, repete a oferta quando existirem vagas disponíveis.

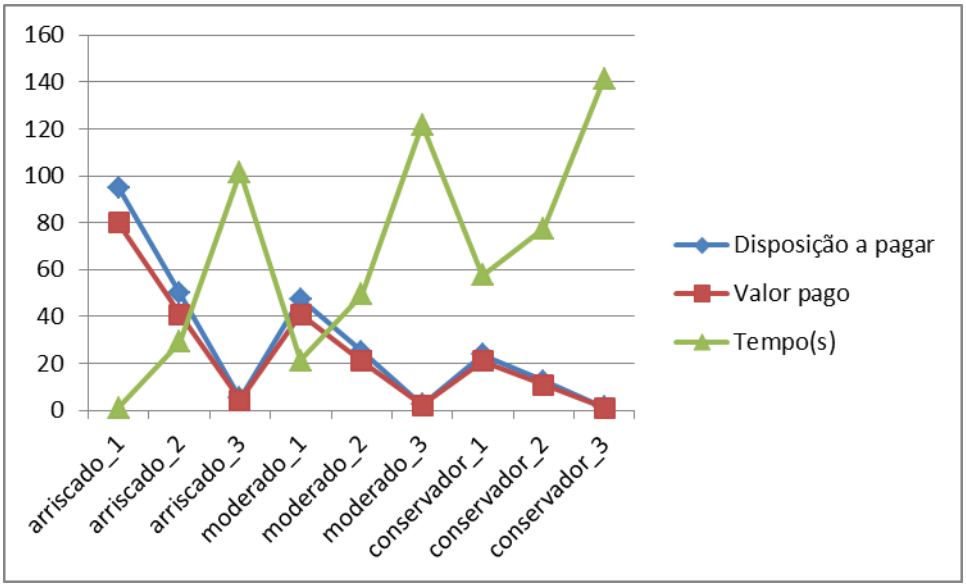
Tabela 15 – Resultados simulação leilão holandês 2

Nome do agente	Créditos	Valor pago	Tempo(s)	Propostas aceitas
arriscado_1	95	80	0,96	1
arriscado_2	50	40,96	29,28	2
arriscado_3	5	4,4	101,58	1
moderado_1	95	40,96	21,18	1
moderado_2	50	20,97	49,38	1
moderado_3	5	2,25	121,68	1
conservador_1	95	20,97	57,48	2
conservador_2	50	10,73	77,52	1
conservador_3	5	1,15	141,42	1
Total		222,39	66,72	11
Média		24,71	600,48	1,22

Fonte: Autoria própria

O Gráfico 10 ilustra de maneira clara a exploração da disposição dos *drivers* a pagarem pelo recurso, e a diferença paga pelo agente *arriscado_1* e o *conservador_3*, onde o que pagou mais, desembolsou quase setenta vezes o valor desembolsado pelo último a receber a mesma vaga.

