**Revista de Informatica Teórica e Aplicada - RITA - ISSN 2175-2745´**

**Vol. 29, Num. 01 (2022) 81-90**

**ARTIGO DE PESQUISA**

**Integração de Sistemas Multiagentes Embarcados com Ferramentas de Simulação Urbana e Aplicações IoT**

Lucas Fernando Souza de Castro1\*, Fabian Cesar P. B. Manoel2, Vin´ıcius Souza de Jesus2,

Carlos Eduardo Pantoja2, Andre Pinz Borges´ 3, Gleifer Vaz Alves3

|  |
| --- |
| **Resumo:** O desenvolvimento de sistemas de cidades inteligentes conectados à Internet das Coisas (IoT) tem sido o objetivo de vários trabalhos no campo do sistema multi-agente. No entanto, apenas alguns projetos demonstram como implantar e fazer a conexão entre os sistemas de empregações. Este artigo propõe uma abordagem para a integração de um MAS através da estrutura JaCaMo mais uma Ferramenta de Simulação Urbana (SUMO), aplicações de IoT (Node-RED, InfluxDB e Grafana) e uma plataforma de IoT (Konker). A integração apresentada aqui se aplica em um cenário de Estacionamento Inteligente com recursos reais, onde é mostrada a integração e a conexão através de todas as camadas, do nível do agente a artefatos, incluindo ambiente real e simulação, bem como aplicações de IoT. Em trabalhos futuros,  pretendemos estabelecer uma metodologia que mostre como integrar adequadamente essas diferentes aplicações, independentemente  do cenário e das ferramentas usadas.  **Palavras-chave:** Smart City — Multi-Agents — Simulação Urbana — IoT |
| 1*Institute of Computing, Universidade Estadual de Campinas (UNICAMP), Campinas - Sao Paulo, Brazil˜*  2*Centro Federal de Educac¸ao Tecnol˜ ogica Celso Suckow da Fonseca (CEFET/RJ), Rio de Janeiro - Rio de Janeiro, Brazil´*  3 *Programa de Pós-Graduação em Ciênciacomputador, Universidade Tecnologica Federal do Paran'a*  *(UTFPR), Curitiba - Paran'a , Brasil\** \***Autor correspondente**: lucas.castro@ic.unicamp.br, {souza.vdj, fabiancpbm}@gmail.com, pantoja@cefet-rj.br, {apesborg, gleifer}@utfpr.edu.br  **DOI:** http://dx.doi.org/10.22456/2175-2745.110837 • **Recebido:** 20/01/2021 • **Aceito:** 29/08/2021  *CC BY-NC-ND 4.0 - Este trabalho é licenciado sob uma Licença Internacional Creative Commons Atribuição-Não Comercial-Não Comercial-Não-Não-0.* |

# Introdução

Em um de seus relatórios, a Cisco aponta cerca de 50 bilhões de dispositivos para a *Internet das Coisas* (IoT) em operação a partir de 2022 [1]. Além disso, há uma demanda crescente por sistemas de computador que fornece funcionalidade e conectividade para esses dispositivos. O desenvolvimento desses sistemas deve atender à dinâmica dos cenários onde estão localizados os dispositivos para IoT, como o cenário da cidade inteligente.

Uma cidade inteligente poderia ser definida como uma cidade que fornece tecnologias integradas aos seus cidadãos para ter uma qualidade de vida [2]. Essa qualidade proporcionada à população pode ser diversificada em diferentes categorias: governança, segurança, economia, meio ambiente e mobilidade. Esta última tem sido uma área de atenção para a pesquisa e investimento industrial em cidades inteligentes [3]. Assim, diante dessas demandas, deve haver um desenvolvimento de sistemas capazes de apoiar a mobilidade dos cidadãos.

Os Sistemas Multi-Agentes (MAS) são candidatos a fornecer o suporte necessário para a mobilidade dos cidadãos. É alcançável devido a algumas características fornecidas por esses sistemas, por exemplo, autonomia, pró-ativação com raciocínio cognitivo, tomada de decisão e, finalmente, a capacidade de se comunicar uns com os outros, a fim de alcançar um objetivo comum ou conflitante [4].

Por meio desses recursos, os problemas relacionados à mobilidade em cidades inteligentes são gerenciados de forma proativa sem interferência ou mesmo conhecimento dos cidadãos, pois os agentes inteligentes podem desempenhar esse papel em benefício do usuário e negociar com seus pares tomar decisões relevantes em um momento específico para qualquer serviço. Considerando que em uma cidade inteligente poderia haver inúmeros dispositivos capazes de trocar informações, a IoT emerge como uma tecnologia essencial na implementação de tais soluções, uma vez que, por definição, a IoT [5] é um conjunto de dispositivos interligados pela Internet. Como o MAS pode ser empregado de forma simulada (virtual) e física, é possível prever dispositivos incorporados com MAS aplicados em aplicações de IoT.

Observando a demanda tecnológica requirida pela mobilidade urbana, há a exigência de um sistema adequado capaz de enfrentar situações inteligentes e organizacionais. Além disso, deve contar com a IoT para dimensionar a resolução do problema. Portanto, o objetivo deste trabalho é descrever uma abordagem para integrar diferentes aplicações, envolvendo MAS, Simulação Urbana, Sistemas Embarcados e aplicações de IoT com a possibilidade de criar soluções construídas sob essas tecnologias, a fim de resolver questões de mobilidade, heterogeneidade e acoplamento entre as tecnologias. Assim, essa integração visa proporcionar tanto uma solução capaz de ser aplicada em diferentes cenários e aplicações (heterogeneidade), quanto também uma solução que proporcione a possibilidade de utilizar apenas uma peça - subconjunto - da integração como solução (baixo acoplamento entre tecnologias).

