Anais do XVII Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações
- WESAAC 2023 -

Organizado por
Marilton Sanchotene de Aguiar
Diana Francisca Adamatti

Universidade Federal de Pelotas
Pelotas, de 30 de agosto a 01 de setembro de 2023.

Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações
(17. : 2023 : Pelotas)
Anais do XVI Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações WESAAC 2023 [recurso eletrônico] / organizado
por Marilton Sanchotene de Aguiar, Diana Francisca Adamatti. – Pelotas : UFPel: 2023.
158 p. : il.
E-book (PDF)
ISBN 978-85-8328-127-6
1. Agentes inteligentes (Software) – Congressos. 2. Sistemas multiagentes. 3. Inteligência artificial. I. Aguiar, Marilton Sanchotene de, org. II. Adamatti, Diana Francisca, org. IV.
Título.

Prefácio

Este documento contém os trabalhos apresentados na décima sétima edição do WESAAC (Workshop Escola de Sistemas de Agentes, seus Ambientes e Aplicações), realizado em Pelotas, na Universidade Federal de Pelotas entre os dias 3 de agosto a 1 de setembro de 2023.

Na edição deste ano, contamos com três palestras: a primeira da professora Celia Ghedini Ralha, da Universidade de Brasília, sobre agentes e gestão de recursos naturais. A segunda, do professor Luis Gustavo Nardin, da cole Nationale Supérieure des Mines de Saint-Étienne, sobre regulação de agentes. E a terceira, da professora Rejane Frozza da Universidade de Santa Cruz do Sul, sobre assistentes pessoais e agentes.

Também contamos com três oficinais/minicursos. No primeiro, os professores Giovani Parente Farias e Miriam Blank Born, da Universidade de Pelotas, apresentaram a ferramenta GAMA, para simulação de recursos naturais. No segundo, o professor Maiquel de Brito, na Universidade Federal de Santa Catarina, tratou sobre organização e o modelo Moise+. No terceiro, professores Nilson Mori Lazarin e Carlos Eduardo Pantoja, Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, abordaram sobre sistemas multiagente embarcados e distribuídos.

E tivemos um panel, que tratou sobre a área de agentes, reflexões, avanços e perspectivas no contexto nacional e internacional. Este panel contou com os professores Antonio Carlos da Rocha Costa, Rafael Heitor Bordini (Pontifícia Universidade Católica do Rio Grande do Sul) e Marilton Sanchotene de Aguiar (Universidade Federal de Pelotas.

Nas sessões técnicas, tivemos a apresentação de 16 artigos científicos, com temas mais diversos e interdisciplinares, que mostrarm a ampla gama de aplicações da área de agentes. Agradecemos ao comitê de programa que auxiliou em todo o processo de revisão e incremento na qualidade dos trabalhos aceitos e apresentados.

Também gostaríamos de agradecer as nossas universidades, por disponibilizarem infraestrutura e tempo para organização deste evento.

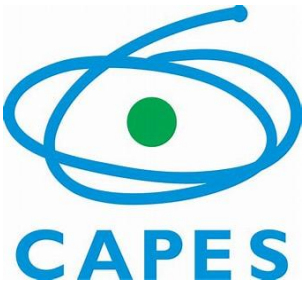
Pelotas/RS, Agosto de 2023.

Diana Francisca Adamatti
Universidade Federal do Rio Grande

Marilton Sanchotene de Aguiar
Universidade Federal de Pelotas

Apoio


UNIVERSIDADE FEDERAL DE PELOTAS
RS-BRASIL


CAPES


ANA
AGÊNCIA NACIONAL DE ÁGUAS
E SANEAMENTO BÁSICO

Organização

Organização Geral: Marilton Sanchotene de Aguiar (UFPel)

Coordenação do Comitê de Programa: Diana Francisca Adamatti (FURG)

Comitê consultivo
Anarosa Alves Franco Brandão (USP)
Carlos Eduardo Pantoja (CEFET-RJ)
Diana Francisca Adamatti (FURG)
Gleifer Vaz Alves (UTFPR)
Gustavo Alberto Giménez Lugo (UTFPR)
Jaime Sichman (USP)
Jerusa Marchi (UFSC)
JoãoLuis Tavares da Silva (UCS)
Jomi Fred Hübner (UFSC)
Mariela Inés Cortés (UECE)
Rafael Heitor Bordini (PUCRS)
Rejane Frozza (UNISC)
Ricardo Choren (IME)
Viviane Torres da Silva (UFF)