Gráfico 10 – Tempo, valor e disposição da simulação leilão holandês 2

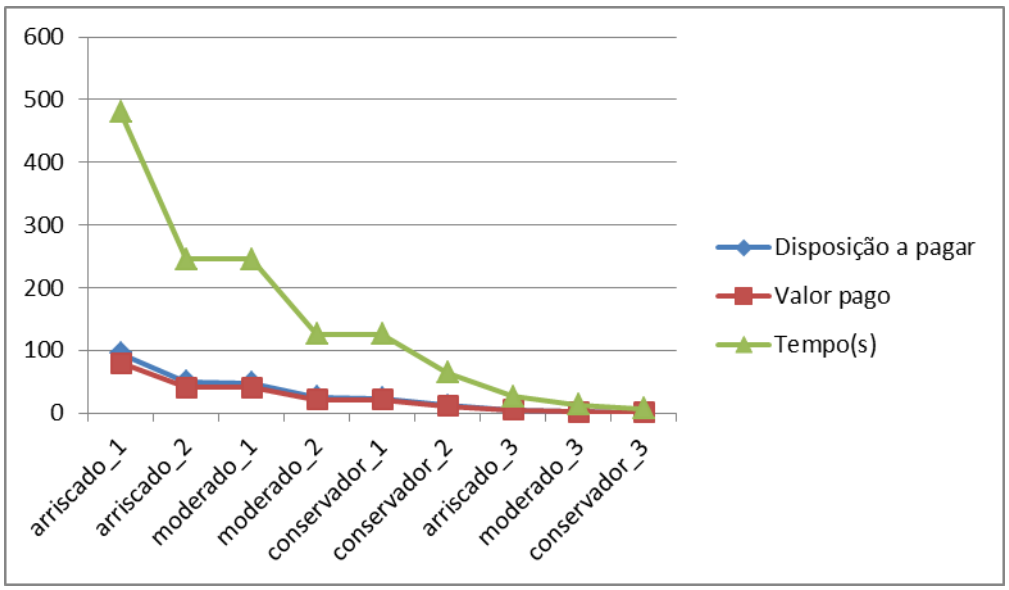


Fonte: Autoria própria

7.4.3 Leilão Holandês Cenário 3

A simulação do cenário 3 no leilão holandês manteve o padrão dos outros cenários. O Gráfico 11 ilustra os valores da simulação, com os agentes ordenados conforme receberam o recurso.

Gráfico 11 – Tempo, valor e disposição da simulação leilão holandês 3



Fonte: Autoria própria

Neste cenário a proporção de *drivers* por vagas é o dobro da proporção. Porém o tempo médio para que os nove *drivers* levaram para conseguir as vagas foi apenas 15,42 segundos maior, 17% do tempo médio do segundo cenário.

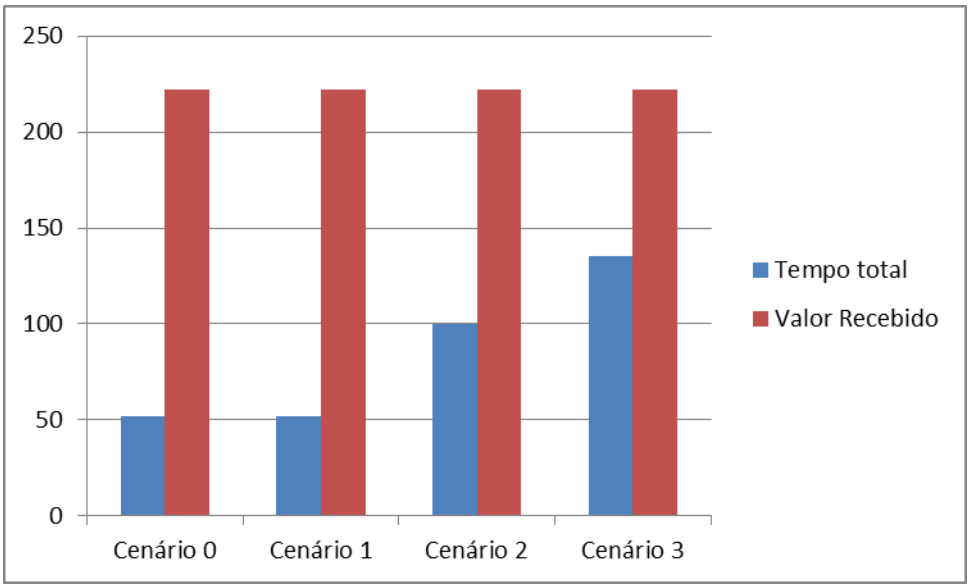
7.4.4 Análise dos resultados Leilão Holandês

Como já citado na apresentação dos resultados, este mecanismo consegue explorar a disposição em pagar de um *driver*, fazendo com que o *Smart Parking* possua um alto valor de lucro.

Neste mecanismo de negociação não há um grande número de envio de mensagens dos *drivers* ao *manager*. Os agentes *drivers* só enviam mensagens para informar o *manager* quando aceitam pagar o valor.

O Gráfico 12 apresenta alguns dados dos cenários aplicados ao mecanismo leilão holandês.