Aqui, usamos a estrutura jacamo [6] para implementar nossa abordagem, que é uma composição de três frameworks de programação: Jason, CArtAgO e Moise. Jason [7] interpreta uma linguagem orientada para agentes chamada *AgentSpeak* em Java para programar agentes BDI (*Crença-Desejo-Intenção*). O CArtAgO é baseado no modelo *Agents & Artifacts (A&A),* que permite o desenvolvimento da camada de ambientes MAS e integra os agentes e artefatos de um MAS [8]. Finalmente, a Moise implementa um modelo organizacional para MAS baseado em agrupamento, comportamento e objetivos [6].

O restante deste artigo é organizado da seguinte forma. A Seção 2 apresenta a abordagem geral para a integração dos sistemas. A Seção 3 descreve a integração entre agentes mas e artefatos com a simulação urbana. Posteriormente, a Seção 4 detalha a integração de agentes e artefatos com o ambiente físico. Enquanto, na Seção 5, é apresentada a integração do MAS com as aplicações de IoT. Finalmente, a Seção 6 descreve as observações finais.

# Integração de Sistemas

Esta seção apresenta uma abordagem heterogênea para integrar MAS, Simulação Urbana, Sistemas Embarcados e a camada IoT. Esta abordagem descreve como desenvolver um MAS incorporado para aplicações de IoT usando a estrutura JaCaMo e integrando várias ferramentas para facilitar a aplicação e visualização de tais sistemas, inclusive em diferentes domínios de aplicativos. A abordagem aqui empregada permite o desprendimento de uma estrutura de sistema em níveis onde as responsabilidades de software, hardware e IoT são programadas separadamente para facilitar a integração dependendo da solução que foi criada e quais níveis serão empregados. A proposta de integração do sistema é composta por quatro níveis bem definidos: Agentes, Meio Ambiente, IoT e Nível de Simulação, conforme illustrado na Figura 1.

A camada de nível de agente é responsável pela criação de agentes de MAS usando a linguagem Jason. Os agentes têm duas extensões específicas: a primeira é para interligar hardware usando ARGO [9] e a segunda é para comunicar e transportar agentes usando IoT. A camada de nível de agente conecta-se com diferentes partes do sistema: i) o nível do ambiente através dos artefatos fornecidos por Cartago [8]; ii) o nível de simulação para permitir a simulação adequada dos agentes utilizando o simulador urbano SUMO [10]; iii) o nível do ambiente físico diretamente através do agente estendido ARGO; iv) e o nível de IoT usado para dimensionar os aplicativos de agente.

O nível do ambiente se divide entre os artefatos (fornecidos por Cartago) e o ambiente físico, onde plataformas de hardware com sensores e atuadores podem interagir com o mundo físico. Assim, é possível criar *Artefatos Simulados*, aqueles que mantêm informações apenas no nível de software, e Artefatos Físicos, que executam a interface hardware através de uma Interface Serial. Ambos os artefatos podem coexistir em uma solução desenvolvida usando a abordagem proposta, uma vez que as tecnologias utilizadas em cada nível são independentes. Artefatos físicos também podem se relacionar com o nível de IoT via MQTT (*Mensagem*

*Conexões de telemetria de telemetria* com *o KonkerLabs* e o *Aplicativo Node-Red*.

No nível de IoT, o *ContextNet* é empregado como um middleware para IoT que pode lidar com várias características de sistemas distribuídos, como conectividade, escalabilidade e comunicabilidade [11], para comunicação entre os agentes. A abordagem proposta com os agentes comunicadores pode se conectar ao *ContextNet* para trocar mensagens ou agentes de transporte de um MAS incorporado em um dispositivo para outro MAS de um dispositivo diferente.

Dessa forma, é possível criar dispositivos usando a abordagem do agente para atuar fisicamente em aplicativos de Sistema Cibernético e IoT. Neste artigo, um Dispositivo poderia ser definido como um componente composto de um MAS incorporado usando Jason e CArtAgO, capaz de interligar sensores e atuadores conectados a um microcontrolador (ATMEGA, PIC, Arduino) e de se conectar a uma rede IoT para trocar informações.

# Integração: Simulação Urbana e MAS

A estrutura jacamo permite o uso de três camadas de desenvolvimento para MAS: agentes, artefatos, and social. Além disso, pode se interligar com diferentes ferramentas, por exemplo, simulação, sistemas embarcados e IoT. Tal capacidade deve-se à flexibilidade da estrutura, fornecendo artefatos ambientais da CArtAgO. Esta seção descreve como JaCaMo e SUMO se conectaram.

|  |
| --- |
| **Figura 1.** Abordagem de integração de sistemas: visão geral    **Figura 2.** UTFPR (Ponta Grossa) - Mapa e Representação no SUMÔ |

Para ilustrar essa integração, um MAS desenvolvido através do JaCaMo Framework é usado para atribuir e negociar vagas de estacionamento em um Estacionamento Inteligente. O MAS desenvolvido abrange um estacionamento inteligente com duas áreas de estacionamento (ver Figura 2): círculo vermelho (à esquerda) para estudantes e visitantes, círculo verde (à direita) apenas para estudantes. Em relação à simulação com o SUM, uma rede foi desarquivada com as vias e as áreas de estacionamento [[1]](#footnote-1)para simular esse cenário como mostrado na Figura 2.

A integração do MAS com o SUM foi dividida em duas camadas: agentes e artefatos. A camada de agente compreende a programação de agentes na linguagem Jason e interações their. Enquanto isso, a segunda camada apresenta os artefatos desenvolvidos no CArtAgO e as interações de seus agentes. Ambas as camadas são descritas detalhadamente nas seções a seguir.

