Uma Aplicação para Gerenciamento de Motoristas Autônomos: Usufruindo da Escalabilidade Oferecida por Sistemas Multiagentes

Leonardo Blanger[1] **, Valmir Junior**[1] **, Alison R. Panisson**[2]

[1] Universidade Regional Integrada do Alto Uruguai e das Missões (URI)
Erechim – RS – Brasil
[2]Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brasil

lblanger@hotmail.com, valmirjunior0088@gmail.com
alison.panisson@acad.pucrs.br

Abstract. *Applications that make use of multi-agent systems have received great interest from the artificial intelligence community on the last few years, due to the potential of this paradigm to handle distributed applications with dynamic environments and entities. In this paper, we describe an application for the management of autonomous drivers, based on the multi-agent systems paradigm. This paradigm proves itself suitable for the development of the proposed application, having features and facilities for the development of scalability techniques, which are the focus of this paper. On the simulations performed, optimistic results were obtained with respect to the waiting time of the clients for a taxi. It has been obtained times under 15 minutes for at most one passenger for each taxi, and times under 30 minutes for a rate of 5 passengers for each taxi.*

Resumo. *Aplicações utilizando sistemas multiagentes ganharam grande interesse da comunidade de inteligência artificial nos últimos anos, devido ao grande potencial do paradigma em tratar aplicações distribuídas com ambientes e entidades dinâmicas. Neste trabalho descrevemos uma aplicação para gerenciamento de motoristas autônomos, baseada no paradigma de sistemas multiagentes. O paradigma de sistemas multiagentes mostra-se bem adequado para o desenvolvimento da aplicação proposta, possuindo característica e facilidades no desenvolvimento de técnicas de escalabilidade, as quais são o foco deste trabalho. Nas simulações realizadas, foram obtidos resultados otimistas com relação ao tempo de espera médio de um cliente por um taxi. Foram obtidos tempos abaixo de 15 minutos para no máximo um passageiro por taxi, e tempos abaixo de 30 minutos para uma taxa de até 5 passageiros por taxi.*

1. Introdução

Dentre as diversas técnicas estudadas pela grande área de Inteligência Artificial (IA), a área de Sistemas Multiagentes (SMA) tem ganho grande foco em pesquisas relacionadas à aplicações distribuídas e soluções de problemas complexos. Isso se deve as características desse paradigma, onde as entidades envolvidas apresentam comportamento autônomo, possibilitando ainda que elas colaborem para solução de problemas [Wooldridge 2009]. Além disso, existe um grande esforço no desenvolvimento de ferramentas e plataformas

que são utilizadas para desenvolver sistemas multiagentes. Por exemplo, o Framework de programação multiagente JaCaMo [Boissier et al. 2013], tem sido desenvolvido por vários anos e é resultado de desenvolvimentos anteriores e paralelos das plataformas Jason [Bordini et al. 2007], utilizada para o desenvolvimento de agentes autônomos, CArtAgO [Ricci et al. 2011], utilizada para o desenvolvimento de ambientes compartilhados, e o modelo para organizações multiagentes Moise [Hubner et al. 2007]. Juntas, as plataformas descritas, fornecem um framework completo para o desenvolvimento de aplicações sobre o paradigma de sistemas multiagentes. Exemplos de aplicações já desenvolvidas utilizando JaCaMo podem ser encontradas na literatura [Santos et al. 2016, Sorici et al. 2011, Schmidt et al. 2016, Panisson et al. 2015a]. Ainda, a plataforma tem sido utilizada para o desenvolvimento de soluções que venceram, por alguns anos, a competição de sistemas multiagentes *Multi-agent Programming Contest Competition*[1], incluindo o novo cenário proposto em 2016.