Gráfico 12 – Tempo e valor dos cenários do leilão holandês



Fonte: Autoria própria

Observa-se que independente da disputa de vagas o valor pago por cada agente é estático, dependendo apenas da sua disposição a pagar pelo recurso. Já o tempo, que é alto para quando existem vagas a mais que requisitantes, não aumenta proporcionalmente a quando a disputa por vagas cresce, isso mostra a importância da eficiência do *manager* ao rodar a negociação.

7.5 COMPARATIVO ENTRE AS ABORDAGENS

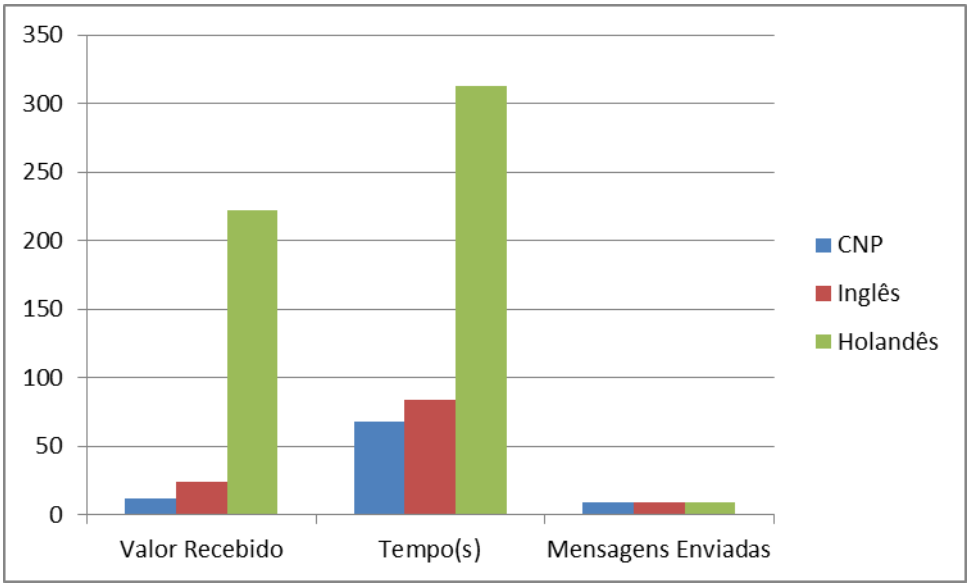
Esta subseção apresenta um comparativo entre os resultados dos três mecanismos implementados. São quatro subseções onde são comparados os resultados para cada cenário, em relação às mensagens enviadas pelos *drivers*, o valor recebido pelo *manager* e o tempo que levou para os agentes *drivers* receberem o recurso. Por fim uma quinta subseção onde apresentaremos uma análise geral sobre a comparação entre as abordagens.

7.5.1 Comparativo dos cenários 0

Com mais vagas disponíveis do que agentes *drivers* necessitando de vagas o leilão holandês é uma solução muito cara para os *drivers* e leva bastante tempo a mais se comparada às outras duas abordagens para alocar a vaga a todos os

agentes. O único parâmetro igualado é o das mensagens que o *manager* recebe dos *drivers* até alocar vagas a todos, nove mensagens. O Gráfico 13 ilustra a disparidade entre o leilão holandês e as outras duas abordagens neste cenário.