Comitê de Programa
Alessandro Ricci (Universidade de Bolonha - Itália)
Ana Paula Lemke (IFRS)
André Pinz Borges (UTFPR)
Antônio Carlos da Rocha Costa (PPGFil/PUCRS)
Carlos Eduardo Pantoja (CEFET/RJ)
Diana Francisca Adamatti (FURG)
Eder Mateus Gonçalves (FURG)
Fernando De La Prieta (Universidade de Salamanca - Espanha)
Fernando Santos (UDESC)
Gleifer Alves (UTFPR)
Gustavo Giménez-Lugo (UTFPR)
Jerusa Marchi (UFSC)
Jomi Fred Hübner (UFSC)
Luis Gustavo Nardin (Mines Saint-Étienne - França)
Maicon Rafael Zatelli (UFSC)
Maiquel de Brito (UFSC)
Marilton Aguiar (UFPEL)
Rafael C. Cardoso (Universidade de Liverpool - Inglaterra)
Ricardo Azambuja Silveira (UFSC)
Ricardo Choren (IME/RJ)
Sara Casare (USP)
Tiago Luiz Schmitz (UDESC)

Sumário

Uma proposta de políticas de migração para sociedade de agentes, Vinicius Machado Pinto, Nicolas da Silva Jatobá and Nilson Lazarin	8
Uma análise da segurança nas comunicações entre agentes inteligentes, de Vitor Sobrinho da Fonseca and Nilson Lazarin	14
Uma arquitetura baseada em Docker para o desenvolvimento de sistemas multiagentes abertos, de Gustavo de Lima and Marilton Aguiar	20
Um Mapeamento de Sistemas Multiagentes e Ativos Digitais com uma Proposta de Aplicação em Smart Parking, de Gabriel Oliveira, Gleifer Alves and Nilson Lazarin	32
The Tupinambá: An Exercise in Societal Modeling, de Antonio Carlos Rocha Costa	44
Uma Proposta de Emulador de Portas Seriais para Sistemas Multiagentes Embarcados, de Bruno Policarpo Toledo Freitas, Nilson Mori Lazarin and Carlos Eduardo Pantoja	55
Coordenação de robôs ROS com Agentes BDI e Moise, de Pedro Henrique Dias Do Nascimento and Maiquel de Brito	67
Desenvolvimento de um protótipo de braço robótico integrado a um agente BDI, de Pedro Kiyoshi Bezerra Kano and Maiquel de Brito	79
Identificação dos Perfis de Agentes de RPG Mediante Análise de suas Estratégias pelo DSC: Um Estudo de Caso no Contexto de Gerenciamento de Recursos Naturais, de Fernanda Mota, Miriam Born, Marilton Aguiar and Diana Adamatti	85
Improving the Mapping of Moise+ to Colored Petri Nets, de Ricardo Machado, Diana Adamatti and Eder Mateus Gonçalves	91
Uma Ferramenta para mapear o Modelo Organizacional Moise+ em Rede de Petri de Forma Automatizada Utilizando Parser, de Arthur Zelindro, Ricardo Arend, Eder Golçalves and Diana Adamatti	101
MASPY: Towards the Creation of BDI Multi-Agent Systems, de Alexandre Mellado, Igor Guilhermer Fidler, André Pinz Borges and Gleifer Alves	106
Benchmarking Scalability of Message Transport Systems in the JADE Platform: Experimental Evaluation and Performance Analysis, de Luis Felipe Ferin Sgursky, Arthur Casals and Anarosa Alves Franco Brandão	118
Uma Proposta de Mapeamento do Ambiente Exógeno de Sistemas Multi-Agentes Usando Visão Computacional, de Douglas Bernardino, Leandro Botelho and Carlos Pantoja	130

Modelagem da Estrutura Funcional do Sistema Multiagente associado a um Jogo RPG no contexto de Recursos Hídricos, de Míriam Blank Born, Marilton Sanchotene de Aguiar and Diana Adamatti 136

Autocuidado de indivíduos com diabetes mellitus com apoio de Agente Conversacional, de Mateus Elias Gundel, Jordana Kich, Rejane Frozza, Andreia Rosane de Moura Valim, Janine Koepp and Lia Gonçalves Possuelo 148

Uma proposta de políticas de migração para sociedade de agentes

Vinicius Machado Pinto, Nicolas da Silva Jatoba, Nilson Mori Lazarin

[1]Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (Cefet/RJ)
Nova Friburgo, RJ – Brasil

{vinicius.pinto,nicolas.jatoba}@aluno.cefet-rj.br

nilson.lazarin@cefet-rj.br

Abstract. *A Multi-Agent System (MAS) is a society of agents that share the same environment and act autonomously to achieve their own goals, being able to cooperate or compete. Some of these societies may be open, and their agents may come and go freely. However, freely migrating agents can present security risks to the SMA, as an agent can be malicious. This work presents an approach that contributes to migration control, aiming to prevent communication or access by unauthorized agents. To this end, we propose an extension of the communicator agents architecture, implementing a firewall model for MAS.*

Resumo. *Um Sistema Multiagentes (SMA) é uma sociedade de agentes que compartilham um mesmo ambiente e agem de forma autônoma para atingir seus próprios objetivos, podendo cooperar ou competir. Essas sociedades podem ser abertas e seus agentes podem entrar e sair livremente. Entretanto, a livre migração de agentes pode apresentar riscos à segurança do SMA, visto que um agente pode ser mal-intencionado. Este trabalho apresenta uma abordagem para o controle migratório, visando impedir a comunicação ou acesso de agentes não autorizados. Para tal, propomos uma extensão da arquitetura dos agentes comunicadores, implementando um modelo firewall para SMA.*

