Implementação de Modelo de Raciocínio e Protocolo de Negociação para um Estacionamento Inteligente com Mecanismo de Negociação Descentralizado

Felipe Felix Ducheiko[1], André Pinz Borges[2] e Gleifer Vaz Alves[2]

***Resumo*—Em grandes cidades normalmente um motorista gasta uma considerável parcela de tempo ou percorre longas distâncias para encontrar uma vaga de estacionamento livre, o que gera desperdício de combustível e ocasiona engarrafamentos. Com o objetivo de auxiliar em questões de mobilidade urbana é idealizado o projeto MAPS (MultiAgent Parking System), o qual possui um *smart parking* (estacionamento inteligente) que utiliza técnicas de sistema multiagente para alocação de vagas de estacionamento e funciona baseado em um mecanismo de negociação centralizado. Este trabalho apresenta uma extensão do projeto MAPS por meio da criação de um modelo de raciocínio e protocolo de negociação para implementar um mecanismo de negociação de vagas descentralizado. O artigo descreve as fórmulas e o protocolo elaborados para a negociação entre os agentes, a implementação do Sistema Multiagente utilizando o *framework* JaCaMo, onde destaca-se a organização social do sistema criada por meio de esquemas e regras do *Moise*.**

***Palavras-chaves*—Cidade Inteligente, Estacionamento Inteligente, Sistemas Multiagentes, Modelo de Raciocínio, Protocolo de Negociação.**

I. Introdução

O conceito *Smart City* surgiu durante a última década com a fusão de várias ideias, onde a essência do conceito é integrar as tecnologias que até agora têm sido desenvolvidas separadamente, mas que tem ligações claras em seu funcionamento e podem ser desenvolvidas de forma integrada [1]. Dentre os desafios a serem enfrentados pelas cidades destacam-se os de mobilidade urbana. Estima-se que em Nova York 40% dos congestionamentos são ocasionados por motoristas buscando vagas de estacionamento [2].

Quando se percebe que a demanda de vagas de estacionamentos não está sendo satisfeita a solução adotada normalmente é um aumento quantitativo do número de vagas. Porém, a utilização das mesmas vagas de modo mais inteligente pode amenizar ou até solucionar o problema. *Smart Parkings* são sistemas compostos por dispositivos de *hardware*, capazes de detectar o nível de ocupação do estacionamento e *softwares* integrados, para gerir a atribuição desses espaços de estacionamento [3]. Tais sistemas são concebidos para auxiliar os motoristas na localização de vagas disponíveis, colaborando com a solução de problemas relacionados à mobilidade urbana. Dentre os vários modelos computacionais que podem ser utilizados para implementar um *Smart Parking* destacam-se os Sistemas Multiagentes (SMA).

Segundo [4] SMA são sistemas compostos de vários elementos computacionais que realizam interações ente si, de modo a atingirem seus objetivos, sendo tais elementos conhecidos como agentes. Esses agentes possuem duas características importantes: primeiramente são capazes de ações autônomas e em segundo lugar têm a capacidade de interagir uns com os outros pela interação análoga às interações sociais humanas [4]. SMA tem raízes na Inteligência Artificia e possuem características excepcionais para simular e testar cenários para apoiar a tomada de decisão [5].

Por meio das habilidades sociais agentes devem ser capazes de negociar uns com os outros, afim de solucionarem problemas de forma distribuída, como ocorre em sociedades. Negociação é um processo complexo de tomada de decisão em que cada parte representa de forma autônoma seus pontos de vista e interage com as outras para resolver conflitos e chegar a acordos, maximizando os ganhos de todas as partes [6].

Para desenvolver um mecanismo de negociação é possível utilizar três configurações: i) *one-to-one*: onde um agente negocia somente com um agente, neste caso os agentes possuem preferências simétricas; ii) *many-to-one* (centralizado): nesta configuração, um único agente negocia com vários agentes, como acontece em um leilão; e iii) *many-to-many* (descentralizado): neste caso, muitos agentes negociam com muitos agentes simultaneamente, como acontece em um mercado [4].

Com o objetivo de aplicar métodos e técnicas usados em Sistema Multiagente na criação de soluções para alocação de vagas e gerenciamento de um *Smart Parking* foi concebido o projeto MAPS (*MultiAgent Parking System*) [7]. O SMA desenvolvido no MAPS possui um mecanismo de negociação centralizado, onde todos os agentes *drivers* negociam exclusivamente com o agente administrador. Esta abordagem possui vantagens, como o fato dos agentes *drivers* trocarem mensagem apenas com o agente administrador, diminuindo o custo computacional. Mas também possui desvantagens, visto que o agente administrador pode falhar, deste modo comprometendo todo o SMA. Outro ponto negativo é a questão

[1] Aluno de Ciência da Computação da Universidade Tecnológica Federal do Paraná (UTFPR) - Ponta Grossa. E-mail: felipeducheiko@alunos.utfpe.edu.br

[2] Professores do Departamento Acadêmico de Informáica da Universidade Tecnológica Federal do Paraná (UTFPR) - Ponta Grossa. E-mail: gleifer@utfpr.edu.br, apborges@utfpr.edu.br

da autonomia dos agentes, visto que os agentes *drivers* em um sistema com mecanismo de negociação centralizado ficam sujeitos as imposições do agente administrador.