Com o grande potencial do paradigma de sistemas multiagente, neste trabalho é proposto a utilização desse paradigma na implementação de um serviço para o gerenciamento de motoristas autônomos. O cenário investigado nesse trabalho tem características altamente dinâmicas, incluindo das entidades envolvidas, do ambiente, e da distribuição das entidades pelo ambiente. Dessa forma, o paradigma se mostra bem adequado para tal desenvolvimento. Além disso, aplicações desse tipo tem ganho grande popularidade nos últimos anos, por exemplo, empresas focadas em serviços de taxis, como a aplicação EasyTaxi[2], que opera em 30 países, incluindo 420 cidades e é particularmente popular no Brasil. Outra aplicação, também relacionada ao serviço de taxis é 99Taxis[3], atualmente trabalhando em mais de 300 cidades brasileiras. Por fim, a aplicação Uber[4] (umas das mais populares na atualidade), em setembro de 2016 estava operando em mais de 500 cidades ao redor do mundo, tornando-se modelo padrão para esse tipo de negócios[5].

Neste trabalho, é proposta uma arquitetura multiagente para sistemas que visem solucionar problemas encontrados na condução clássica desse tipo de negócio, onde tarefas como alocação de motoristas nem sempre são triviais, devido a existência de inúmeros fatores imprevisíveis a serem considerados em uma aplicação como esta no mundo real. Entre elas, a dificuldade em prever congestionamentos de tráfego, acidentes, falhas em veículos, etc. Além disso, fatores como o momento e o lugar da cidade em que clientes e motoristas podem surgir podem ser assumidos como aleatórios na prática. Estes e outros problemas levam as companhias de transportes a frequentemente delegar esta tarefa de alocação para operadores humanos [Glaschenko et al. 2009]. Como esta abordagem depende de mais mão de obra, ela induz altos custos para o serviço, além de ser baixa escalabilidade, já que a companhia dificilmente poderá contratar e demitir funcionários de acordo com as variações na demanda pelo serviço. A maioria desses problemas podem ser sanados com a utilização de sistemas multiagentes, os quais são altamente escaláveis (como será descrito nesse artigo) e as entidades envolvidas podem constantemente ser atualizadas com novas técnicas de predição, estatística, otimização, etc.

[1] `https://multiagentcontest.org`
[2] `http://www.easytaxi.com/`
[3] `http://www.99taxis.com/`
[4] `https://www.uber.com`
[5] Dados disponíveis no site oficial de cada aplicação.

2. Linguagem de Programação Orientada à Agentes

Entre as muitas linguagens de programação orientadas à agentes, tais como Jason, Jadex, Jack, AgentFactory, 2APL, GOAL, Golog, and MetateM, como discutido em [Bordini et al. 2009], nesse trabalho foi utilizada a plataforma Jason [Bordini et al. 2007], em particular o Framework de programação JaCaMo [Boissier et al. 2013]. Jason estende AgentSpeak(L), uma linguagem de programação orientada a agentes baseada em lógica abstrata introduzida por [Rao 1996], a qual é uma das mais bem conhecidas linguagens inspirada pela arquitetura BDI (*Beliefs-Desires-Intentions*), uma das mais estudadas arquiteturas para agentes cognitivos.

Uma das características mais importantes na Plataforma Jason [Bordini et al. 2007], relacionada a aplicação aqui proposta, é a abordagem utilizada na comunicação entre agentes. A comunicação entre agentes em Jason é feita através de ações (internas) predefinidas. A sintaxe mais comum para essas ações é:

```
.send(receiver,illocutionary_force,content)
```

onde `receiver` representa o identificador (nome) do agente que vai receber a mensagem, `content` representa um literal que pode representar um evento, um plano, ou uma lista de literais ou planos, e `illocutionary_force` representa o ato de fala (ou performativa) que pode ser utilizado para comunicação. As performativas já disponíveis em Jason [Bordini et al. 2007], utilizadas nesse trabalho, são listadas abaixo com uma breve explicação[6]

- tell: é utilizado quando o agente o qual envia a mensagem deseja que o agente que recebe a mensagem acredite no literal, o qual é conteúdo da mensagem.
- achieve: é utilizado quando o agente o qual envia a mensagem deseja que o agente que recebe a mensagem tente alcançar algum objetivo.
- askOne: é utilizado quando o agente o qual envia a mensagem deseja saber se o conteúdo da mensagem é verdadeiro para o agente que recebe a mensagem.