1. Introdução

Sistemas Multiagentes (SMA) são uma classe de sistemas compostos por entidades autônomas, chamadas agentes, que compartilham um ambiente e interagem entre si para alcançar objetivos individuais ou coletivos. Esses agentes podem cooperar ou competir, criando uma dinâmica complexa e adaptativa dentro do SMA. No caso de SMA abertos, os agentes estão livres para se movimentar, criando desafios de segurança significativos para o SMA. A invasão de agentes maliciosos ou a ocorrência de acessos não autorizados podem comprometer o bom funcionamento do sistema e afetar a confiabilidade e integridade das interações entre os agentes [Hubner 1995, Alvares and Sichman 1997, Vila et al. 2007].

Este trabalho propõe uma abordagem para prevenir acessos indesejados e agentes maliciosos com base nas políticas de migração da sociedade dos agentes. Para isso, é proposta uma extensão para a arquitetura de *Agentes Comunicadores* [Jesus et al. 2018], através da implementação de um modelo de firewall para controlar a migração e a comunicação do agente para outro sistema multiagente.

O modelo proposto busca controlar a migração e a comunicação do agente para outro SMA, por meio de regras e políticas de segurança definidas. Dessa forma, o *Agente Comunicador* pode examinar e verifica a identificação da origem e seus privilégios, antes de permitir que eles entrem ou se comuniquem.

Este artigo está organizado da seguinte forma: na Seção 2 são apresentados os conceitos básicos necessários para o entendimento da proposta; na Seção 3 são apresentados e comentados alguns trabalhos relacionados; A metodologia proposta neste trabalho é apresentada na Seção 4; por fim, os resultados esperados são apresentados na Seção 5.

2. Fundamentação Teórica

Os denominados *Agentes Comunicadores* são uma extensão da arquitetura de agentes Jason [Bordini et al. 2007] capazes de se comunicar com outros SMA, além de permitirem a migração de agentes entre SMA distintos, através do ContextNet [Endler et al. 2011], um gateway IoT (*Internet of things*). Nesta arquitetura, os protocolos permitem migrar um SMA inteiro ou de agentes específicos conforme a relação que será estabelecida com o SMA de destino, sendo elas baseadas nas relações ecológicas: Inquilinismo, Mutualismo e Predatismo. No Inquilinismo, ocorre a transferência de um SMA inteiro para outro, visando se tornar apenas um com o destino; no Mutualismo, o objetivo é transmitir e obter novos conhecimentos e depois retornar ao SMA de origem; e no Predatismo, o intuito é dominar o SMA de destino, preservando seus agentes [Jesus et al. 2018].

Um *firewall* atua como uma espécie de barreira, deixando apenas determinadas comunicações seguirem, através das regras e politicas, previamente definidas. Um firewall podem ser baseado em hardware ou em software. Os baseados em hardware são dispositivos externos que atuam normalmente no ponto de conexão com a internet, podendo dar suporte a uma pequena rede local ou até mesmo em uma rede corporativa agindo como um concentrador/comutador (hub/switch) de rede. Um firewall baseado em software é normalmente projetado para trabalhar com sistemas operacionais específicos. Estes firewalls após a instalação normalmente vem com seu próprio conjunto de políticas predefinidas, políticas que permitem especificar qual nível de segurança se deseja obter [Ford 2002].

Nesta proposta, iremos implementar um firewall baseado em software, onde o firewall fará parte da própria arquitetura do *Agente Comunicador*, dado que ele recebe todas as comunicações, seja transferência ou apenas uma mensagem, assim fazendo o controle conforme as regras e políticas definidas.

3. Trabalhos relacionados

Em [Chebout et al. 2016], embora não mencione explicitamente *firewalls*, é proposta uma abordagem baseada em aspectos para o controle preventivo em sistemas multiagentes abertos, mediante uma observação dos movimentos dos agentes, interceptando todas as solicitações externa e em seguida é realizada uma análise para verificar se o agente possui ou não determinado recurso para poder prosseguir com sua solicitação, tendo um certo custo de processamento, que leva mais tempo dado que é feito toda uma análise antes de liberar o acesso ou bloqueá-lo. Diferentemente, em nossa abordagem, propomos a criação de regras e políticas para verificação se o agente pode ou não se comunicar, ou realizar transferências.

Em [Gupta et al. 2017], é abordado o paradigma de Internet das Coisas (IoT) e a quantidade de dados em nuvem que são sensíveis a ataques, podendo ser comprometidos. O trabalho propõe uma solução baseada em firewall, embarcado em um Raspberry Pi que protege a comunicação com o banco de dados localizado na nuvem. Através da instalação desse firewall em determinada rede, o trabalho pôde analisar e proteger cada pacote entrando e saindo da mesma. Diferentemente o nosso trabalho, propõe a criação de um modelo de firewall baseado em software para controlar o acesso e a comunicação extra-SMA que podem estar hospedados na mesma rede ou computador.