Para o desenvolvimento de agentes autônomos com capacidades sofisticadas e flexíveis de negociação três itens devem ser definidos: i) protocolo de negociação, o qual define as ações que os agentes podem tomar em uma negociação; ii) o problema que a negociação quer solucionar; e iii) modelo de raciocínio, o qual define as ofertas iniciais, a gama de ofertas aceitáveis, as contraofertas, quando a negociação deve ser abandonada e quando um acordo deve ser fechado [8]. O objetivo do presente trabalho é estender o projeto MAPS, propondo um modelo de raciocínio e o protocolo de negociação que servirá como base para implementar um SMA para alocação de vagas de estacionamento com um mecanismo de negociação descentralizado, onde não existe a figura de um agente centralizador que administra o sistema e os agentes *drivers* negociam as vagas entre si. Neste artigo é apresentada a implementação das fórmulas e protocolo de negociação encapsuladas em um SMA, que possui diferentes agentes os quais possuem o objetivo de comprar e/ou vender vagas num estacionamento. Este SMA é implementado utilizando o *framework* JaCaMo.

O restante do artigo está organizado da seguinte maneira: na Seção 2 aborda o Projeto MAPS e a versão atual do sistema. Na Seção 3 é apresentado o modelo de raciocínio e na seção 4 o protocolo de negociação. A Seção 5 apresenta a implementação do SMA por meio do *framework* JaCaMo e na Seção 6 encontram-se as considerações finais.

II. Projeto MAPS

O projeto MAPS é desenvolvido no GPAS (Grupo de Pesquisa em Agentes de Software - UTFPR - PG) com a meta principal de elaborar soluções para Estacionamentos Inteligentes. O projeto utiliza o *framework* JaCaMo para o desenvolvimento do SMA [9].

O JaCaMo é um *framework* especifico para a programação de SMA baseados na arquitetura BDI. Este framework é composto de três plataformas independentes: (i) *Jason*: utilizado para programação dos agentes autônomos com arquiterura BDI; (ii) *Cartago*: utilizado para programação do ambiente compartilhado-distribuído baseado em artefatos, onde os agentes estão situados; e (iii) *Moise*: utilizado para programação da organização do SMA, onde as regras sociais são definidas [9].

Para implementar a versão atual do projeto foram definidos dois tipos de agentes (implementados em *Jason*): os agentes *drivers* que interagem e utilizam o Sistema Multiagente e o agente *manager* que é responsável por informar, alocar e gerenciar as vagas do estacionamento [7].

O sistema também é composto por dois artefatos (implementados no *Cartago*): o artefato *control* que é responsável pelo gerenciamento dos *drivers* na fila e o artefato *gate* que é responsável pela cancela, a qual controla o acesso ao estacionamento.

A alocação de vagas é realizada conforme o grau de confiança (*degree of trust*, ou apenas *trust*) de cada *driver*. O grau de confiança para cada agente *driver* é incrementado a cada momento que o *driver* usa o estacionamento. Logo, aquele *driver* que utiliza o estacionamento com maior frequência, terá uma maior grau e maior prioridade no momento de disputar uma vaga com outros *drivers* que não utilizam o estacionamento com a mesma frequência.

III. Modelo de Raciocínio

A modelagem computacional pode facilitar processos de negociação, computando uma ampla gama de alternativas e examinando seus resultados em busca de tendências [10]. Para modelar computacionalmente problemas utilizando técnicas de negociação em SMA os agentes devem ser capazes de gerar, por meio de táticas, ofertas e contraofertas. Táticas são o conjunto de funções que determinam como calcular o quão valioso é um determinado recurso, por meio de características como preço, volume, duração, qualidade, entre outros e considerando um critério, como tempo, recursos, distância, entre outros [11]. Táticas são geradas por combinações lineares de funções simples, sendo que o conjunto de valores utilizados para avaliar o quão valioso é um recurso é a Imagem da função e o critério utilizado é o Domínio [11]. No decorrer desta seção serão apresentadas algumas funções geradoras de táticas para o contexto de *Smart Parking* com mecanismo de negociação descentralizado.

Este modelo de raciocínio assume a existência de apenas um tipo de agente no SMA, o agente *driver*, o qual pode assumir dois papeis distintos: *seller*, quando o agente está deixando uma vaga e *buyer*, quando o agente está procurando uma vaga. Neste modelo a negociação é iniciada pelo agente *driver seller*, quando ele anuncia para todos os agentes que está deixando uma vaga e deseja vende-la [12].