Além das características descritas, Jason permite a definição de novas performativas, incluindo a possibilidade de dar significado especial para elas. Por exemplo, os autores em [Panisson et al. 2015b], definiram novas performativas em Jason para diálogos baseados em argumentação. Neste trabalho, utilizamos somente as performativas existentes, as quais são suficientes para a nossa necessidade.

Jason também permite a implementação de ações internas em Java para vários outros propósitos além de comunicação. Por exemplo, na criação de uma interface para código legado, regras de negócio, operações matemáticas complexas, hardware, etc. Neste trabalho, entre outras ações internas, foi implementado uma ação interna para calcular o quanto um motorista é adequado para uma requisição de cliente, ou seja, ela calcula o melhor motorista para uma requisição. Nessa primeira versão do trabalho, utilizou-se uma distância simples entre o motorista e o cliente. Cálculos mais refinados vão ser utilizados em versões futuras da aplicação.

3. Rede e Ambiente Distribuído

Um ambiente multiagente é uma abstração de um ambiente real ou virtual, o qual pode ser percebido e manipulado pelos agentes. Para o desenvolvimento de aplicações multiagente em Jason, normalmente é utilizada a plataforma CArtAgO [Ricci et al. 2011],

[6]Similarmente, agentes podem executar um *broadcast*.

devido ao Framework JaCaMo [Boissier et al. 2013], que combina a plataforma Jason [Bordini et al. 2007] para o desenvolvimento de agentes, a plataforma CArtAgO [Ricci et al. 2011] para o desenvolvimento de ambientes multiagente e o modelo Moise [Hannoun et al. 2000] para especificação organizacional. Para o desenvolvimento de aplicações móveis, o Framework JaCa-Android[7] [Santi et al. 2010] pode ser utilizado. Ele provê um nível de abstração orientado a agentes para a modelagem e programação de aplicações móveis para a plataforma Android utilizando uma combinação de Jason e CArtAgO.

4. Serviço para Motoristas Autônomos

Neste trabalho é proposta uma arquitetura inspirada no paradigma multiagentes, a ser empregada em sistemas de alocação de motoristas autônomos. De maneira breve, a aplicação funciona com 3 tipos de agentes:

(i) o primeiro deles, agentes que rodam em dispositivos móveis dos *clientes* e são responsáveis por coletar informações do usuário cliente, interagir com o usuário e comunicar-se com demais agentes do sistema. Esse agente, mantem a posição do usuário de maneira atualizada em sua base de crenças e, sempre que o cliente requisitar uma viagem, ele interage com o usuário requisitando o ponto de destino. O agente *cliente* também é responsável por comunicar tais informações (posição inicial e final da viagem) para agentes chamados de *agencias*, e interagir com agentes que representam os motoristas na aplicação, com a finalidade de informar ao usuário a posição do motorista alocado para ele;

(ii) o segundo tipo de agente corresponde aos *motoristas* na aplicação. O agente que representa um motorista também é embarcado em dispositivos móveis e é responsável por coletar a posição atual do motorista representado por ele e informar quando o mesmo está disponível para novas viagens. Ele também é responsável por informar o cliente que for alocado para ele, a sua atual posição no percurso realizado para coletar o cliente no ponto de partida da viagem;

(iii) o último agente utilizado, corresponde às chamadas *agencias*, que mantem o maior poder de processamento da aplicação, e são responsáveis por gerenciar a alocação de motoristas disponíveis com requisições de clientes.

De forma geral, a aplicação funciona da seguinte forma: quando um motorista está disponível (informação fornecida pelo usuário motorista ao seu agente rodando no dispositivo móvel), o agente informa todas as agências do sistema sobre a disponibilidade daquele particular motorista, através da ação `tell: available`, como mostrado na Figura 1. Quando uma agência recebe tal informação, ela filtra esta informação, verificando se aquele motorista pertence à área da qual aquela agência em particular é responsável, e caso pertencer, uma crença é criada, armazenando aquele motorista como uma das opções entre os motoristas disponíveis desta agência, caso contrário, a informação é descartada. Cada agência é responsável por uma única área de cobertura de motoristas, e estas áreas não se sobrepõe, isso significa que a informação de cada motorista disponível na aplicação vai estar armazenada em apenas uma agência em um dado momento. Deslocamentos significativos dos motoristas são também informados as agências, para que estas crenças sejam atualizadas. Conforme será analisado adiante, a extensão e localização destas áreas podem se comportar dinamicamente no sistema.