Em [Vila et al. 2007] são apresentados os desafios existentes na busca de soluções dos problemas de segurança em sistemas multiagentes baseados no framework JADE. Dentre das soluções apresentadas destaca-se a autenticação de usuários por criptografia e a autorização do acesso a serviços por determinado grupo de usuários. O gerenciamento apresentado conseguiu garantir que os dados não fossem acessados por usuários sem permissão. Diferentemente, este trabalho propõe a extensão da arquitetura de agente especialista capaz de gerenciar a movimentação e a comunicação entre diferentes SMA baseados em JASON.

4. Proposta

Buscando adicionar uma camada de segurança e possibilitar o controle de comunicação e acesso ao SMA, propomos a utilização de políticas e regras. Para isso, será necessário estender a arquitetura do *Agente Comunicador*, adicionando ações internas capazes de analisar os cabeçalhos da comunicação KQML [Finin et al. 1994] e da migração Bio-Inspirada [Souza de Jesus. et al. 2021]. No contexto deste trabalho definimos regras e políticas como:

Regra:KQML *Critérios tais como origem, tipo de interação, força ilocucionária ou protocolo que definem se uma comunicação e/ou migração deve ser permitida ou negada.*

Política: *Define os critérios para permitir ou negar uma comunicação e/ou uma migração de agentes, caso não exista uma regra que se aplique à comunicação*

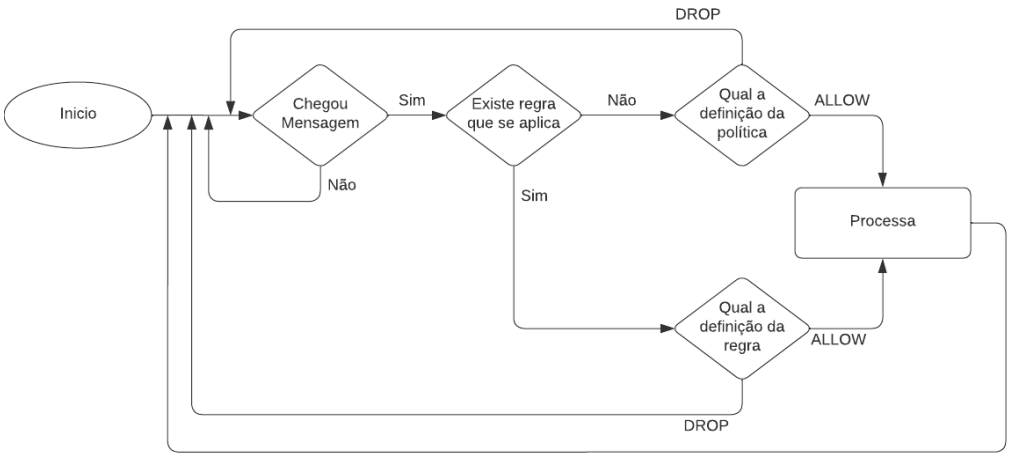


Figura 1. Fluxograma de análise de comunicação KQML de entrada ou saída do SMA.

Esta abordagem visa garantir que apenas agentes confiáveis ingressem na sociedade de agentes. Para isso, propomos um mecanismo para restringir ou permitir a comunicação entre agentes com base em políticas de segurança estabelecidas. Seu objetivo é garantir que sejam processadas pelo agente Comunicador apenas comunicações KQML autorizadas, o acesso não autorizado é impedido. Na Figura1 é apresentado o fluxograma de análise da comunicação KQML.

4.1. Ações Internas propostas

Em resumo, a abordagem proposta neste trabalho inclui estender a arquitetura dos *Agentes Comunicadores*, adicionando ações para gerenciar políticas e regras de encaminhamento de mensagens e agentes móveis, contribuindo com o aumento da segurança de SMA Abertos. Esse agente irá analisar o cabeçalho que chegará na tentativa de comunicação, definindo qual será a ação a ser realizada, sem considerar a mente do agente. Abaixo é apresentada uma ilustração, figura 2, sobre o modelo de plano de ação de um agente Comunicador, definindo a política e as regras para comunicação KQML externa ao SMA.

```
+!firewall <-
    .policy(TIPO,ABRANGÊNCIA,FORÇA|PROTOCOLO,DETERMINAÇÃO);
    .rule(TIPO,ABRANGÊNCIA,ORIGEM,DESTINO,FORÇA|PROTOCOLO,DETERMINAÇÃO).
```

Figura 2. Uma proposta de plano para gerenciamento do *firewall*

Nas políticas, temos que definir os parâmetros Tipo, Abrangência, Força ou Protocolo e a determinação, a tabela abaixo ilustra a definição de cada parâmetro e seus possíveis valores.