Gráfico 13 – Comparativo dos Cenários 0



Fonte: Autoria própria

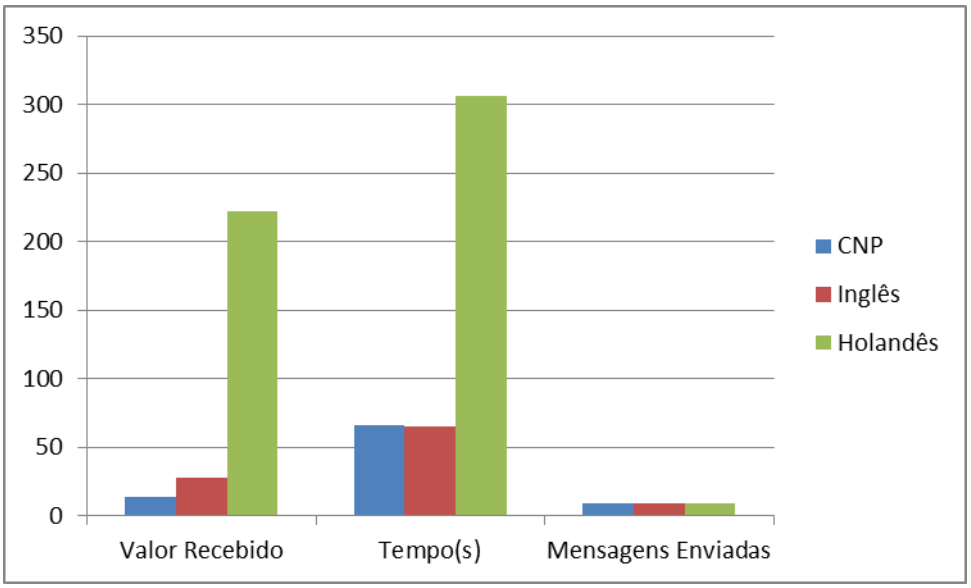
O leilão inglês busca explorar uma competição entre os agentes mesmo que não exista a necessidade, sendo que o recurso não é escasso. Utilizando este leilão, o lucro do *Smart Parking* foi maior que o dobro do lucro recebido pelo *Contract Net*, e o sistema levou pouco mais de 16 segundos a mais para alocar todas as vagas. Mas ele acaba punindo agentes que possuem poucos créditos e que não foram capazes de dar o seu lance antes dos mais dispostos a gastar, fazendo que eles esperem mesmo existindo vagas disponíveis.

O *Contract Net* aparece como a melhor solução para os *drivers* neste cenário. Todos recebem a vaga quase que ao mesmo tempo, na mesma rodada da negociação, e pagam o valor inicial randômico, o mínimo, pois não há competição.

7.5.2 Comparativo dos cenários 1

Em todas as abordagens os resultados para o estacionamento com vagas disponíveis a mais que requisitantes, e com o número de vagas disponíveis igual ao número de requisitantes tiveram resultados muito semelhantes. O Gráfico 14 ilustra esse fato.

Gráfico 14 – Comparativo dos Cenários 1



Fonte: Autoria própria

O leilão holandês novamente aparece com resultados distantes dos resultados apresentados pelas outras duas abordagens em relação a valor recebido e o tempo para que todos os agentes recebam as vagas. Vale ressaltar que isso acontece por causa dos valores fixos durante o leilão, que não é alterado em relação à lotação do *Smart Parking.*

7.5.3 Comparativo Cenário 2

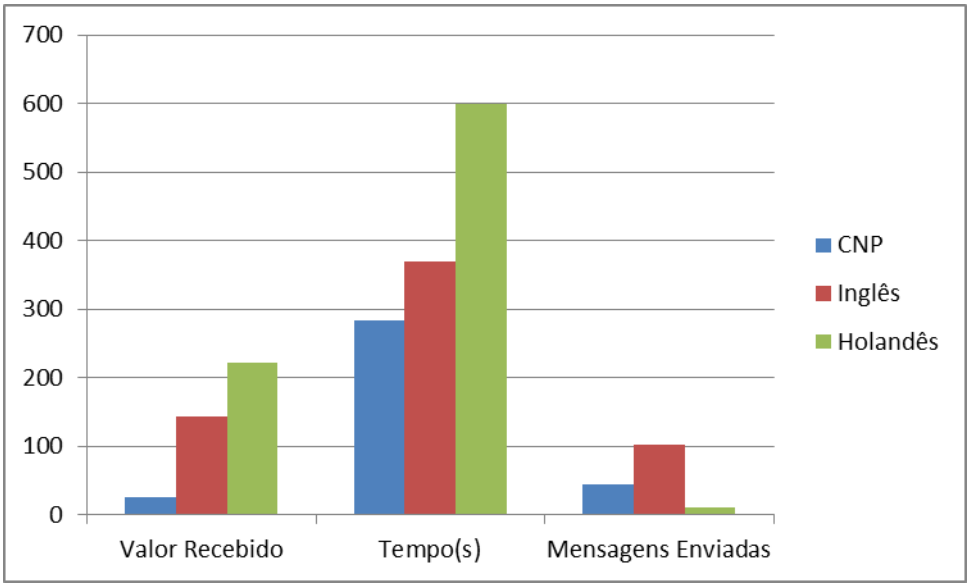
Com vagas escassas o *Contract Net Protocol* é a abordagem que menos consegue explorar a disposição em pagar dos *drivers*, dobrando o seu lucro com relação ao estacionamento com vagas de sobra. Já o tempo que o *manager* levou para alocar as vagas a todos os *drivers* chegou a ser quatro vezes maior comparado aos cenários anteriores, assim como o número de mensagens recebidas pelo *manager* subiu de nove nos cenários anteriores para quarenta e cinco nessa.