[7] `http://jaca-android.sourceforge.net/`

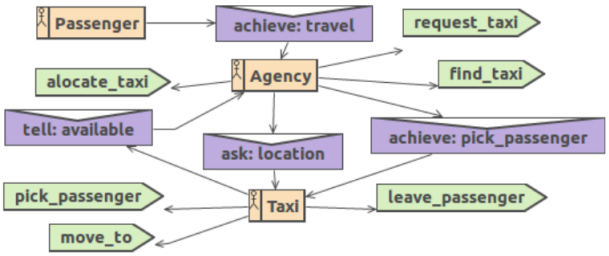

Figura 1. Diagrama Prometheus.

Quando um cliente requisita uma viagem, utilizando seu dispositivo móvel, o agente rodando em seu dispositivo solicita o local de destino e faz uma requisição à todas as agências do sistema, através da ação `achieve: travel`, como mostrado na Figura 1. Quando as agências recebem tal requisição, ela também é filtrada, analisando se esta requisição foi realizada em uma área que é monitorada por aquela agência ou não. Se a área é monitorada pela agencia, ela localiza sua melhor opção de motorista e informa as demais agências também responsáveis por requisições de clientes naquela área (as áreas de requisição de clientes são sobrepostas, de modo a encontrar o melhor motorista de forma global, conforme será melhor detalhado adiante).

Quando todas as agências localizarem sua melhor opção e comunicaram as demais, a agência que possui a melhor opção global comunica o agente do respectivo motorista, e aquele cliente é alocado para tal motorista, através da ação `achieve: pick_passenger` (Figura 1), e remove a crença informando que este motorista estava disponível em sua área. Em seguida, as devidas interações com os usuários ocorrem. O agente do motorista passa a ser capaz de comunicar sua posição ao agente do cliente alocado para ele. Uma visão geral da aplicação pode ser vista na Figura 2, e a interação entre os agentes do sistema, bem como das ações realizadas por eles pode ser vista no diagrama Prometheus da Figura 1.

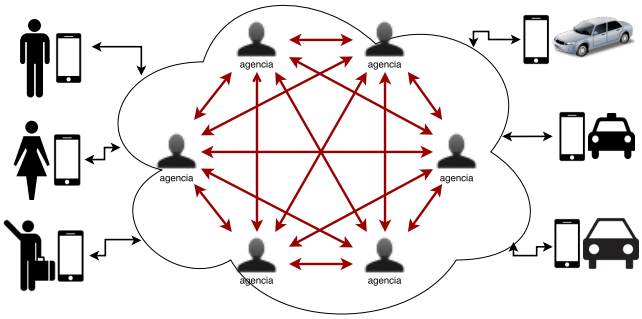

Figura 2. Visão Geral da Aplicação.

Entre diversas características interessantes na utilização do paradigma de sistemas multiagentes em nossa aplicação, neste trabalhos descrevemos as questões relacionadas a escalabilidade do sistema. Para esse fim, dividimos as próximas subseções para explicar a escalabilidade de agentes em relação a capacidade deles se auto-replicarem para melhorar o desempenho da aplicação, bem como a capacidade deles se moverem para servidores diferentes, com o objetivo de reduzir a sobrecarga na rede.

4.1. Escalabilidade de Agentes

Na aplicação de serviço para motoristas autônomos proposta neste trabalho, os agentes responsáveis pelo maior processamento (as agências) são organizados de maneira que eles tratem o espaço geográfico de forma diferente para a disponibilidade de motoristas e requisições de clientes. Como descrito, cada motorista disponível é registrado em apenas uma agência, portanto, as áreas tratadas por cada agência não se sobrepõem. Requisições de clientes são tratadas, possivelmente, por várias agências, dado que as áreas de tratamento de requisições de cada agência sobrepõe as áreas de agências adjacentes (elas correspondem a uma porcentagem variável de acréscimo à área tratada para motoristas).