## Camada de agente

Ao todo, três grupos de agentes compõem o MAS: *construtor, pspace, motorista*. A Figura 3 ilustra os agentes, crenças e interações (mensagens) do sistema.

O grupo *de agentes* possui os seguintes membros:

* *construtor*: ele é responsável por instanciar todos os artefatos CArtAgO, espaços de trabalho e agentes *pspace*.
* *pspace*: é capaz de representar a vaga de estacionamento, tanto física quanto virtual (em SUMÔ, como mostrado na Figura 2).

|  |
| --- |
| **Figura 3.** Diagrama de Prometeu - Agentes: crenças e mensagens |

* *motorista*: ele representa o motorista que quer obter uma vaga de estacionamento.

O agente *pspace* tem as seguintes crenças:

* *preço*: o preço para cada hora de uso da vaga de estacionamento;
* *status Alterado*: o valor correspondente à variação do sensor de distância (ultrassônico) (em uso ou livre);
* *posição*: a posição do caminho da rede em SUM (0 - uma via / 1 - regressão);
* *maxTime*: o período máximo de tempo que um motorista pode usar a vaga de estacionamento. Se o tempo acabar, haverá uma taxa extra no preço;
* *borda*: o valor que identifica a borda (rua) da rede SUMO em que a vaga de estacionamento está posicionada. Por padrão, o SUMO fornece valores editáveis pré-estabelecidos, como (ID de borda);
* *motorista de* espera: informa se a vaga de estacionamento está no estado *à espera de um motorista*. Essa crença é usada se o espaço de parking estiver ocupado sem ser previamente atribuído a um determinado driver. Assim, se isso ocorrer, a vaga de estacionamento foi indevidamente ocupada por um objeto ou motorista indesejado;
* *estacionado*: indica a presença de um motorista estacionado no estacionamento que foi atribuído ao motorista.

Entre o grupo de agentes, as seguintes mensagens podem ser trocadas:

* *requestPSpace*: o agente do *motorista* envia a mensagem ao agente *pspace* informando a solicitação de vaga de estacionamento por um preço predeterminado pela crença de *preço*;
* *systemReady*: mensagem enviada ao instanciar agentes *pspace*  e *espaços de trabalho* cartago e artefatos. Nesse processo, o agente *construtor* envia uma mensagem aos agentes *do pspace* informando que o sistema está funcionando. Assim, *os agentes do PSPACE* estão abertos a solicitações dos *agentes motoristas*;
* *psInfo*: quando o agente *do pspace* aceita uma solicitação de vaga de estacionamento, esse agente deve informar ao respectivo agente motorista as informações sobre a vaga de estacionamento atribuída (por exemplo, localização).

Para esclarecer, não apresentaremos os detalhes de como a vaga de estacionamento é solicitada e negociada. No entanto, o leitor pode encontrar soluções específicas de MAS para a negociação de vagas de estacionamento nas seguintes obras [12], [13], e [14].

## Camada de artefato - Conexão com SUMO

Os artefatos descritos neste scenario compreendem a interconexão de MAS com SUM, ilustrada na Figura 4. Com base na Figura 4, os artefatos usados para conectar o MAS ao SUMO são SUMOConnect, StructureInfo e PSControl. O artefato MQTTConnection é usado para integrar o MAS às ferramentas de IoT, o que é explicado na Seção 5.

Os seguintes artefatos são usados no sistema:

• *SUMOConnect:* O artefato é usado para se conectar ao SUMO. A conexão com o SUMO é feita através da biblioteca TraaS [[2]](#footnote-2) [15]. Existem duas operações que os agentes *pspace* podem realizar no artefato *SUMOConnect*.

|  |
| --- |
| **Figura 4.** Interação entre JaCaMo e SUMO |

* addVehicle: operação que adiciona veículos à rede SUMO. Os agentes *de motorista*  mas em JaCaMo são considerados veículos em SUMO, enquanto os agentes *pspace*  são os recursos do *Espaço de estacionamento* da SUMO. A lista 1 apresenta a função *addVehicle*. Os parâmetros da função são o nome do agente (identificação do veículo SUMO), o id rodoviário SUMO (*borda*) e a posição da estrada (0 - uma via, 1 - regressão). A função inicialmente (linha 5) adiciona o veículo à rede SUMO. Nesta linha, há também a *rotaToParking*, que é previamente definida como uma sequência de estradas utilizadas pelo veículo para encontrar a vaga de estacionamento para ele. Por fim, a linha 6 define qual vaga de estacionamento o veículo ocupará, o que é previamente negociado na camada de agente.

**Lista 1.** Função: addVehicle - CArtAgO

|  |
| --- |
| 1 **A\_SUMOConnect de classe pública**  **estende** artefato {  SumoTraciConnection conn;  3 **@OPERATION vazio público** addVehicle (**String** agentName, **String** edge, **double** position) {  5 conn.do\_job\_set(Veículo. **adicionar** (agentName",  DEFAULT\_VEHTYPE", routeToParking, pista  , posição, **velocidade,Byte.valueOf**("0")); conn.do\_job\_set(Vehicle.setParkingAreaStop  (agentName, parkingAreaName, duração, timeOut, flag);} |

* removerVehicle: Listar o Código 2 apresenta a função de remover o veículo da rede em SUMO. Depois que um agente *de motorista* termina usando um *pspace*, o veículo correspondente em SUMO é solicitado a sair do *estacionamentoArea*[[3]](#footnote-3) (Linha 3) e iniciar o trajeto para a saída de rede (routeToLeave) (Linha 4).