Os *drivers* que recebem a informação que a vaga está disponível para negociação e estão procurando uma vaga de estacionamento assumem o papel de *buyer*. Considerando que o agente no papel de *buyer* receba esta notificação de vaga disponível, então o agente *buyer* irá fazer uma proposta com base na Função 1 (apresentada abaixo e desenvolvida pelos autores deste artigo) e envia para o *driver* no papel de *seller* que ofertou a vaga. Os *drivers* que recebem a oferta de vaga do *seller* e não estão procurando uma vaga ignoram a mesma. Neste artigo utilizam-se as siglas UMOG e UMEG para representar unidades genéricas.

$$Pc = \lambda - \frac{Dist(PD, PV)}{\alpha} \tag{1}$$

Onde:

- Pc = proposta do agente no papel de *buyer*, em unidade monetária geral (UMOG);
- $Dist()$ = distância entre dois pontos, em unidade de medida geral (UMEG);
- PD = ponto da localização onde o *driver buyer* deseja obter a vaga;

- PV = ponto onde se localiza a vaga que o *driver seller* está deixando;
- λ = valor máximo que um *driver* pode pagar por uma vaga; e
  α = distância máxima entre PD e PV que um *driver buyer* aceita.

A Função 1 gera um valor em UMOGs, valor este que o agente *driver buyer* está disposto a pagar por esta vaga ofertada pelo agente *seller* e varia proporcionalmente a proximidade entre a vaga que está sendo ofertada e o local onde o *driver buyer* deseja estacionar, em outras palavras, quanto mais próxima a vaga é do local onde ele deseja estacionar maior é o valor que ele estará disposto a pagar.

Suponha λ, isto é o valor máximo que um *driver* pode pagar para obter uma vaga, igual a 10 e suponha também α, isto é distância máxima entre PD e PV que um *driver* aceita, igual a 25 (neste caso a distância máxima aceita será 250). Para estes valores um *driver buyer* irá pagar para o *driver seller* λ UMOGs, no caso 10 UMOGs, quando distância entre ponto desejado (PD) e o ponto da vaga (PV) é zero. Ainda para estes valores um *driver* não aceitará uma vaga onde a distância entre PD e PV for maior que 250, pois neste caso o valor que um *buyer* irá pagar pela vaga seria negativo e logicamente o *driver seller* não irá aceitar um valor negativo. Portanto, α limita a distância máxima entre PD e PV. A proposta enviada pelo *driver buyer* para o *driver seller* deve ser analisada, isto acontece por meio da Função 2, que foi desenvolvida com base no trabalho de [11].

$$CPv = \begin{cases} Se \quad Pc \geq M \quad Então \quad ACEITAR \\ Se \quad Pc < M \quad \& \quad PC \geq \frac{M}{2} \quad Então \quad CP(M) \\ Se \quad Pc < \frac{M}{2} \quad Então \quad REJEITAR \end{cases} \tag{2}$$

Onde:

- CPv = contraproposta do *seller*;
- Pc = proposta do agente *buyer*;
- M = Média do Valor de Venda;
- $CP()$ = envia uma contraproposta para o *buyer*.

A Função 2 será utilizada para analisar a proposta do *driver buyer* com base no valor médio das vagas próximas a vaga sendo negociada. O valor M é a média do valor de venda das vagas próximas e será obtido por meio de um histórico contendo as últimas negociações realizadas com sucesso que cada vaga do sistema possuirá registrado. A proposta recebida é aceita e o acordo é fechado se a proposta for maior ou igual ao valor médio das vagas próximas e rejeitado se for menor que a metade do valor médio das vagas próximas, em ambos os casos a negociação entre estes dois agentes acaba. Caso contrário uma contraproposta é gerada com base no valor médio das vagas próximas e enviada para o agente *buyer*. Caso o agente *driver buyer* receba esta contraproposta a Função 3 é utilizada para avaliar a mesma.

$$U = \begin{cases} Se \quad (CPv - Pc) \leq (\delta * Pc) \quad Então \quad ACEITAR \\ Se \quad não \quad REJEITAR \end{cases} \tag{3}$$

Onde:

- U = Ultimato;
- Pc = primeira proposta do *buyer*;
- CPv = contraproposta recebida do *driver seller*;
- δ = variação da faixa de fechamento de acordo ($0 < \delta \leq 1$).

A Função 3 garante um fim a negociação, visto que, ou a proposta é aceita e os agentes entram em um acordo, ou a proposta é rejeitada sem que nenhum acordo seja fechado. A proposta é aceita quando a diferença entre a proposta inicial da negociação (Pc) e a contraproposta recebida (CPv) é inferior a δ %. Este valor de δ pode ser dinamicamente modificado, quanto mais próximo de 1 maior a gama de contraproposta que serão aceitas e mais próximo de 0 menor a gama de contraproposta que serão aceitas.

As negociações entre o agente *seller* podem acontecer simultaneamente com vários agentes *buyer*, visto que é uma negociação descentralizada. O agente *driver seller* irá fechar acordo com o primeiro agente *driver buyer* que lançar uma oferta aceitável pelo modelo de raciocínio. Caso o agente *seller* não consiga fechar um acordo com nenhum *buyer* este aguarda por um determinado período de tempo (o qual deverá ser estabelecido antes de iniciar o sistema) e envia uma nova mensagem ofertando a vaga para todos os *drivers* do sistema, lembrando que se trata de um SMA aberto onde os agentes podem entrar e sair do sistema, portanto possivelmente novos agentes aptos a fechar um acordo devem chegar ao estacionamento desejando estacionar.