No decorrer do dia, neste tipo de cenário, já é conhecido que a quantidade de motoristas disponíveis em cada região e a quantidade de requisições de clientes pode variar. Por exemplo, tendo-se poucos motoristas e requisições em períodos noturnos, e grandes quantidades de motoristas e requisições de clientes em horas de pico durante o dia. Com base nisso, é proposto uma arquitetura que permite a criação e destruição de agentes do tipo agência, conforme a necessidade da aplicação, de acordo com a quantidade de motoristas e requisições de clientes. Com a utilização da plataforma Jason, a criação e destruição de agentes pode ser feita facilmente em tempo de execução do sistema.

Na aplicação proposta, há uma numeração sequencial de agências, por exemplo `agencia03` para a terceira agência criada no sistema, e o código de todas as agências é o mesmo (`agencia.asl`). As agências são diferenciadas pela área à qual são responsáveis, e pelos motoristas disponíveis em sua área, ou seja, suas crenças. Portanto, quando uma agência está sobrecarregada com muitas requisições e/ou motoristas cadastrados em sua área, essa agência executa a ação interna de criação de uma nova agência, seguindo a ordenação de nomes. Após a criação dessa nova agência, a agência a qual a criou divide a sua área de cobertura em duas partes, ficando esta responsável por uma destas partes, e informando a nova agência que esta será responsável pela outra parte, e pelos motoristas que estiverem disponíveis nesta respectiva parte (a agência criadora envia para a agência criada as crenças referentes aos motoristas da nova área, e as remove de seu conjunto de crenças). Este processo reduz o número de motoristas cadastrados e requisições em uma única área. Caso ainda exista uma concentração de motoristas e requisições em uma dessas novas áreas, as agências são capazes de subdividir tal área novamente, até que a sobrecarga seja eliminada. Da mesma forma, quando as agências estão com poucos motoristas registrados e requisições de clientes, elas são capazes de reagrupar uma área, passando as informações para uma única agencia, que se torna responsável pela área maior, e as demais deixando de executar.

4.2. Escalabilidade da Rede

Outra questão bastante interessante neste tipo de aplicação é a escalabilidade da rede, onde muitas vezes existe a necessidade de redistribuir processamento, devido a sobrecarga de servidores. Ainda, existe a possibilidade de servidores ficarem *offline* devido a problemas de infraestrutura, etc. A abordagem proposta neste trabalho, é capaz de lidar com as situações mencionadas de maneira eficiente e transparente para os usuários do serviço, onde uma agência originada de uma replicação (como descrito na subseção anterior) pode ser instanciada em um servidor diferente da agência que originou a replicação, permitindo uma distribuição adequada de processamento na rede. Também, em relação a inatividade de servidores, é possível utilizar técnicas de tolerância à falhas, com o objetivo de mo-

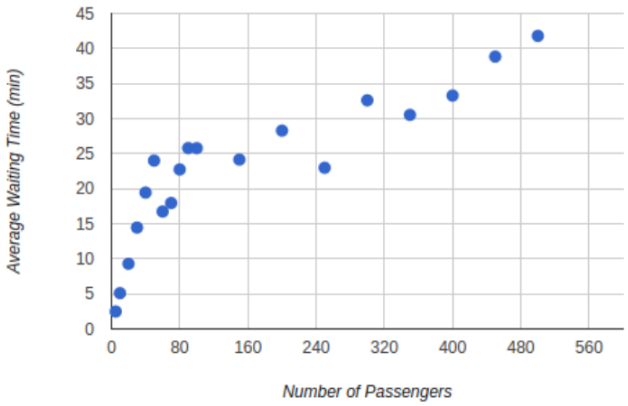

Figura 3. Tempo médio de espera para uma frota de 50 taxis

nitorar agências que deixarem de responder, criando agências substitutas com o mesmo estado mental (crenças) de agências que se tornarem inativas, e capazes de assumir as tarefas de agências localizadas servidores faltosos de forma totalmente transparente para os outros agentes e para os usuários do serviço.