O leilão inglês se aproximou ao leilão holandês comparado aos cenários anteriores, mas ainda teve uma receita 79,68 créditos menor. Também se aproximou ao holandês no tempo que levou para alocar todas as vagas, enquanto o holandês praticamente dobrou o tempo com relação ao cenário 1, o tempo que o inglês levou foi seis vezes maior que no cenário anterior.

O holandês seguiu recebendo poucas mensagens nesse cenário pois nesse mecanismo o agente licitante não é muito interativo, apenas responde quando o

valor lhe aprouver. Em comparação o leilão inglês onde os *drivers* interagem entre si, superando uns as ofertas dos outros, o número de mensagens enviadas pelos *drivers* passou a ser mais de dez vezes maior que no cenário anterior.

Gráfico 15 – Comparativo dos Cenários 2



Fonte: Autoria própria

Os resultados do leilão inglês nesse caso ficam no meio termo para *Smart Parking* e *drivers.* O estacionamento consegue ter uma receita que passa a explorar a disponibilidade em pagar pelas vagas do *driver*. Enquanto aos *drivers* que conseguem a vaga por um preço não tão alto como o do leilão holandês, e esperam menos tempo pelo recurso. Porém para que os *drivers* consigam a vaga, devem ser muito participativos durante a negociação.

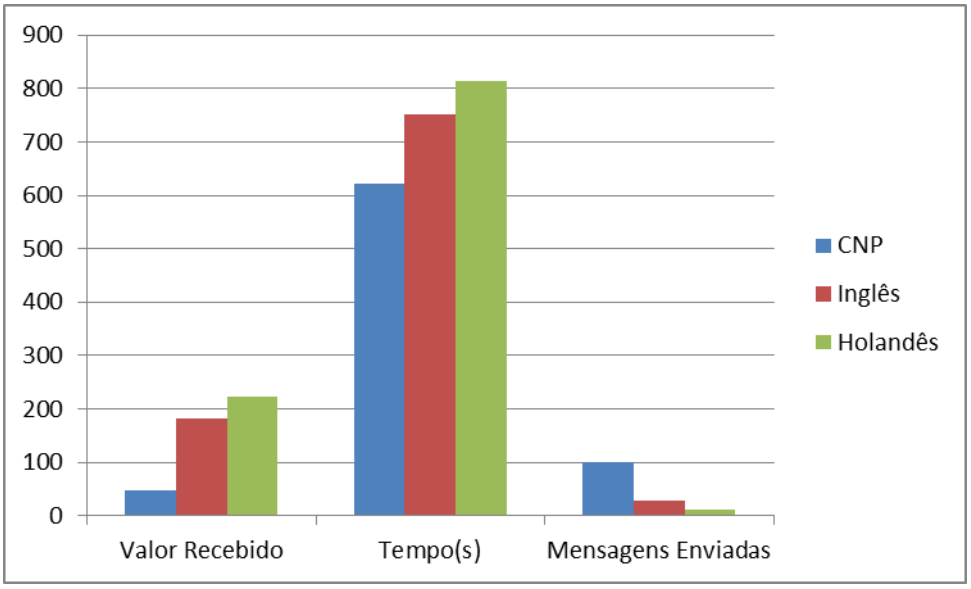
7.5.4 Comparativo dos cenários 3

No último cenário existe uma disputa ainda maior pelas vagas, onde noventa agentes disputam apenas cinco vagas. O Gráfico 16 ilustra os valores agregados pelos nove agentes *drivers* conhecidos utilizados nas simulações.