Para que a negociação aconteça os agentes *drivers* do SMA precisarão realizar transações de UMOGs, para tanto será necessário utilizar o conceito de carteira. Esta carteira consiste no número de UMOGs que o agente possui para negociar vagas, utilizando o conceito de carteira os *drivers buyer* poderão transferir UMOGs para *seller* para pagar por sua vaga, e, estas UMOGs transferidas poderão ser utilizadas posteriormente para a negociação de novas vagas. Uma porcentagem destas UMOGS irá para o estacionamento outra ficara com o agente *seller*.

IV. Protocolo de Negociação

Um protocolo de negociação é um conjunto de regras que especificam a gama de movimentos legais disponíveis para cada agente em qualquer fase de um processo de negociação [13]. A Figura 1 apresenta o protocolo de negociação que é utilizado para implementar o SMA para alocação de vagas de estacionamento com mecanismo de negociação descentralizado, utilizando como base o modelo de raciocínio apresentado na seção anterior.

Este protocolo define que sempre a negociação deverá ser iniciada pelo agente *driver seller*, informando os outros *drivers* do sistema que possui uma vaga para negociar (na Figura

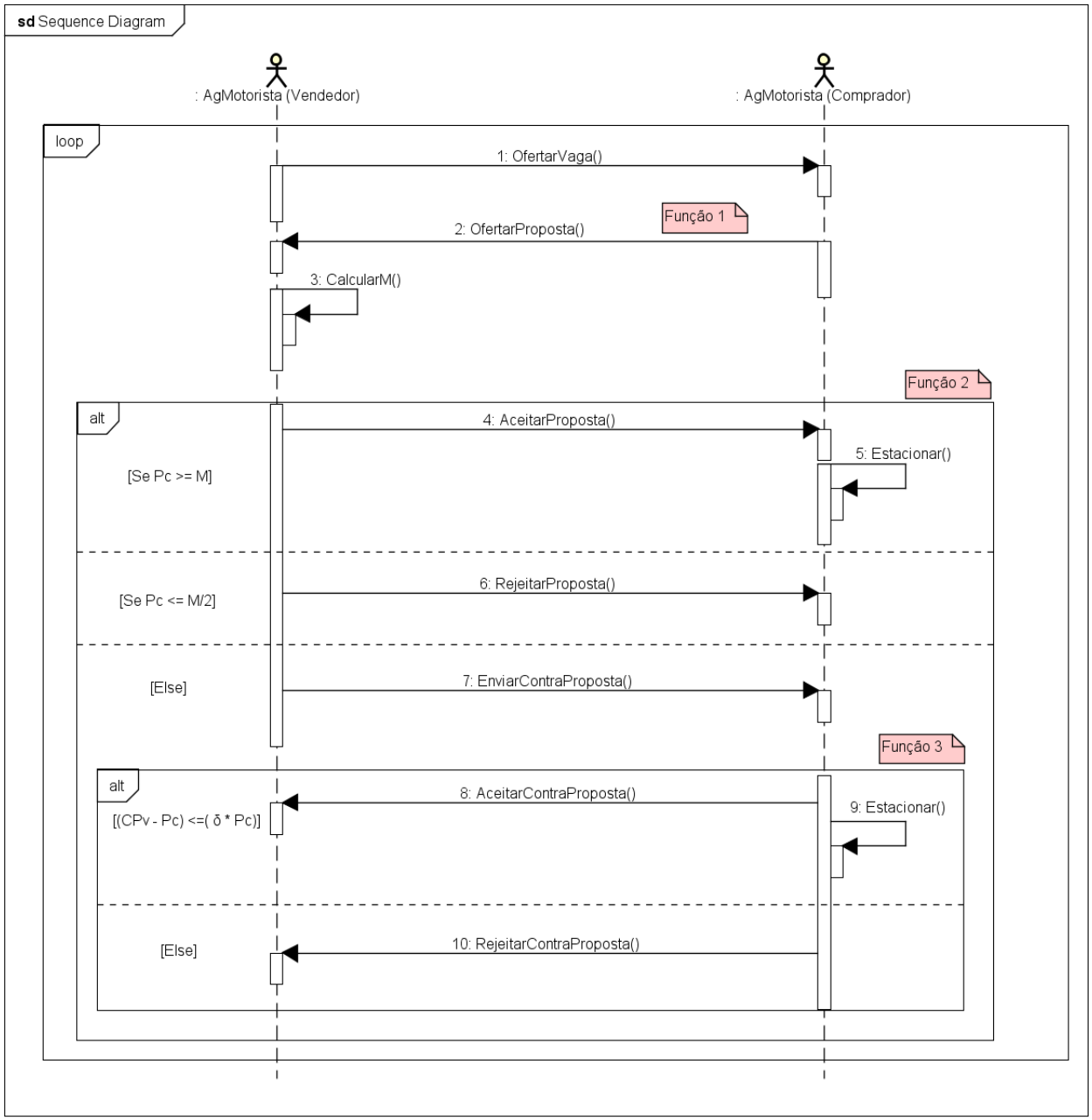

Figura 1. Protocolo de Negociação

1, item 1:OfertarVaga()). Então, os agentes *drivers* que estão procurando uma vaga (*buyer*) geram uma proposta, utilizando a Função 1, e a enviam para o *driver seller* (na Figura 1, item 2:OfertarProposta()). Então, o *driver seller* analisa estas propostas utilizando a Função 2, podendo tomar as seguintes decisões: (i) aceitar proposta (na Figura 1, item 4:AceitarProposta()), onde a negociação acaba e o acordo é fechado; (ii) rejeitar proposta (na Figura 1, item 6:RejeitarProposta()), onde a negociação entre estes dois agentes acaba; ou (iii) enviar contraproposta (na Figura 1, item 7:EnviarContraProposta()), onde o agente *seller* gera uma contraproposta utilizando ainda a Função 2 e envia para o agente *buyer*.

Quando o agente *buyer* recebe esta contraposta ele a analisa utilizando a Função 3 e pode tomar duas ações: (i) aceitar a contraproposta (na Figura 1, item 8:AceitarContraProposta()), onde um acordo entre os dois *drivers* é fechado e a negociação acaba; ou (ii) rejeitar contraproposta (na Figura 1, item 10:RejeitarContraProposta()), onde a negociação entre estes dois agentes acaba e o agente *seller* inicia uma negociação com outro *driver buyer*, isto se repete até que não haja mais

agentes para negociar.

V. Implementação

Para simular o funcionamento do modelo de raciocínio e protocolo de negociação aqui propostos foi implementado um SMA utilizado o *framework* JaCaMo.