**Lista 2.** Função: removevehicle - CArtAgO

|  |
| --- |
| **@OPERATION**  2 **vazio público** removevehicle(**String** agentName) { conn.do\_job\_set(Vehicle.resume(agentName));  4 conn.do\_job\_set(Vehicle.setRoute(  agentName, routeToLeave));} |

Finalmente, no *SUMOConnect* há o uso do sinal *sumoFail* em caso de falha de conexão com *traci* e problemas com a rede SUMO.

• *EstruturaInfo e PSControl:* O artefato *PSControl* é usado para registro, alocação, liberação e controle de *pspaces*. Por sua vez, o artefato *StructureInfo* é usado pelo agente *construtor* para informar os outros agentes sobre o status MAS.

# Integração: Ambiente Físico eMAS

Esta seção apresenta o JaCaMo como um sistema embarcado que emprega um MAS que interage com os ambientes físicos. Há duas maneiras de interligar o mundo real: i. usando agentes que interface o ambiente físico diretamente; e ii. usando artefatos como intermediário entre agentes e o hardware.

O primeiro permite que os agentes acessem dados brutos de sensors e os processem como percepções diretamente em seu ciclo de raciocínio em cada ciclo realizado, e eles podem ativar atuadores enviando comandos para o hardware usando uma interface serial. Uma vez que o hardware é gerenciado por um agente autônomo e cognitivo, permite melhorar o processo de tomada de decisão na borda do sistema uma vez que as informações foram processadas e fundamentadas. No entanto, quanto mais dados o ambiente pode produzir, mais informações o agente lidará e seu ciclo de raciocínio poderá ser carregado na conversão em percepções. Algumas soluções foram implementadas para enfrentar esse problema, como filtros de percepção [16], que incorpora um mecanismo de percepção filtrante dentro do intérprete de Jason para eliminar as percepções irrelevantes e redobrar o tempo de processamento. Mesmo com o filtro de percepção, ainda é possível sobrecarregar o ciclo de raciocínio de um agente dependendo de como o filtro é projetado ou mesmo se novas informações se tornam disponíveis no ambiente considerando sua imprevisibilidade.

A segunda maneira de interagir com o mundo real é usar artefatos para se comunicar com microcontroladores no ambiente físico. Usando esses artefatos, os agentes não precisam acessar o ambiente físico em cada ciclo de raciocínio realizado e access os artefatos apenas quando é realmente necessário. Além de aliviar a responsabilidade do ciclo de raciocínio, o artefato também pode ser usado por outros agentes de MAS, permitindo que sensores e atuadores sejam acessados por qualquer agente de MAS.

Tanto as camadas de agente quanto de ambiente usam uma interface serial para acessar sensores e atuadores. O Javino [17] é um protocolo de comunicação bidirecional, que fornece confiabilidade, pois possui um processo para verificar a integridade da mensagem para garantir que não haja perda de informações durante o processo de comunicação entre os níveis baixo e alto.

## ARGO: Jason Agent e Integração de Hardware

A integração dos agentes Jason com hardware permite desenvolver MAS em ambientes físicos reais. A noção de autonomia, pró-atividade e colaboração por parte de um agente mas justifica a necessidade de integrar esse sistema em um ambiente real para lidar com a imprevisibilidade desse ambiente físico.

ARGO [9] é uma arquitetura de agente personalizada para se comunicar com microcontroladores na linguagem de programação AgentSpeak. Os agentes que usam essa arquitetura são chamados de agentes ARGO e permitem o desenvolvimento de um MAS interagindo em um ambiente físico através de microcontroladores Arduino, ESP ou PIC. A ARGO usa a interface de comunicação serial javino como um middleware para estabelecer a comunicação entre o hardware e o MAS.

Portanto, com a interface serial javino, um agente ARGO pode enviar e receber informações de microcontroladores. As informações recebidas são tratadas como percepções que o agente ARGO recebe do ambiente através de sensores do hardware que o agente está controlling e, portanto, são automaticamente adicionadas como crenças na base de crenças deste agente. As informações enviadas ao microcontrolador são tratadas como ações realizadas no ambiente físico por meio de atuadores de hardware.

Para o uso dessas características dos agentes argo, quatro novas ações internas (*ou seja*, ações ou comportamentos pré-programados inerentes ao agente [7]) foram desenvolvidas exclusivamente para controlar microcontroladores, ou seja:

* *.porta("Porta serial")*: define qual porta serial o agente irá controlar e, consequentemente, qual microcontrolador o agente irá controlar. Esta ação interna tem um argumento que representa qual porta serial o agente irá controlar;
* *.perceive(aberto/bloco)*: define se o agente vai perceber o ambiente ou não. Essa ação interna argumenta e tem duas opções: *abrir*, usar para fazer o agente abrir percepções e receber informações do hardware; e *bloquear* usado para fechar as percepções do agente e não receber mais informações;
* *.limite(Tempo em milissegundos)*: define um intervalo de tempo para o agente perceber o ambiente. Esta ação interna tem um argumento que representa o tempo em milissegundos de que o agente mudará a percepção de aberta para fechada;
* *.act("Ação")*: define uma ação que o microcontrolador deve realizar. Essa ação internal tem um argumento que representa a ação que deve ser enviada ao microcontrolador para realizar.

Além dos agentes ARGO, existem também artefatos físicos que permitem integrar artefatos convencionais da estrutura CArTAgO com dispositivos físicos e seus microcontroladores [18].

## Artefatos Físicos: CArtAgO e Integração de Hardware

A arquitetura JaCaMo [6] apresenta a diferença entre dois ambientes: o ambiente interno, onde os artefatos são logicamente organizados, e o ambiente externo, que pode ser representado por cenários simulados ou reais. Ao adotar um ambiente físico composto por tecnologias de hardware representadas como artefatos, é possível desenvolver sistemas cibernéticos físicos (CPS) empregando agentes usando nossa abordagem proposta.