Realizando as simulações o CNP confirmou-se como o melhor modelo para os *drivers*. O valor total pago pelos nove *drivers* para conseguirem o recurso não chegou a 50 créditos, valor facilmente superado pelo agente *arriscado_1* nos outros mecanismos de negociação. Além disso, o CNP mais uma vez foi o mecanismo que levou menos tempo para conseguir alocar todas as vagas requisitadas. O único

inconveniente é o número de mensagens que passa a aumentar quanto mais aumenta a concorrência.

Gráfico 16 – Comparativo dos Cenários 3



Fonte: Autoria própria

O leilão holandês consegue extrair bastante créditos dos *drivers* e o *manager* não precisa manipular um grande número de mensagens para conseguir concluir a negociação. Sendo assim um bom modelo de negociação na visão do sistema.

Por fim o leilão inglês, num cenário de grande disputa se aproxima bastante dos valores recebidos no leilão holandês e também no tempo que demorou para todos os agentes *drivers* conseguirem estacionar. O número de lances enviados caiu pois com mais agentes envolvidos na negociação, os valores máximos são atingidos antes que novos lances sejam feitos.

7.6 CONSIDERAÇÕES FINAIS

Este Capítulo apresentou os cenários de testes utilizados para analisar os resultados da implementação dos mecanismos de negociação. À partir desses resultados pode-se desenvolver o Capítulo 8, o qual apresenta a conclusão deste trabalho.

8 CONCLUSÃO

É evidente a necessidade de projetos no desenvolvimento das cidades inteligentes a fim de melhorar a qualidade de vida e auxiliar no desenvolvimento sustentável dos ambientes urbanos. Uma das questões críticas nesse assunto é o desenvolvimento dos estacionamentos inteligentes os quais buscam alternativas para automatizar o gerenciamento e organização dos recursos de um estacionamento.

O projeto MAPS almeja encontrar soluções nessa área utilizando SMA, focado em um estacionamento fechado. Para contribuir para que essas soluções sejam encontradas, o trabalho desenvolvido apresentou mecanismos de negociação utilizados em SMA. Os mecanismos foram desenvolvidos para verificar o funcionamento e os resultados das negociações do SMA. Além disso foram realizados testes com diferentes configurações apresentados no Capítulo 7.

Utilizando funções da linguagem Jason para a programação de agentes no modelo BDI, e das funções para a negociação foram alcançados resultados interessantes, não só para o contexto do projeto MAPS, mas de uma maneira geral para a alocação de recursos em SMA.

O mecanismo de negociação *Contract Net Protocol* foi apresentado como um modelo onde é possível, a partir de uma estratégia de incremento consistente, que o recurso seja alocado sem que o custo ao requisitante seja muito elevado nem o tempo para que receba o recurso seja alto.

Se o interesse maior for o lucro para quem fornece o recurso, o leilão holandês entre as abordagens testadas foi o que obteve mais sucesso. O problema é que, além de ser um modelo caro para o consumidor, o mecanismo também mostrou-se como o mais lento para o recurso ser alocado nos testes realizados. Também é interessante ao vendedor por necessitar de uma menor troca de mensagens para a alocação dos recursos.

Por fim o leilão inglês se apresentou como o modelo intermediário entre os outros dois. Neste mecanismo, os valores sobem bastante quando a concorrência é alta, e caem quando não existe grande concorrência. Além disso, o mecanismo é bastante interessante ao conceito dos SMA devido à grande interação entre os agentes que competem entre si pelo recurso. Como desvantagem, a troca de

mensagens é muito elevada para realizar as negociações utilizando essa abordagem.

8.1 TRABALHOS FUTUROS

Espera-se, como trabalhos futuros, o desenvolvimento de um mecanismo de negociação híbrido, que una os três mecanismos de negociação explorando em quais cenários cada um tem mais vantagens.