Para a implemetação do sistema foram utilizados as três camadas do *framework* JaCaMo. Na camada do *Jason* foram definidos dois tipos de agentes, os agentes *drivers* que utilizam o sistema e negociam vagas entre si e os agentes *parkingSpot-Controller*, os quais possuem um artefato vaga relacionado e controlam o mesmo.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="http://moise.
    sourceforge.net/xml/os.xsl" type="text/
    xsl" ?>

<organisational-specification
id="organization"
os-version="0.8"
xmlns='http://moise.sourceforge.net/os'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-
    instance'
xsi:schemaLocation='http://moise.sourceforge.
    net/os
http://moise.sourceforge.net/xml/os.xsd' >

<structural-specification>

<role-definitions>
<role id="driver" />
<role id="buyer">
<extends role="driver"/>
</role>
<role id="seller">
<extends role="driver"/>
</role>
</role-definitions>

<group-specification id="parkingLot">

<roles>
<role id="buyer"/>
<role id="seller"/>
<role id="parkingSpotController"/>
</roles>

<links>
<link from="buyer" to="seller" type="
    communication"  scope="intra-group" bi-
    dir="true"/>
<link from="seller" to="buyer" type="
    communication"  scope="intra-group" bi-
    dir="true"/>
<link from="buyer" to="parkingSpotController"
     type="authority"  scope="intra-group" bi
    -dir="false"/>
<link from="seller" to="parkingSpotController
    " type="authority"  scope="intra-group"
    bi-dir="false"/>
</links>

<formation-constraints>
<compatibility from="buyer" to="seller" />
<compatibility from="seller" to="buyer" />
<compatibility from="parkingSpotController"
    to="seller" />
<compatibility from="parkingSpotController"
    to="buyer" />
</formation-constraints>
</group-specification>

</structural-specification>

<functional-specification>
<scheme id="parkingLotScheme">

<goal id="ParkingNegotiation">
<plan operator="sequence">

<goal id="arriveParking">
<plan operator="sequence">
<goal id="answerStatusParkingSpot"></goal>
</plan>
</goal>

<goal id="startNegotiation">
<plan operator="sequence">
<goal id="generateOffer"></goal>
<goal id="analyzeOffer"></goal>
<goal id="generateCounterOffer"></goal>
<goal id="ultimatum"></goal>

</plan>
</goal>

<goal id="leaveParking"/>

</plan>
</goal>

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

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

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

</scheme>
</functional-specification>

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

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

<norm id="norm3" type="permission" role="
    seller" mission="departure"/>
</normative-specification>