Artefatos podem ser usados para interfacear o hardware no mundo real, já que qualquer agente do MAS pode apescá-los se estiverem disponíveis. Além disso, os agentes não precisam coletar informações dos sensores e processá-la como percepção em seu ciclo de raciocínio, o que sobrecarrega os agentes dependendo do domínio do aplicativo. Além disso, quando os agentes interagem tais artefatos, não precisam processar as informações coletadas em cada ciclo e solicitam sempre que forem necessárias e disponíveis.

Uma vantagem desses artefatos é a abstração dos detalhes técnicos do hardware empregado desde que os agentes acessam suas operações fornecidas pela interface de uso. Um artefato responsável por interligar hardware em um ambiente físico é chamado artefato físico neste artigo. Ele troca comandos e dados entre hardware e agentes, controla os atuadores e monitora os sensores deste dispositivo.

O Artefato Físico é uma extensão de artefatos tradicionais da CArtAgO e também possui (1) um conjunto de operações que podem ser realizadas por agentes, (2) instruções que descrevem como esses artefatos devem ter sua functionalidade acessada, (3) a finalidade da existência e (4) as estruturas internas de suas funcionalidades [8 ]. No CArtAgO, os métodos chamam de Operações, que determinam o comportamento do artefato, e propriedades observáveis permitem que o artefato notifique o agente de qualquer evento do ambiente. Assim, os agentes podem obter dados de um Artefato Físico usando Propriedades Observáveis, e podem executar comandos em atuadores usando Operações. Os Artefatos Físicos também usam Javino para interligar o microcontroller do dispositivo. Assim, é necessário implementar três métodos abstratos no projeto:

* *String definePort()*: retorna o valor da porta de comunicação a ser usada para trocar dados com o microcontrolador;
* *int defineAttemptsAfterFailure()*: retorna o número de times que o artefato tentará enviar a mesma mensagem quando um erro ocorreu durante a troca de mensagens;
* *int defineWaitTimeout()*: retorna o tempo de espera em milissegundos entre duas solicitações feitas ao dispositivo.

Quando necessário implementar Operations e Propriedades Observáveis de Artefatos Físicos, dois métodos podem ser empregados:

* *Leitura de string()*: retorna uma mensagem enviada pelo microcontrolador do dispositivo, geralmente mostrando as medições coletadas por seus recursos;
* *vazio enviar (mensagem de string)*: envia o mago passado por parâmetro para o dispositivo, geralmente apresentando um comando para operar os recursos deste dispositivo.

# Integração: Aplicações de IoT e MAS

A abordagem proposta até agora mostrou como integrar o MAS com o ambiente externo, seja físico ou simulado. Embora a abordagem ofereça esses mecanismos, ainda há uma lacuna quando a MAS precisa se comunicar com pares ou com outras entidades. Se sua comunicação se tornasse possível, o MAS e outras entidades inseridas em ambientes semelhantes ou diferentes poderiam compartilhar seus conhecimentos e otimizar processos.

Nesse contexto, a Internet das Coisas (IoT) promete que dispositivos e clientes podem estabelecer a comunicação como um nó de rede em larga escala. No entanto, um dos desafios da IoT é garantir que dispositivos de diferentes tipos se comuniquem na mesma língua e de forma segura e escalável.

Para desenvolver tal característica, comunicando o MAS com outras entidades de IoT através da rede, projetamos diferentes módulos que, juntos, proporcionam interação entre plataformas MAS e IoT. A Figura 5 apresenta esses módulos, bem como suas conexões e interações sequenciais, que são descritas abaixo.

* IoT Prototype: O protótipo consiste em três itens: Dois

Arduino (Um Arduino MEGA e um Arduino UNO), e um Raspberry Pi 3. Inicialmente, o Arduino realiza a leitura e envia os dados do sensor (distância ultrassônica) para os agentes pspace em MAS. O Raspberry Pi 3 é responsável por hospedar o MAS, o aplicativo NODE-RED, o Influx Database e o Grafana Server e seus Dashboards;

* MAS: O MAS é composto pelos seguintes agentes e artefatos para conectar o MAS às Ferramentas de IoT;

– *Artefato: Conexão MQTT* É capaz de estabilish uma conexão com a Plataforma IoT (MQTT Broker) a fim de notificá-lo sobre os sensores pspace e drivers. A comunicação entre o MQTTConnection e a Plataforma IoT é fornecida pelo Eclipse Paho[[4]](#footnote-4). Este artefato é composto por três operações, que são:

* *connect2Broker:* Esta operação é capaz de conectar o MAS ao Corretor MQTT (ver Listagem 3). Inicialmente nas linhas 4-8, há configuração para atribuir o id do dispositivo, url e porta do corretor, e o usuário e senha. Depois, na linha 10, o MAS tenta se conectar.

**Lista 3.** Operação:connect2Broker - CArtAgO

|  |
| --- |
| **@OPERATION**  2 **void** connect2MQTTBroker() { mqttClient = **novo** MqttClient(MQTT\_Config.BROKER .getValue(), MQTT\_Config.DEVICE\_ID.  getValue(), persistência);  4 cOptions = **novas** Opções MqttConnect(); cOptions.setCleanSession(**true**);  6 cOptions.setPassword(MQTT\_Config.PASS.getValue  ().toCharArray()); cOptions.setUserName(MQTT\_Config.USER.getValue  ());  8 **try** {  mqttClient.connect(cOptions);  10System.out.println("Conectando ao corretor:  " + MQTT\_Config.BROKER.getValue()); |

* *sendMSG2Broker:* Após uma conexão bem sucedida com o Corretor, o MAS envia uma mensagem ao Corretor com um valor de sensor atualizado se algum sensor tiver seu valor alterado. A mensagem é baseada no formato JSON devido à carga required do Broker. A mensagem JSON é composta de dois campos: valor e unidade. Como usamos sensores ultrassônicos, a unidade é centímetros.