Nas regras, será necessário definir os parâmetros tipo, abrangência, origem, destino, força ou Protocolo e a determinação, a tabela abaixo ilustra a definição de cada parâmetro e seus possíveis valores.

- **Tipo:** define se a regra ou política se aplica à entrada ou saída $(input|output)$;
- **Abrangência:** define se a regra ou política se aplica a comunicação, transferência ou ambos $(all|communication|migration)$;
- **Origem:** define a qual SMA ou agente origem a regra se aplica $(source)$;
- **Destino:** define a qual SMA ou agente de destino a regra se aplica $(destination)$;
- **Força/protocolo:** define qual é a força da mensagem ou o protocolo de transferência $(all|illocutionary_{force}|BioInsp_{protocol})$;
- **Determinação:** determina se a regra ou política irá liberar, ou bloquear todos os acessos $(accept|drop)$.

4.2. Cenário comparativo

A proposta do nosso trabalho pode ser comparada o funcionamento de um condomínio, onde esse ocupa o lugar do SMA e o agente comunicador atua como telefonista e porteiro desse sistema. O telefonista do condomínio ao atender uma ligação, ele identifica a origem e para qual morador, a ligação é destinada. Em seguida, ele verifica se existe alguma anotação que seja aplicável a ligação. Por exemplo, algum morador pode estar aguardando essa ligação ou informar que não quer receber a ligação. Caso não exista anotação

específica, o telefonista irá seguir as recomendações gerais do condomínio sobre encaminhamento de ligações. Por exemplo, podem existir regras gerais sobre encaminhamento de telemarketing ou horários de não perturbe.

No caso de migração entre sistemas, dentro do cenário comparativo proposto, uma pessoa, ao chegar até a portaria do condomínio, primeiro o porteiro irá verificar através da identificação dessa pessoa se ela é moradora, caso seja, ela tem sua entrada liberada. Se for um visitante, o porteiro irá verificar se existe alguma anotação sobre entrada pré-autorizada. Caso contrário, o porteiro irá seguir as regras do condomínio, negando a entrada, por exemplo.

5. Resultados Esperados

Este trabalho propõe uma abordagem e espera-se que a políticas de migração e controle de acesso para sociedades de agentes, contribua para fortalecer a segurança de SMA Abertos. Alguns resultados esperados incluem:

- Prevenção de acessos não autorizados: A implementação do modelo de firewall permitirá o controle da migração de agentes para uma sociedade de agentes, evitando a entrada de agentes maliciosos ou não confiáveis. Isso ajudará a proteger o SMA contra invasões e ameaças à segurança.
- Mitigação de riscos de segurança: A definição de políticas de migração permitirá que a sociedade de agentes estabeleça critérios para permitir ou negar a migração de agentes. Essas políticas podem incluir a verificação de credenciais do agente, tipo de comunicação ou transferência de dados e a força da mensagem. Com isso, espera-se reduzir os riscos de agentes maliciosos comprometerem a integridade e confiabilidade das interações no SMA.
- Fortalecimento da confiabilidade de SMA Aberto: Com a implementação das políticas de migração e controle de comunicação, espera-se fortalecer a confiabilidade dos sistemas multiagentes como um todo. Agentes maliciosos terão maior dificuldade em penetrar no sistema, garantindo a integridade das interações entre agentes confiáveis e evitando a interferência de atores não autorizados.
- Melhoria da segurança global: A abordagem proposta visa aprimorar a segurança global dos sistemas multiagentes abertos, considerando a livre circulação de agentes. Ao adotar políticas de migração e controle de comunicação, o SMA estará mais preparado para lidar com riscos de segurança, garantindo a confidencialidade, integridade e disponibilidade das interações entre os agentes.

Em resumo, a implementação da abordagem baseada em políticas de migração e controle de comunicação espera fortalecer a segurança de sistemas multiagentes abertos, reduzir riscos de acessos não autorizados, mitigar ameaças de agentes maliciosos, controlar a comunicação entre SMAs e melhorar a confiabilidade global do sistema. Esses resultados contribuirão para a criação de ambientes mais seguros e confiáveis para o desenvolvimento de sociedades de agentes.

Além disso, será necessário para a validação do modelo, uma análise do custo computacional adicionado no processo de comunicação. Será necessário comparar a taxa de transferência e migração entre agente comunicador padrão, agente comunicador estendido, sem regras e com política padrão *accept* e agente comunicador com regras de controle de acesso.

Referências

Alvares, L. O. and Sichman, J. S. (1997). Introdução aos sistemas multiagentes. In *Jornada de Atualização em Informática*. UnB.

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, Inc., Hoboken, NJ, USA.

Chebout, M. S., Mokhati, F., Badri, M., and Chaouki Babahenini, M. (2016). Towards preventive control for open mas. ICINCO 2016, page 269–274, Setubal, PRT. SCITEPRESS - Science and Technology Publications, Lda. https://doi.org/10.5220/0006005602690274.