</organisational-specification>
```

Código 1. Especificação Organizacional *Moise*

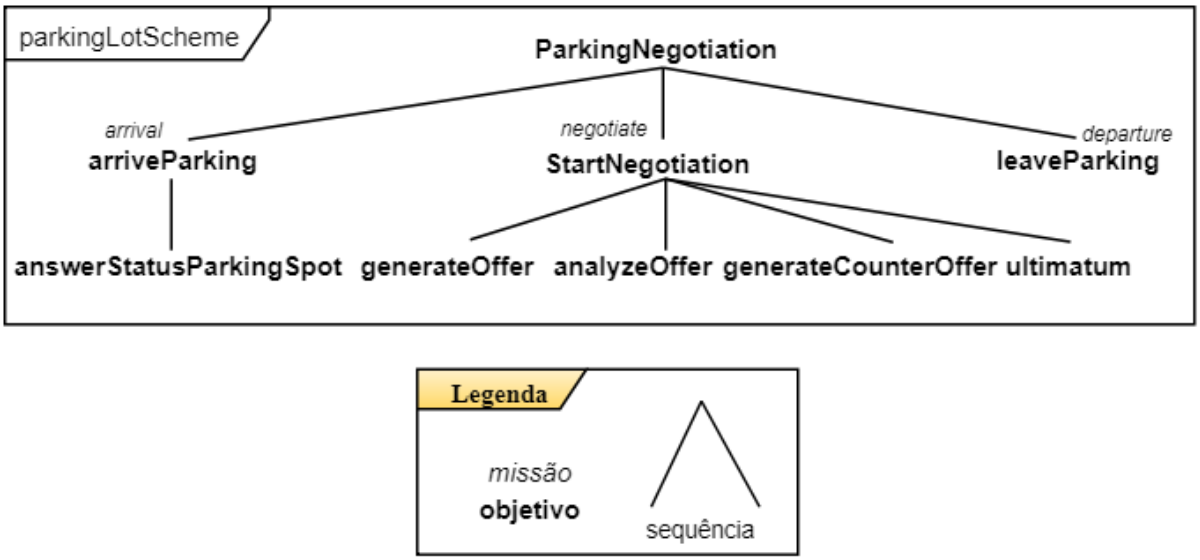

Figura 2. Esquema Moise

Conforme definido no modelo de raciocínio os agentes *drivers* podem assumir dois papeis distintos no sistema: (i) *buyer*: quando estão procurando uma vaga para estacionar; e (ii) *seller*: quando já possuem uma vaga e desejam liberar a mesma para um outro agente estacionar, isto foi implementado utilizando a camada organizacional do JaCaMo, o *Moise*, onde é possível realizar uma especificação social do SMA, definindo papeis e tarefas para cada agente dentro do sistema.

Na cama do *Cartago* do sistema foram implementados dois artefatos: o artefato `ParkingSpot`, que é a vaga a qual o agente *parkingSpotController* controla e o artefato `ProposalGenerator`, que é um artefato gerador de proposta, o qual os agentes *drivers* utilizam para calcular as propostas da negociação.

No Código 1 é apresentado a definição organizacional do sistema. Nas linhas 12 à 47 é definida a especificação estrutural do sistema, onde são definidos os papeis, as relações entre os papeis e os grupos do sistema.

Nas linhas 14 à 22 do Código 1 é possível verificar a definição dos papeis do sistema. Onde observarmos a especificação dos papeis: *parkingSpotController*, *seller* e *buyer*, estes dois últimos têm uma relação de herança com o papel *driver*, isto é, eles herdam as características do papel.

Nas linhas 32 à 37 do Código 1 está a especificação das relações entre os papeis, onde existe uma relação de comunicação entre os papeis *buyer* e *seller*, e, uma relação de autoridade dos papeis *buyer* e *seller* sobre o *parkingSpotController*.

Ainda no Código 1 nas linhas 49 à 88 está a especificação funcional do sistema, onde são definidos o esquema e as missões do sistema.

Um esquema é definido como uma árvore de decomposição de uma meta global, onde a decomposição é feita em planos [14]. A Figura 2 possui uma representação visual desta decomposição, definida no esquema do sistema (linhas 50 à 87), onde é possível visualizarmos que os planos são executados de maneira sequencial da esquerda para direita. É importante ressaltar que o código de execução de cada plano está na camada do *Jason* no *framework* JaCaMo, portanto para cada objetivo definido no *Moise* existe um plano de execução correspondente no *Jason* (alguns destes planos podem ser visualizados no trecho de código disponível no Código 2). Nas linhas 75 à 85 são definidas as missões do sistema, uma missão é um conjunto de objetivos que um agente deve cumprir [14].

Inicialmente missão `arrival` é executada pelo *driver buyer*, iniciando pelo objetivo `arriveParking`, o qual é decomposto no objetivo `answerStatusParkingSpot`, onde o agente *driver buyer* envia uma mensagem para todos os agentes *parkingSpotController* do sistema perguntando se a vaga pela qual ele é responsável está livre (vaga sem *driver* proprietário). Na primeira resposta assertiva o agente *buyer* estaciona na vaga livre e o agente *parkingSpotController* muda o *status* da vaga para ocupada, caso não haja vagas livres a segunda missão é executada.

A segunda missão executada é a `negotiate`, que inicia com o objetivo `StartNegotiation` onde a negociação da vaga acontece entre dois agentes (um *buyer* e um *seller*). Esta negociação é iniciada em um agente no papel de *seller* mandando uma mensagem informando a todos os agentes do sistema que ele possuí uma vaga para vender, então os agentes interessados na vaga (*buyer*) respondem esta mensagem com uma oferta de compra da vaga utilizando o plano `generateOffer`, que pode ser visualizado no trecho do código do agente *driver* (Código 2, linhas 2 à 14), onde a oferta é gerada pela operação `generateOffer` definida no artefato `ProposalGenerator` (linha 7), que consiste na implementação da Função 1 do modelo de raciocínio. Esta proposta é enviada do *driver buyer* para o *driver seller* na linha 8.