Como não existe restrição, em sistemas multiagente, sobre em que servidor um agente deva rodar, a aplicação pode funcionar de forma totalmente distribuída. Nesta situação, a abordagem de criar novas agências para suprir demandas de carga é capaz de tirar proveito dos paradigmas de computação paralela e distribuída, permitindo não apenas uma maior escalabilidade no sistema, mas também um melhor balanceamento de carga e um melhor aproveitamento dos recursos computacionais.

5. Testes

Uma versão inicial do sistema, ainda não distribuída em dispositivos móveis, foi avaliada utilizando uma cidade simulada que consiste de um grid quadrado de 40 ruas horizontais e 40 ruas verticais, totalizando 1600 intersecções. Por motivos de simplificação, foi assumido que um veículo normal trafegando em horário normal leva aproximadamente 6 segundos para percorrer uma quadra, portanto, a velocidade assumida para todos os taxis do sistema é de 6 segundos por unidade de distância no grid.

Foram realizados testes considerando uma frota de 50 taxis e variando-se o número clientes. O sistema foi avaliado em quantidades de clientes múltiplas de 10 maiores que 0 e menores ou iguais a 500 (10, 20, .., 490, 500). Todos os taxis iniciam ao mesmo tempo no centro do grid, enquanto os passageiros possuem tanto a posição inicial quanto o destino gerados aleatoriamente. Todos os clientes são instanciados ao mesmo tempo e estão codificados para requisitar a viagem imediatamente (com a limitação de cada um executar como uma thread na simulação, e portanto, não há garantia de que as requisições sejam feitas exatamente ao mesmo tempo).

Para cada simulação, foi registrado o tempo médio de espera dos clientes, ou seja, o tempo decorrido do momento em que o cliente efetua a requisição até o momento em que o taxi chega para atendê-lo. Os resultados são mostrados na Figura 3. Como é possível observar, o tempo de espera se mantém abaixo de 15 minutos para quantidade de passageiros menores que a quantidade de carros. Além disso, o tempo se mantém abaixo de 30 minutos para até 250 passageiros, o que pode ser considerado um tempo de es-

pera aceitável, considerando a taxa de 5 passageiros para cada motorista, e considerando a independência de operadores humanos nesta alocação. É importante lembrar que, em um ambiente real, existe ainda a latência de comunicação entre os dispositivos móveis dos clientes, motoristas, e os servidores encarregados da execução das agências. Entretanto, considerando a constante evolução das tecnologias de redes móveis atualmente, este tempo foi desconsiderado para efeitos de simulação.

6. Trabalhos Relacionados

[Alshamsi et al. 2009] propõe uma abordagem multiagente auto-organizável para tratar a tarefa de associar taxis com clientes, com o objetivo de melhorar a performance da atual abordagem em uma companhia de taxis real. A companhia opera dividindo a cidade em regiões, e chamadas realizadas por clientes de dentro de qualquer destas regiões deve ser atendida, preferencialmente, por um taxi disponível que estiver atualmente localizado na mesma região da chamada. Esta estratégia busca reduzir não apenas o tempo de espera do passageiro, mas também o tempo de deslocamento do motorista. No caso de não haver taxis disponíveis na mesma região da chamada, o sistema busca por taxis disponíveis em regiões adjacentes, onde estas relações de adjacência entre regiões são estáticas e precisam ser manualmente especificadas no sistema. A abordagem multiagente auto-organizável implementada na companhia busca estabelecer dinamicamente as relações de adjacência entre as regiões buscando fazer um melhor uso de informações sobre a atual distribuição dos taxis pelas regiões e alterações nos padrões de tráfego. Isto é alcançado através do tratando as regiões como agentes no sistema, as chamadas e os taxis como recursos. Neste trabalho, a companhia foi capaz de reduzir significativamente o tempo de espera, e aumentar a utilização dos taxis (possivelmente maximizando os lucros da companhia).