Esta operação é apresentada na Lista 4. Primeiro, na linha 1 é o valor do sensor está sendo lançado como uma mensagem JSON. Depois na linha 2 é a configuração QoS e finalmente na linha 3 é a publicação da mensagem JSON no MQTT

Corretor.

**Lista 4.** Operação: sendMSG2Broker - CArtAgO

|  |
| --- |
| Mensagem MqttMessage = **novo** MqttMessage(**este**. string2JSON(msg, "cm").getBytes());  2 message.setQos(Integer.valueOf(MQTT\_Config.QOS  .getValue()); mqttClient.publish(MQTT\_Config.PUBLISH\_TOPIC.  getValue(), mensagem); |

|  |
| --- |
| **MAS**  **Agentes**  **Protótipo de IoT**  **Plataforma IoT**  **InfluxDB**  **Base de dados**  **Grafana**  **Painel**  Serial  foto a ser inserida  **ARGO**  **Artefatos**  **SUMOConnect**  **MQTT\_Connection**  **Comunicador**  **PSpace**  **Construtor**  **Motorista**  **EstruturaInfo**  **PSControl**  MQTT  MQTT  **Aplicativo nodeRED**  HTTP  HTTP  **Figura 5.** Integração: Aplicações protótipo, MAS, plataforma e IoT |

* *Agente ARGO*: estabelece a conexão com o protótipo IoT para perceber o ambiente físico através de sensores de distância ultrassônica. Ao obter as percepções, o agente ARGO notifica o artefato MQTTConnection e o agente comunicador se houver mudança de ambiente (mudança significativa nos valores do sensor). Em nosso trabalho, os agentes pspace são considerados aos agentes argo, uma vez que os agentes pspace têm que perceber o ambiente físico para verificar se algo ocorreu;
* *Agente comunicador (ContextNET)*: o agente é responsável por se comunicar com outro MAS através da rede ContextNET. Usando este agent, o MAS pode comunicar-se com outros MAS para a cooperação múltipla entre os sistemas. Por exemplo, uma rede de estacionamentos inteligentes disponíveis em vários locais da mesma cidade. Assim, um único estacionamento pode fornecer vagas de estacionamento.
* Plataforma IoT: a plataforma IoT é uma Corretora MQTT que visa integrar diferentes dispositivos IoT através de rotas MQTT (publicar/assinar). Ao receber as mensagens através do MQTT do protótipo, a plataforma redireciona as mensagens via MQTT para NodeRED. A plataforma utilizada é a fornecida pelo KonkerLabs. Além disso, existem outras soluções fornecidas pelo Eclipse, Google e outros;
* NodeRED: usado para decomposição e extração de mensagens JSON da plataforma IoT e armazenamento no InfluxDB via HTTP. A decomposição e extração é usada para obter o valor dos sensores protótipo;
* InfluxDB: Bancos de dados da série time (TSDB). O uso desse tipo de banco de dados deve-se à sua característica de ter um data-hora como sua chave primária. Assim, o armazenamento dos valores do sensor é realizado de acordo com o tempo de medição;
* *Dashboard* Grafana: O painel grafana fornece diferentes componentes para visualização de dados em tempo real. O painel de instrumentos acessa os dados do sensor via HTTP no banco de dados InfluxDB.

## O ContextNet

O ContextNet [11] é um middleware created baseado na Internet das Coisas (IoT) e serviço de contexto que se concentra em aplicativos colaborativos, coordenação de atividades de entidades móveis e compartilhamento de informações. Entidades móveis podem ser dispositivos móveis, como veículos, smartphones, tablets, laptops ou até robôs autônomos conectados à rede IoT. Entre as tecnologias utilizadas para o desenvolvimento do ContextNet, a Scalable Data Distribution Layer (SDDL) é um middleware incorporado no ContextNet responsável pela capacidade de se comunicar e distribuir contexto. As outras características do ContextNet foram desenvolvidas em módulos de software no topo da camada de distribuição. Além disso, o ContextNet permite que os dispositivos entrem na rede e os tratem adequadamente.

Além da integração do MAS com o ambiente físico e simulação urbana, nossa abordagem também permite que a MAS seja integrada com a IoT. Assim, cada sistema incorporado pode se comunicar com outro sistema incorporado ou mesmo com outros nós na rede ContextNet.

## O Agente Comunicador

O Agente Transmissor [19] é uma extensão de agente que implementa um nó IoT usando ContextNet. Com isso, o Agente Comunicador pode se comunicar com entidades distribuídas pela rede, não importando que sejam sensores, atuadores, clientes web ou até mesmo outros Agents comunicadores. Cada Agente Comunicador possui um Identificador Universalmente Único (UUID), que deve ser conhecido pela entidade que deseja trocar mensagens. Ao enviar mensagens, os Agentes Comunicadores podem definir a força illocutiva como *dizer* ou *alcançar*. A primeira é escolher a base de crenças do agente receptor, e a segunda é atualizar a biblioteca do plano do agente receptor.

A integração entre MAS e simulação urbana utilizando o mapa UTFPR (Ponta Grossa) (ver anteriormente na Figura 2) também pode ser feita com Agentes Comunicadores. Neste caso, *pspace* e *motorista* tornam-se Agentes Comunicadores que se comunicam através do ContextNet para negociar vagas de estacionamento. Com isso, esses agentes podem até estar em DIFERENTES MAS, o que permite ampliar o alcance mas que realiza a negociação.