Endler, M., Baptista, G., Silva, L. D., Vasconcelos, R., Malcher, M., Pantoja, V., Pinheiro, V., and Viterbo, J. (2011). Contextnet: Context reasoning and sharing middleware for large-scale pervasive collaboration and social networking. In *Proceedings of the Workshop on Posters and Demos Track*, PDT '11, New York, NY, USA. ACM. https://doi.org/10.1145/2088960.2088962.

Finin, T., Fritzson, R., McKay, D., and McEntire, R. (1994). KQML as an agent communication language. In *Proceedings of the Third International Conference on Information and Knowledge Management*, CIKM '94, page 456–463, New York, NY, USA. ACM. https://doi.org/10.1145/191246.191322.

Ford, J. L. (2002). Manual completo de firewalls pessoais: tudo o que você precisa saber para proteger o seu computador. Pearson.

Gupta, N., Naik, V., and Sengupta, S. (2017). A firewall for internet of things. In *2017 9th International Conference on Communication Systems and Networks (COMSNETS)*, pages 411–412. https://doi.org/10.1109/COMSNETS.2017.7945418.

Hubner, J. F. (1995). *Migração de agentes em sistemas multi-agentes abertos*. Dissertação (Mestrado, Universidade Federal do Rio Grande do Sul. Instituto de Informática. Curso de Pós-Graduação em Ciência da Computação., Porto Alegre. https://lume.ufrgs.br/handle/10183/25032.

Jesus, V., Manoel, F., Pantoja, C. E., and Viterbo, J. (2018). Transporte de agentes cognitivos entre sma distintos inspirado nos princıpios de relaçoes ecológicas. In *Workshop-Escola de Sistemas de Agentes, seus Ambientes e apliCações—XII WESAAC*, pages 179–187.

Souza de Jesus., V., Pantoja., C. E., Manoel., F., Alves., G. V., Viterbo., J., and Bezerra., E. (2021). Bio-inspired protocols for embodied multi-agent systems. In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 1: ICAART*, pages 312–320. INSTICC, SciTePress. https://doi.org/10.5220/0010257803120320.

Vila, X., Schuster, A., and Riera, A. (2007). Security for a multi-agent system based on jade. *Computers Security*, 26(5):391–400. https://doi.org/10.1016/j.cose.2006.12.003.

Uma análise da segurança nas comunicações entre agentes inteligentes

Vitor Sobrinho da Fonseca, Nilson Mori Lazarin

[1]Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (Cefet/RJ)
Nova Friburgo, RJ – Brazil

vitor.sobrinho@aluno.cefet-rj.br, nilson.lazarin@cefet-rj.br

Abstract. *Multi-Agent Systems (MAS) have an approach for modeling solutions to complex and distributed problems based on communications between autonomous and cognitive agents. However, some communication protocols between different MAS are based on IoT gateways that do not implement security mechanisms. This article presents the exploitation of a vulnerability in this type of communication, making it possible to capture messages between systems and beliefs, plans and intentions of agents in migration.*

Resumo. *O uso de Sistemas Multiagentes (SMA) possuem uma abordagem para modelagem de soluções para problemas complexos e distribuídos, baseados nas comunicações entre agentes autônomos e cognitivos. Entretanto, alguns protocolos de comunicação entre SMA distintos, estão baseados no uso de gateways IoT que não implementam mecanismos de segurança. Neste artigo é apresentada a exploração de uma vulnerabilidade neste tipo de comunicação, possibilitando a captura de mensagens entre sistemas e de crenças, planos e intenções de agentes em migração.*

1. Introdução

Agentes são softwares autônomos capazes de interagir e tomar decisões para alcançar objetivos individuais e/ou coletivos que compõem um Sistema Multiagentes (SMA) [Wooldridge 2009]. Esses sistemas podem ser abertos ou fechados. Um SMA fechado só permite a interação entre agentes do mesmo sistema, por outro lado, um SMA aberto permite a interação e migração entre agentes móveis de diferentes SMA [Hubner 1995]. O ContextNet é um *middleware* para Internet das Coisas que atua como uma camada intermediária para fornecer suporte a atividades de contexto, permitindo a captura, processamento e disseminação de informações relevantes sobre o ambiente, além de possibilitar uma ampla quantidade de conexões [Endler et al. 2011]. Dessa forma, a aplicação do ContextNet pode prover uma interconexão entre SMA através da internet, permitindo a troca de mensagens e transferência de agentes, possibilitando a criação de um ecossistema digital inteligente [Jesus et al. 2018].