[Seow et al. 2010] propõe um sistema multiagente onde agentes, associados com taxis, são capazes de negociar suas requisições depois destas terem sido entregues a eles pelo sistema, permitindo que os taxis "troquem" passageiros que ainda não tiverem sido atendidos, de forma a maximizar a satisfação média de grupos de requisições. O sistema opera agrupando as requisições dos clientes que ocorrem dentro de um determinado intervalo de tempo e disponibilizando a cada um destes grupos o conjunto de taxis que estiverem disponíveis no final de cada intervalo. O sistema então aplica um algoritmo descentralizado chamado MA^3-LM, o qual funciona dividindo o seu processo de raciocínio em um número de *rounds* de negociação entre os agentes. Nestes *rounds* os taxis são capazes de trocar suas requisições de forma a aumentar o somatório da qualidade de serviço *(QoS)* dentre todas as requisições. O ponto principal desta abordagem é maximizar a média de satisfação dos clientes de forma global em um grupo, em oposição as abordagens existentes que visam maximizar a satisfação local para cada das requisições de forma serial. Maximizando globalmente a satisfação, é esperado que um maior número de clientes possa ser bem satisfeito. Simulações experimentais mostraram que o sistema é capaz de reduzir o tempo de espera dos clientes, e o tempo de viagem sem passageiros.

Em [Glaschenko et al. 2009], foi implementado um sistema multiagente para a tarefa de associar taxis com clientes em uma grande companhia de taxis operando em Londres. Até então, a companhia tratava manualmente as requisições de clientes e o gerenciamento da frota, precisando manter uma equipe de 130 funcionários para esta finalidade. Devido ao sistema ter sido aplicado na prática (no mundo real), os autores tiveram

que lidar com eventos complexos, como por exemplo: pode-se assumir o momento das requisições e a localização atual dos taxis como eventos imprevisíveis; existem diferentes tipos de clientes com diferentes necessidades (normal, deficiente, necessitando cadeira para criança, transportando *pets*, carregando bagagem, etc...); uma grande e dinâmica variedade de veículos, sendo que nem todos são capazes de satisfazer todas as situações anteriores; existem motoristas *freelancers* que são permitidos a usarem seus veículos para seus próprios clientes, entrando e deixando o sistema a qualquer momento; imprevisibilidade de congestionamentos, falhas no veículo, acidentes e situações similares. O sistema é composto por um número de módulos interconectados por filas de eventos, e faz uso de uma abordagem conhecida como "*adaptative scheduling*". O ponto principal desta técnica é o fato de que, a cada evento (surgimento de um cliente ou mudança no *status* de um motorista), o sistema não precisa reconstruir toda a distribuição entre taxis e clientes do início, mas apenas as partes desta organização que forem afetadas pelo evento. O sistema é capaz de estabelecer o mapeamento entre clientes e taxis baseado em vários critérios, tais como a distância física até o cliente, tráfego, tipo de cliente, bem como lidar com situações como um eventual cancelamento de uma requisição e gerenciar o tempo ocioso dos motoristas, sendo capaz de melhorar a satisfação tanto do cliente como do motorista. O sistema também implementa técnicas para penalizar motoristas que tentarem trapacear no sistema provendo informações erradas para benefício próprio. Com esta implementação, a companhia foi capaz de tratar automaticamente 98,5% das requisições (sem qualquer interferência humana), além de reduzir o número de requisições perdidas e o número de viagens ociosas.

A abordagem proposta nesse trabalho, difere de [Alshamsi et al. 2009] nas questões de escalabilidade descritas, inclusive no tratamento de requisições de clientes. Nós acreditamos que o trabalho descrito em [Alshamsi et al. 2009] poderia se beneficiar com nossa abordagem. Além disso, o foco aqui é diferente do apresentado em [Seow et al. 2010], onde não existe o tratamento de questões relacionadas à escalabilidade do sistema, e esse trabalho também poderia se beneficiar com a proposta apresentada aqui para escalabilidade. Finalmente, esse trabalho visou atacar os problemas descritos em [Glaschenko et al. 2009]. Da mesma forma que esse último trabalho pode usufruir da escalabilidade de nossa abordagem, a nossa aplicação pode incorporar os cálculos e técnicas utilizadas em [Glaschenko et al. 2009].

7. Conclusão

Neste trabalho descrevemos uma abordagem para serviço de motoristas autônomos baseada no paradigma de sistemas multiagentes, o qual permite tratar de forma eficiente e transparente, os problemas de escalabilidade de tal aplicação.