O Agente Communicator tem uma ação interna chamada *sendOut* que toma como parâmetro o UUID do destinatário, a força elocucionária da mensagem (alcançar ou dizer) e a mensagem. As mensagens enviadas entre os Agentes Comunicadores são dadas da seguinte forma:

* *requestPSpace*: enviado de motorista agente para agente *pspace* ligando para *sendOut*. Os parâmetros são o UUID do pspace do agente, a mensagem de força elocucionária e a mensagem de solicitação de vaga de estacionamento. Exemplo:

*.sendOut ("788b2b22-baa6-4c61-b1bb-01cff1f5f879", alcançar, solicitarPSpace(MT))*, onde *solicitaçãoSpot(MT)* é a mensagem de solicitação.

* *psInfo*: enviado do agente *pspace* para o motorista do agente ligando para *sendOut*. Os parâmetros são o UUID do motorista do agente, a força elocucionária *e* a mensagem com as informações do estacionamento.

# Conclusão

Este trabalho apresenta uma abordagem para a integração de um MAS desenvolvido em JaCaMo com ferramentas de Simulação Urbana, Sistemas Embarcados e Aplicações IoT. O objetivo principal é implantar uma variedade de ferramentas e sistemas de forma que um forte acoplamento entre todos os níveis seja evitado.

A vantagem dessa abordagem é sua generalização. Os níveis propostos são heterogêneos, permitindo que diferentes aplicações usem este approach como suporte para a integração de sistemas. Outra vantagem é o baixo acoplamento entre as camadas, o que dá flexibilidade para usar tanto a arquitetura totalmente integrada quanto apenas as peças. Através deste trabalho, problemas que exigem escalabilidade e um sistema inteligente podem ser resolvidos com IoT integrado em um MAS. Ainda é possível escolher qual tipo de ambiente deve ser usado: simulado ou físico.

Como trabalho futuro, pretendemos desenvolver uma metodologia de integração como estrutura para soluções integradas de mobilidade urbana MAS, IoT, Simulação e Sistemas Embarcados. Também é necessário implantar um aplicativo (robusto e complexo) no domínio de Smart Cities, que utiliza todos os níveis propostos. Com essa aplicação, também será possível executar experiments para avaliar as características da metodologia de integração, como escalabilidade, baixo acoplamento, confiabilidade na troca de mensagens e informações, entre outras. Por fim, também pretendemos desenvolver a camada de organização social usando Moise (JaCaMo). Assim, será possível estabelecer regras e comportamentos nas sociedades de agentes, promover uma organização social e criar grupos sociais de agentes; esses elementos podem representar adequadamente cidades inteligentes e recursos de mobilidade urbana.

# Agradecimentos

O work do primeiro autor é apoiado pelo Coordenac İao de Aperfeic İoamento de Pessoal de N'ıvel Superior - Brasil (CAPES) - Código Financeiro 001. O autor agradece o apoio financeiro.

# Contribuições autorais

* Conceituação: Todos os autores.
* Analise Formal: Todos os autores.
* Investigação: Todos os autores.
* Practical experiments: Lucas Fernando Souza de Castro, Fabian Cesar P. B. Manoel, Vin´ıcius Souza de Jesus.
* Supervision: Lucas Fernando Souza de Castro, Gleifer Vaz Alves.
* Writing-original draft: Lucas Fernando Souza de Castro, Fabian Cesar P. B. Manoel, Vin´ıcius Souza de Jesus.
* Redação-revisão e edição: Todos os autores.

# Referências

1. EVANS, D. Como a Próxima Evolução da Internet

As coisas estão mudando tudo. *Cisco Internet Business Solutions Group*, p. 11, 2011.

1. ALBINO, V.; BERARDI, U.; DANGELICO, R.M.

Cidades inteligentes: Definições, dimensões, desempenho e iniciativas. *Journal of urban technology*, Taylor & Francis, v. 22, n. 1, p. 3-21, 2015.

1. NEIROTTI, P. et al. Trends atuais em iniciativas de cidades inteligentes: Alguns fatos estilizados. *Cidades*, v. 38, p. 25 – 36, 2014. Dispon'ıvel em: [h](http://www.sciencedirect.com/science/article/pii/S0264275113001935) [http://www.sciencedirect.com/science/ artigo/pii/S0264275113001935i.](http://www.sciencedirect.com/science/article/pii/S0264275113001935)
2. WOOLDRIDGE, M.J. *Raciocínio sobre agentes racionais*. [S.l.]: imprensa do MIT, 2000.
3. ZHANG, D. et al. Internet das coisas. *J. UCS*, v. 18, p. 1069-1071, 2012.
4. BOISSIER, O. et al. Programação orientada multi-agente com jacamo. *Ciência da Programação computacional*, Elsevier, v. 78, n. 6, p. 747-761, 2013.
5. BORDINI, R. H.; HuBNER, J.F.; WOOLDRIDGE, ̈ M. *Programação Sistemas Multi-Agentes em Agent Speak usando Jason*. John Wiley & Sons Ltd, 2007. 273 p.
6. RICCI, A.; VIROLI, M.; OMICINI, A. Programação

MAS com artefatos. v. 3862 LNAI, p. 206-221, 2006. Dispon'ıvel em: [h](https://www.scopus.com/inward/record.uri?eid=2-s2.0-33745661012&doi=10.1007%2f11678823_13&partnerID=40&md5=6c8f34aae1c78bccd47fabd2c3705e5e) [https://www.scopus.com/ para dentro/record.uri?eid=2-s2.0-33745661012&doi=](https://www.scopus.com/inward/record.uri?eid=2-s2.0-33745661012&doi=10.1007%2f11678823_13&partnerID=40&md5=6c8f34aae1c78bccd47fabd2c3705e5e)