A comunicação é uma parte fundamental no funcionamento de um SMA, pois permite que os agentes compartilhem crenças e planos, coordenem suas ações e realizem movimentações de forma colaborativa, além disso, podem trocar percepções sobre o ambiente em que estão inseridos [Jesus et al. 2021]. Além disso, a arquitetura do ContextNet viabiliza o crescimento da rede garantindo a escalabilidade da comunicação e baixa

latência, pois utiliza do protocolo MR-UDP para fornecer uma comunicação confiável baseada no UDP (*User Datagram Protocol*), com baixa sobrecarga e com recursos focados em mobilidade [David et al. 2012]. Esta é uma opção muito atraente para dispositivos IoT inteligentes, pois possuem recursos limitados de processamento e memória. No entanto, a confiabilidade da implementação desse mecanismo está ligada a integridade da comunicação e não à segurança. Ou seja, mesmo com a implementação desses mecanismos para melhorar o protocolo UDP as comunicações não possuem confidencialidade e autenticidade.

A falta de uma comunicação segura em um SMA gera preocupações e pode levar a consequências negativas, como vazamento de informações sensíveis, sabotagem, manipulação de dados, entre outras. Por exemplo, em um SMA utilizado para gerenciamento de tráfego, os SMA envolvidos podem estar embarcados em veículos autônomos e semáforos inteligentes. Dessa forma, um agente malicioso pode interceptar as comunicações entre os veículos autônomos, enviar informações falsas aos semáforos, criando condições de tráfego caóticas.

Para demonstrar a vulnerabilidade foram utilizadas duas maquinas virtuais (VM) executando a ChonIDE [Souza de Jesus et al. 2023], uma plataforma para programação de SMA com agentes inteligentes que utilizam o modelo BDI (*Belief-Desire-Intention*) [Bratman 1987] que dá suporte à comunicação entre diferentes SMA e migração de agentes baseada em protocolos bio-inspirados [Jesus et al. 2021]. Cada VM executará um SMA, e cada um deles terá um agente Comunicador para realizar a interação entre os sistemas. Essas VMs estarão funcionando em uma rede controlada, onde a comunicação será interceptada com o WireShark[1] a fim de capturar as mensagem e a migração de agentes com suas crenças, planos e intenções trafegados pela rede. Além disso, através do prompt de comando são enviadas falsas mensagens para o SMA destinatário. Dessa forma serão validados os ataques de confidencialidade e autenticidade entre SMA.

O objetivo deste trabalho é explorar uma vulnerabilidade de segurança na comunicação entre SMA que utilizam o ContextNet [Endler et al. 2011], demonstrando a captura de informações trocadas por eles com um ataque de homem-do-meio e classificar quais propriedades de segurança esse ataque inflige. Este trabalho está organizado da seguinte forma, na Seção 2 é apresentada uma fundamentação teórica. na Seção 3 será demonstrado dois cenários de ataques a SMA. Por fim na Seção 4 é apresentada uma discussão e próximos passos nesta pesquisa.

2. Fundamentação Teórica

Nesta seção serão apresentados os conceitos sobre o formato da mensagem entre agentes comunicadores e o ataque de homem no meio, necessários para o entendimento deste trabalho.

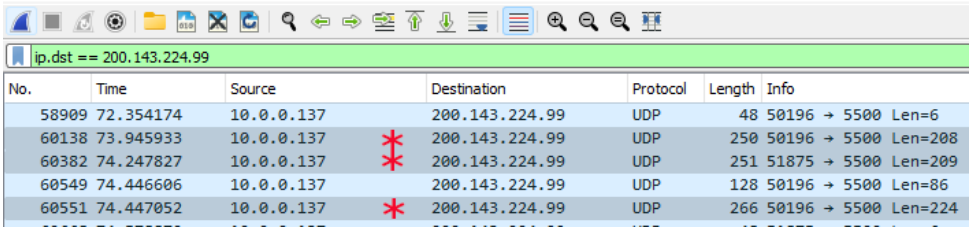
2.1. Formato da mensagem entre agentes Comunicadores

Agentes Comunicadores [Jesus et al. 2021], são uma extensão dos agentes Jason [Bordini and Hübner 2006] responsáveis por facilitar a comunicação entre diferentes SMAs por meio de uma rede IoT utilizando o ContextNet [Endler et al. 2011]. Cada

[1]https://www.wireshark.org/

agente Comunicador é identificado por um *Universally Unique IDentifier* (UUID) e emprega ações internas para enviar mensagens utilizando *Knowledge Query and Manipulation Language* (KQML) [Finin et al. 1994] para outros agentes comunicadores em diferentes SMAs. Além disso, esses agentes são capazes de migrar outros agentes ou a si para diferentes sistemas.

A Figura 1a apresenta a captura dos pacotes de rede utilizando o WireShark. O formato da mensagem enviada na rede, apresentado na Figura 1b é composto pelos seguintes campos: um preâmbulo fixo (*0xFFFE*); um campo para o tamanho em bytes do UUID de destino, seguido pelo endereço do UUID do destino; um campo para o tamanho em bytes da força ilocucionária, seguido pela força da mensagem; um campo para o tamanho em bytes da mensagem, seguido pela mensagem em si. O preâmbulo e os campos de tamanho são todos representados em hexadecimal.