As contribuições deste trabalho foram no sentido de propor uma abordagem para um problema real, com alta escalabilidade, e por ser uma abordagem essencialmente distribuída, capaz de tirar proveito dos benefícios das técnicas de computação distribuída. A proposta também é eficiente no sentido de balanceamento de carga, sendo automaticamente capaz de se redimensionar em caso de sobrecarga. A arquitetura da aplicação também é capaz de tirar proveito da recente popularização de tecnologias móveis, como *smarthphones* e redes sem fio. A arquitetura implementada e simulada se mostra aceitavelmente eficiente, e de bom custo-benefício, pois é capaz de atender os requisitos do sistema de forma automatizada, eliminando a necessidade de operadores humanos.

Referências

Alshamsi, A., Abdallah, S., and Rahwan, I. (2009). Multiagent self-organization for a taxi dispatch system. In *8th Int. Conf. on autonomous agents and multiagent systems*, pages 21–28. Citeseer.

Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A., and Santi, A. (2013). Multi-agent oriented programming with jacamo. *Science of Computer Programming*, 78(6):747–761.

Bordini, R. H., Dastani, M., Dix, J., and Seghrouchni, A. E. F. (2009). *Multi-Agent Programming: Languages, Tools and Applications*. Springer Publishing Company, Incorporated, 1st edition.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology)*. John Wiley & Sons.

Glaschenko, A., Ivaschenko, A., Rzevski, G., and Skobelev, P. (2009). Multi-agent real time scheduling system for taxi companies. In *8th Int. Conf. on Autonomous Agents and Multiagent Systems*.

Hannoun, M., Boissier, O., Sichman, J. S., and Sayettat, C. (2000). Moise: An organizational model for multi-agent systems. In *Advances in Artificial Intelligence*, pages 156–165. Springer.

Hubner, J. F., Sichman, J. S., and Boissier, O. (2007). Developing organised multiagent systems using the moise+ model: programming issues at the system and agent levels. *Int. J. of Agent-Oriented Software Engineering*, 1(3-4):370–395.

Panisson, A. R., Freitas, A., Schmidt, D., Hilgert, L., Meneguzzi, F., Vieira, R., and Bordini, R. H. (2015a). Arguing About Task Reallocation Using Ontological Information in Multi-Agent Systems. In *12th Int. Workshop on Argumentation in Multiagent Systems*.

Panisson, A. R., Meneguzzi, F., Vieira, R., and Bordini, R. H. (2015b). Towards practical argumentation in multi-agent systems. In *Brazilian Conf. on Intelligent Systems, BRACIS 2015*.

Rao, A. S. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language. In *Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world : agents breaking away: agents breaking away*, MAAMAW '96, pages 42–55, New York.

Ricci, A., Piunti, M., and Viroli, M. (2011). Environment programming in multi-agent systems: An artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192.

Santi, A., Guidi, M., and Ricci, A. (2010). Jaca-android: an agent-based platform for building smart mobile applications. In *Int. Workshop on Languages, Methodologies and Development Tools for Multi-Agent Systems*, pages 95–114. Springer.

Santos, F., Adamatti, D. F., Rodrigues, H., Dimuro, G., Jerez, E. D. M., and Dimuro, G. (2016). A multiagent-based tool for the simulation of social production and management processes of urban ecosystems using the jacamo framework: A case study of san jerónimo vegetable garden-seville, spain. *J. of Artificial Societies and Social Simulation*, 19(3).

Schmidt, D., Panisson, A. R., Freitas, A., Bordini, R. H., Meneguzzi, F., and Vieira, R. (2016). An ontology-based mobile application for task managing in collaborative groups. In *Florida Artificial Intelligence Research Society Conference*.

Seow, K. T., Dang, N. H., and Lee, D.-H. (2010). A collaborative multiagent taxi-dispatch system. *IEEE Transactions on Automation Science and Engineering*, 7(3):607–616.

Sorici, A., Boissier, O., Picard, G., and Santi, A. (2011). Exploiting the jacamo framework for realising an adaptive room governance application. In *Proc. of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11, & VMIL'11*, pages 239–242. ACM.

Wooldridge, M. (2009). *An introduction to multiagent systems*. John Wiley & Sons.