```
1  //Buyer
2  +!generateOffer(locationParkedSpot(
       LocationParkedSpotX, LocationParkedSpotY)
       )[source(AG)]: targetLocation(
       TargetLocationX, TargetLocationY) &
       parked(Parked) & arrived(Arrived) <-
3
4  if (Arrived = true){
5          if (Parked = false){
6                  .print("Oferta de vaga
                       recebida de ",  AG, ",
                       iniciando negociacao");
7                  generateOffer(
                       LocationParkedSpotX,
                       LocationParkedSpotY,
                       TargetLocationX,
                       TargetLocationY, Offer);
8                  .send(AG, achieve,
                       analyzeOffer(offer(Offer)
                       ));
9                  -+offer(Offer);
10                 .print("Proposta ", Offer, "
                       enviada para o agente ",
                       AG, " - ", Offer)}
11         else{
12         .print("Oferta de vaga recebida de ",
                AG, " ignorada, ja tenho vaga.")

13         }
14 }.
15
16 //Seller
17 +!analyzeOffer(offer(Offer))[source(AG)]:
       parkedSpot(ParkedSpot) & agreementDone(
       AgreementDone)  <-
18 .print("Recebi a oferta ", Offer, " do agente
       ", AG);
19
20 .send(ParkedSpot, achieve,
       askAverageSaleValue);
21 .wait(500);
22 ?averageSaleValue(M);
23 -+averageSaleValue(M);
24
25 if (Offer >= M){
26         .print("Negociacao encerrada, vaga
               vendida para o agente ", AG);
27         .send(AG, achieve, park(parkedSpot(
               ParkedSpot)));
28         .send(ParkedSpot, achieve, park);
29         -+agreementDone(true);
30 }
31
32 if (Offer < M/2){
33         .print("Negociacao encerrada com o
               agente ", AG, ", sem trato");
34 }
35
36 if ((Offer < M) & (Offer >= M/2)){
37         !generateCounterOffer(source(AG));
38 }.
39
40 //Buyer
41 +!ultimatum(counterOffer(CounterOffer),
       parkedSpot(ParkedSpot))[source(AG)]:
       offer(Offer) <-
42
43 .print("Contra oferta ", CounterOffer, "
       recebida do agente ", AG);
44
45 if ((CounterOffer - Offer) <= (Offer*0.1)){
46         .print("Contra proposta aceita do
               agente ", AG, ", estacionando na
               vaga ", ParkedSpot);
47         -+parked(true);
48         .send(ParkedSpot, achieve, park);
49 } else {
50         .print("Contra proposta rejeitada,
               sem trato. Encerrada negociacao
               com o agente ", AG);
51 }.
```

Código 2. Trecho código agente driver (*Jason*)

O agente no papel de *seller* processa esta oferta usando o plano `AnalyzeOffer`, que pode ser verificado no Código 2, linhas 17 à 38. Este plano pode ocasionar três estados possíveis para o agente *driver seller*: aceitar a proposta do *buyer*, rejeitar a proposta ou enviar uma contra oferta para o *driver buyer*, conforme definido pela Função 2.

Caso o agente *driver seller* envie esta contra oferta o *driver buyer* executa o plano `ultimatum`, que pode ser verificado no Código 2, linhas 41 à 51. Este plano garante um fim a negociação, e pode gerar dois estados possíveis: o *buyer* aceita a contra oferta ou recusa, com base na Função 3 do modelo de raciocínio.

Na parte final do Código 1, nas linhas 90 à 96, está a especificação normativa do sistema, onde são definidas por meio de normas as missões que cada papel deve executar.

Existem três normas definidas no sistema (linhas 91 à 95). A `norm1` define que o *driver buyer* tem obrigação de executar a missão `arrival`. A `norm2` define que os agentes no papel *driver* podem executar a missão `negotiate`, como existe uma herança entre os papeis *buyer* e *seller* e o papel *driver* ambos os papeis, *buyer* e *seller*, podem executar esta missão. A `norm3` define que o agente no papel *seller* tem a permissão de executar a missão `departure`.

VI. Considerações Finais

Estacionar em áreas de grande concentração urbana vem se tonado uma tarefa cada vez mais difícil, uma solução que pode ajudar a solucionar o problema é a implementação de tecnologias que auxiliem os motoristas a encontrarem vagas de estacionamentos livres.

Este trabalho propõe um modelo de raciocínio e protocolo de negociação para um *Smart Parking* com mecanismo de negociação descentralizado juntamente com uma implementação inicial, onde não existe a figura de um agente centralizador no sistema e todos os agentes independem de um módulo central, buscando uma alternativa ao projeto MAPS para lidar com negociações entre agentes de forma descentralizada.

Faz-se necessário destacar que o mecanismo de negociação implementado neste trabalho tem características descentralizadas, visto que não apresenta nenhum agente no SMA com o papel de *manager* (ou agente centralizador). Todos os agentes