Outros pontos a serem trabalhados são o estudo e aprimoramento a respeito das estratégias de negociação usadas por agentes, bem como a negociação das vagas de maneira descentralizada, retirando a necessidade da utilização do agente *manager*, e dessa forma implantar mecanismos de coordenação e cooperação entre os agentes.

8.2 PUBLICAÇÕES

Este trabalho teve inicio com o desenvolvimento do artigo “Análise de modelos de confiança e reputação em sistemas baseados em agentes para alocação de vagas em um estacionamento inteligente” (MARINI; ALVES, 2016), o qual foi apresentado no *Workshop* de Pesquisa em Computação dos Campos Gerais. Nele foram apresentados modelos de reputação e confiança para SMA que podem ser utilizados em trabalhos futuros como parâmetro para a negociação de recursos, e uma breve análise dos cenários em que cada modelo pode ser melhor aplicado.

REFERÊNCIAS

BANASZEWSKI, Roni F. **Modelo Multiagentes Baseago em um Protocolo de Leilões Simultâneos Para Aplicação no Problema de Planejamento de Transferências de Produtos no Segmento *Downstream* do Sistema Logístico Brasileiro de Petróleo**. 2014. 325f. Tese (Doutorado em Engenharia da Computação) – Universidade Tecnológica Federal do Paraná, Curitiba, 2014.

BASTOS, Ricardo M. **O planejamento de alocação de recursos baseado em sistemas multiagentes**. 1998. 267 f. Tese (Doutorado em Ciência da Computação) – Universidade Federal do Rio Grande do Sul, Porto Alegre, 1998.

BORDINI, Rafael H.; HüBNER, Jomi Fred; WOOLDRIDGE, Michael. **Programming Multi-Agent Systems in AgentSpeak Using Jason (Wiley Series in AgentTechnology).** [S.l.]: John Wiley & Sons, 2007. ISBN 0470029005.

BRATMAN, Michael E.; ISRAEL, David J.; POLLACK, Marta E. Plans and resource-bounded practical reasoning. **Computational intelligence**, Wiley Online Library, v. 4, n. 3, p. 349–355, 1988.

CARTAGO, 2006. Disponível em: <http://cartago.sourceforge.net/>. Acesso em: 23 jun. 2018.

CASTRO, Lucas F. S. de., ALVES, Gleifer V., BORGES, André P. A proposal of an architecture for a Smart Parking based on intelligent agents and embedded systems. Em: **Workshop de Pesquisas em Computação dos Campos Gerais.** [ISSN: 2526-1371], v.1, 2016.

CASTRO, Lucas F. S. de., ALVES, Gleifer V., BORGES, André P., Using trust degree for agents in order to assign spots in a Smart Parking. Em: **Advances in Distributed Computing and Artificial Inteligence Journal – ADCAIJ.** [SI: s.n.], (Aguardando Publicação), 2017.

DA SILVA, Juliana. **Aplicativo de vagas pode reduzir trânsito nas cidades e facilitar sua vida,** 2015. Disponível em: <http://www.infomoney.com.br/negocios/startups/noticia/3853310/aplicativo-vagas-pode-reduzir-transito-nas-cidades-facilitar-sua-vida>. Acesso em: 23 jun. 2018.

FARATIN, Peyman; SIERRA, Carles; JENNINGS, Nick. **Negotiation Decision Functions for Autonomous Agents**, Londres, Inglaterra, 1998.

FERBER, Jacques. **Multi-Agent Systems. An Introduction to Distributed Artificial Intelligence**, Addison Wesley, Boston, 1999.

FIPA, **FIPA Contract Net Interaction Protocol Specification**, 2002. Disponível em: <http://www.fipa.org/specs/fipa00029/SC00029H.pdf >. Acesso em: 23 jun. 2018.

FULLAN, Karen; BARBER, Suzanne. Dynamically Learning Sources of Trust Information: Experience vs. Reputation. **Proceedings of the 6th International Conference on Autonomous Agents and Multiagent Systems**, volume 5, pp. 1055–1062, 2007.