[10.1007%2f11678823 13&partnerID=40&md5= 6c8f34aae1c78bccd47fabd2c3705e5e5ei.](https://www.scopus.com/inward/record.uri?eid=2-s2.0-33745661012&doi=10.1007%2f11678823_13&partnerID=40&md5=6c8f34aae1c78bccd47fabd2c3705e5e)

1. PANTOJA, C. E. et al. Argo: Uma arquitetura jason estendida que facilita a programação de agentes robóticos incorporados. In: SPRINGER. *Workshop Internacional de Engenharia Multi-Agente Sistemas*. [S.l.], 2016. p. 136-155.
2. KRAJZEWICZ, D. et al. Sumo (simulação de urbano

mobilidade)-uma simulação de tráfego de código aberto. In: *Proceedings of the 4th Middle East Symposium on Simulation and Modeling (MESM20002)*. [S.l.: s.n.], 2002. p. 183-187.

1. ENDLER, M. et al. Contextnet: contexto reassoning e compartilhamento de middleware para colaboração e redes sociais em larga escala. In: ACM. *Procedimentos do*

*Oficina sobre Pôsteres e Trilha de Demos*. [S.L.], 2011. p. 2.

1. CASTRO, L.F. S.D.; ALVES, G. V.; BORGES, A.P.

Usando o grau de confiança para agentes, a fim de atribuir vagas em um Estacionamento Inteligente. *ADCAIJ: Avanços em Revista de Computação Distribuída e Inteligência Artificial*, v. 6, n. 2, p. 45-55, jun. 2017. Dispon'ıvel em: [h](http://revistas.usal.es/index.php/2255-2863/article/view/ADCAIJ2017624555) [http://revistas.usal.es/index.php/ 2255-2863/article/view/ADCAIJ2017624555i.](http://revistas.usal.es/index.php/2255-2863/article/view/ADCAIJ2017624555)

1. DUCHEIKO, F.F.; ANDRe, P.B.; GLEIFER, V. A.

Implementac¸ao de Modelo de Racioc˜ ´ınio e Protocolo de

Negociac¸ao para um Estacionamento Inteligente com˜

Mecanismo de Negociac¸ao Descentralizado.˜ *Revista*

*Junior de Iniciac¸ao Cient˜ ´ıfica em Ciencias Exatas eˆ Engenharia*, v. 1, n. 19, p. 25–32, 2018. Dispon´ıvel em: [hhttp://www.icceeg.c3.furg.br/index.php?Itemid=837&](http://www.icceeg.c3.furg.br/index.php?Itemid=837&option=bloco_texto&id_site_componente=1241)  [option=bloco texto&id site componente=1241i.](http://www.icceeg.c3.furg.br/index.php?Itemid=837&option=bloco_texto&id_site_componente=1241)

1. ALVES, B.R. et al. Experimentação de Protocolos de Negociação para Problemas de Consenso em Sistemas de Estacionamento Inteligente. In: MARIK, V. et al. (Ed.). *Aplicações Industriais de Sistemas Holônicos e Multi-Agentes*. Cham: Springer International Publishing, 2019. (Notas de Palestra em Ciência da Computação), p. 189-202.
2. KRUMNOW, M. Sumo como um serviço - construindo um serviço web para interagir com o sumô. In: SPRINGER. *Simulação da Conferência de Usuários de MObilidade Urbana*. [S.L.], 2013. p. 62-70.
3. STABILE Jr., M. F.; PANTOJA, C. E.; SICHMAN, J. S.

Análise experimental do efeito da filtragem de percepções em agentes do BDI. *International Journal of Agent-Oriented Software Engineering*, Inderscience Publishers (IEL), v. 6, n. 3-4, p. 329-368, 2018.

1. LAZARIN, N.M.; PANTOJA, C.E. Uma plataforma de agente robótico para incorporar agentes de software usando placas Raspberry Pi e Arduino. In: 9º*Programa agentes, ambientes e aplicativos escola*. [S.l.: s.n.], 2015.
2. MANOEL, F.C. P.B. et al. Artefatos físicos para agentes em um sistema cibernético-fisical: Um estudo de caso no cenário de óleo & gás (EEAS). In: GARC'IA-CASTRO, R.

(Ed.). *A 32ª Conferência Internacional de Software*

*Engenharia de Engenharia e Conhecimento, SEKE 2020,*

*KSIR Virtual Conference Center, EUA, 9 a 19 de julho de 2020*.

KSI Research Inc., 2020. p. 55-60. Dispon'ıvel em: [h](https://doi.org/10.18293/SEKE2020-154) <https://doi.org/10.18293/SEKE2020-154> [i.](https://doi.org/10.18293/SEKE2020-154)

1. PANTOJA, C. E. et al. Uma arquitetura para o desenvolvimento de sistemas de inteligência ambiental gerenciados por agentes incorporados. In: *SEKE*. [S.l.: s.n.], 2018. p. 215-214.

1. O termo rede em SUM é usado para denotar um mapa usado na simulação. [↑](#footnote-ref-1)
2. Biblioteca baseada em Python que se comunica com o SUMO TraCI (conexão com *Soquete*). [↑](#footnote-ref-2)
3. O lugar onde o motorista estaciona o carro que chamamos de vaga de estacionamento em this papel. No entanto, o SUMO chama-o de estacionamentoArea em sua ferramenta. [↑](#footnote-ref-3)
4. Esta é a biblioteca da Fundação Eclipse que fornece conexões MQTT (publicar/assinar) entre o cliente e o corretor MQTT [↑](#footnote-ref-4)