do SMA possuem certa autonomia e caso um deles falhe o sistema como um todo continua funcionando e a falha permanece isolada em um agente específico, minimizando os danos ao sistema.

Por exemplo, utilizando a função *kill agent*, disponibilizada pelo *framework* JaCaMo, para eliminar agentes aleatoriamente durante a execução do sistema, é possível verificar que o SMA como um todo não deixa de funcionar, os outros agentes continuam a execução de seus planos e não ficam travados.

Porém, desenvolver um sistema multiagente completamente descentralizado não é o objetivo deste trabalho, visto que o sistema depende de funções disponibilizadas pelo *framework* JaCaMo que possuem características centralizadas, como é o caso de funções de troca de mensagens entre os agentes no sistema. Portanto, o modelo de raciocínio e protocolo de negociação aqui apresentados são descentralizados, mas são implementados por meio do *framework* JaCaMo que possui algumas funções centralizadas.

Como sequência imediata deste trabalho, pretende-se desenvolver os seguintes passos: (i) estender a implementação inicial do SMA, aqui proposta adicionando as operações de transações de UMOGs entre os agentes. Para implementação desta extensão será necessário que os *driver* do sistema possuam o conceito de carteira, a qual representará a quantidade de UMOGs que eles possuem, também será necessário implementar um método de transação de UMOGs entre os agentes; (ii) implementar este mesmo modelo de raciocínio com mais *rounds* de negociação entre os *drivers buyer* e *seller*, na tentativa de ampliar a quantidade de acordos fechados; (iii) executar simulações com o SMA desenvolvido e coletar dados; (iv) analisar os dados coletados buscando determinar possíveis pontos falhas no sistema, determinar as características do agente (local da vaga desejada, tempo de permanência, entre outras) que fazem com ele tenha um melhor desempenho na negociação (pagar um valor baixo pela vaga, esperar pouco tempo para estacionar, entre outros), e, determinar possíveis ajustes que podem ser feitos para incentivar a cooperação entre os agentes; (v) ajustar e calibrar os parâmetros do modelo de raciocínio, por meio da análise dos dados coletados; e (vi) desenvolver modelos complementares de raciocínio e protocolo de negociação, para assim elaborar comparação entre os diferentes modelos implementados, tentando verificar quais modelos raciocínio e protocolo de negociação têm melhor desempenho e determinar em quais contextos possuem este desempenho.

REFERÊNCIAS

[1] M. Batty, K. Axhausen, G. Fosca, A. Pozdnoukhov, A. Bazzani, M. Wachowicz, G. Ouzounis, and Y. Portugali, "Smart cities of the future," 2012.

[2] A. Koster, F. Koch, and A. Bazzan, "Incentivising crowdsourced parking solutions," 2014.

[3] D. D. Nocera, C. D. Napoli, and S. Rossi, "A social-aware smart parking application. ceur workshop proceedings," vol. 1260, 2014.

[4] M. Wooldridge, *An Introduction to MultiAgent Systems*, 2nd ed. New York: J. Wiley, 2009.

[5] M. Mensonides, B. Huisman, and V. Dignum, "Towards agent-based scenario development for strategic decision support," in *Agent-Oriented Information Systems IV*. Springer Berlin Heidelberg, 2008, pp. 53–72.

[6] S. Choi, J. Liu, and S.-P. Chan, "A genetic agent-based negotiation system," *Comput. Netw.*, vol. 37, no. 2, pp. 195–204, 2001.

[7] L. F. S. D. Castro, G. V. Alves, and A. P. Borges, "Using trust degree for agents in order to assign spots in a Smart Parking," *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal*, vol. 6, no. 2, pp. 45–55, Jun. 2017. [Online]. Available: http://revistas.usal.es/index.php/2255-2863/article/view/ADCAIJ2017624555

[8] G. M. P. O'Hare and N. R. Jennings, Eds., *Foundations of Distributed Artificial Intelligence*. New York, NY, USA: John Wiley Amp; Sons, Inc., 1996.

[9] O. Boissier, R. Bordini, J. Hübner, A. Ricci, and A. Santi, "Multi-agent oriented programming with JaCaMo," pp. 747–761, 2011.

[10] J. Oliver, "A machine-learning approach to automated negotiation and prospects for electronic commerce," *Journal of Management Information Systems*, vol. 13, no. 3, pp. 83–112, 1996.

[11] P. Faratin, C. Sierra, and N. Jennings, "Robotics and autonomous systems," *Negotiation decision functions for autonomous agents*, vol. 24, pp. 159–182, 1998.

[12] F. Ducheiko and G. Alves, "Modelo de raciocínio e protocolo de negociação para um estacionamento inteligente com mecanismo de negociação descentralizado," *WESAAC: Workshop-Escola de Sistemas de Agentes, seus Ambientes e apliCações*, no. 12, pp. 250–256, 2018.

[13] U. Endriss, N. Maudet, F. Sadri, and F. Toni, "Journal of artificial intelligence research," *Negotiating Socially Optimal Allocations of Resources*, vol. 25, pp. 315–348, 2006.

[14] J. F. Hübner, J. S. Sichman, and O. Boissier, "Moise tutorial (Moise 0.7)," 2010.