GONÇALVES, Wesley R. C.; ALVES, Gleifer V. Análise de modelos de confiança e reputação em sistemas baseados em agentes para alocação de vagas em um estacionamento inteligente. Em: **Anais do IX Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações – WESAAC**. [S.l.: s.n.], 2015.

GRANATYR, Jones.; SCALABRIN, Edson.; LESSING, Otto.; BOTELHO, Vanderson.; BARTHES, Jean-Paul.; ENEMBRECK, Fabrício. **Trust and Reputation Models for Multiagent Systems.** ACM Comput. Surv. 48, 2, Article 27, 2015.

HUYNH, Trung D; JENNINGS, Nicholas R.; SHADBOLT, Nigel R. **An integrated trust and reputation model for open multi-agent systems**. Springer Science, LLC 2006.

JACAMO. **The JaCaMo approach,** 2011. Internet. Disponível em: <http://jacamo.sourceforge.net/?page_id=40>. Acesso em: 23 jun. 2018.

JENNINGS, Nicholas R. **On agente-based software engineering**. Southampton, Reino Unido, 2000.

JENNINGS, Nicholas .R.; PARSONS, Simons.; NORIEGA, Pablo.; SIERRA, Carles. **On agente-based software engineering**. in: Proceedings of the International Workshop on Multi-Agent Systems, Boston, Estados Unidos da América, 1998.

LEITE, Carlos. Inteligência Territorial: Cidades Inteligentes com Urbanidade. Em: **Cadernos FGV Projetos – Cidades Inteligentes e Mobilidade Urbana.** Rio de Janeiro v. 9, n.24, p. 46-54, jul. 2014.

MARINI, Angelo B; ALVES, Gleifer V. Smart parking: mecanismo de leilão de vagas de estacionamento usando reputação entre agentes. Em: **Workshop de Pesquisas em Computação dos Campos Gerais.** v.1, 2016.

MOISE, 2006. Disponível em: <http://moise.sourceforge.net/>. Acesso em: 23 jun. 2018.

NAPOLI, Claudia di, NOCERA, Dario Di, ROSSI, Silvia. Using Negotiation for Parking Selection in Smart Cities. Em: **Advances in Practical Applications of Heterogeneous Multi-Agent Systems**. The PAAMS Collection, v. 8473, p. 331–334. Springer International Publishing. ISBN 978-3-319-07550-1. doi:10.1007/ 978-3-319-07551-8_31. 2014.

O'HARE, Greg.; JENNINGS, Nicholas. **Foundations of Distributed Artificial Intelligence**. Nova York, Nova York, Estados Unidos da América, 1996.

REIS, Luís. "**Coordination in Multi-Agent Systems: Applications in University Management and Robotic Soccer**", 2003. 451f. Tese (Doutorado em Engenharia da Computação) – Faculdade de Engenharia da Universidade de Porto, Porto, Portugal, (2003)

RUSSELL, Stuart J.; NORVIG, Peter. **Artificial Intelligence: A Modern Approach**. 2. ed. [S.l.]: Pearson Education, 2003. ISBN 0137903952.

SABATER, Jordi; SIERRA, Carles. Reputation in gregarious societies. In: **Proceedings of the Fifth International Conference on Autonomous Agents**. Nova York, Estados Unidos da América, 2001.

SABATER, Jordi; SIERRA, Carles. **Review on computational trust and reputation models**. Bellaterra, Barcelona, Espanha, IIIA – CSIC, 2003

REZENDE, Solange. **Sistemas Inteligentes: fundamentos e aplicações**. Barueri, São Paulo, 2003. ISBN 85-204-1683-7.

SMITH, Reid G. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. Em: **Readings in Distributed Artificial Intelligence.** San Mateo, p.357-366. 1988.

STIMMEL, Carol. **Building Smart Cities. Analytics, ICT, and Design Thinking.** 1st. ed. [S.I] CRC Press is an imprint of Taylor & Francis Group, an Informa business, 2016. ISBN 13: 978-1-4987-0277-5.

WOOLDRIDGE, Michael. **An Introduction to MultiAgent Systems**. 2nd. ed. [S.l.]: Wiley Publishing, 2009. ISBN 0470519460, 9780470519462.