(a) Captura de pacotes de rede utilizando o WireShark.

```
fffe24cdc19c19-5616-4fc5-8d54-372631dd8eff07achieve0cdamageReport"

fffe24788b2b22-baa6-4c61-b1bb-01cff1f5f87804tell0report("Deck 2")"

fffe24cdc19c19-5616-4fc5-8d54-372631dd8eff07achieve1cretransmit(scott,redAlertOn)"
```

(b) Conteúdo dos pacotes de comunicação entre o SMA e o ContextNet.

Figura 1. Captura dos pacotes transmitidos.

2.2. Ataque do homem no meio

O Ataque de Homem no meio é uma forma de ataque cibernético em que um terceiro malintencionado intercepta a comunicação entre dois pontos legítimos. Durante um ataque de homem no meio, o invasor se posiciona entre o remetente e o destinatário da comunicação, interceptando e possivelmente alterando as mensagens trocadas entre eles. O objetivo principal de um ataque de homem no meio é obter acesso não autorizado a informações confidenciais, como senhas, informações bancárias ou dados pessoais [Stallings 2008].

3. Ataque à comunicação entre agentes

Este trabalho demonstra a violação das propriedades de confidencialidade, integridade e autenticidade, através dos ataques de divulgação e de identidade falsa [Bijani and Robertson 2014] nas comunicações entre SMA. No ataque de divulgação conseguimos fazer a interceptação de mensagens dos agentes e capturar suas crenças, planos e intenções, já no ataque de identidade falsa, enviamos mensagens falsas ao SMA de destino para atrapalhar a execução de um plano. Nas subseções a seguir será demonstrado como foram realizados os ataques tanto na comunicação quanto na migração.

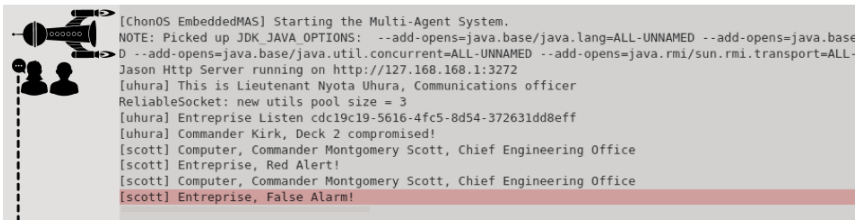
Os ataques foram realizados da seguinte forma: primeiramente executamos os SMAs nas VMs e deixamos os agentes Comunicadores interagirem normalmente na rede,

porém monitorando os envios dos pacotes utilizando a ferramenta WireShark. Logo após é feito uma análise nos pacotes interceptados onde é possível identificar o UUID e a mensagem enviados pelos agentes, dessa forma obtivemos dados suficientes para injetar mensagens falsas na rede e atrapalhar a comunicação. Abaixo são descritos dois cenários de ataque, um de comunicação e outro de migração entre agentes de SMA distintos.

3.1. Cenário 1: Ataque na Comunicação

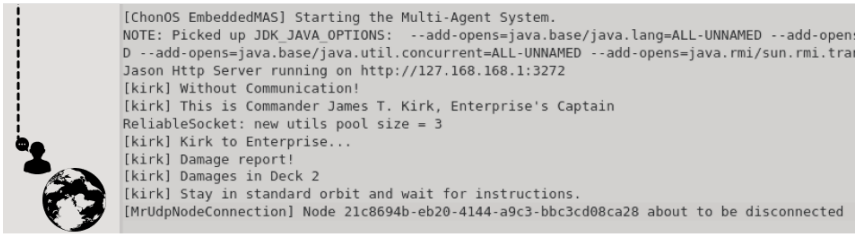
No primeiro cenário, um SMA (*ncc1701.mas2j*) com os agentes Scott e Uhura (Comunicador) representados na Figura 2a. O segundo SMA (*bajor.mas2j*) com o agente Kirk (Comunicador) representado na Figura 2c. O agente Kirk inicia a comunicação enviando mensagens para o agente Uhura que aguarda esse primeiro contato para identificar que foi estabelecido uma conexão. Posteriormente o agente Kirk solicita ao agente Uhura o relatório de danos, que responte que o Deck 2 está comprometido, Kirk pede para aguardar novas atualizações e envia uma crença (*RedAlert*) para ser encaminhada ao agente Scott, que imediatamente executa um plano.

Entretanto, o atacante capturou as comunicações e descobriu o UUID dos comunicadores Uhura e Kirk, uma vez que a comunicação não é criptografada. Através do terminal de comando, é enviada uma mensagem ao agente Uhura, identificando-se como agente Kirk, solicitando que uma crença *RedAlertOff* seja encaminhada ao agente Scott, cancelando a ação anterior do agente como vemos na Figura 2b.



(a) Log do SMA *ncc1701.mas2j*.

(b) Ataque de *Injeção de Mensagem Falsa*.



(c) Log do SMA *bajor.mas2j*.

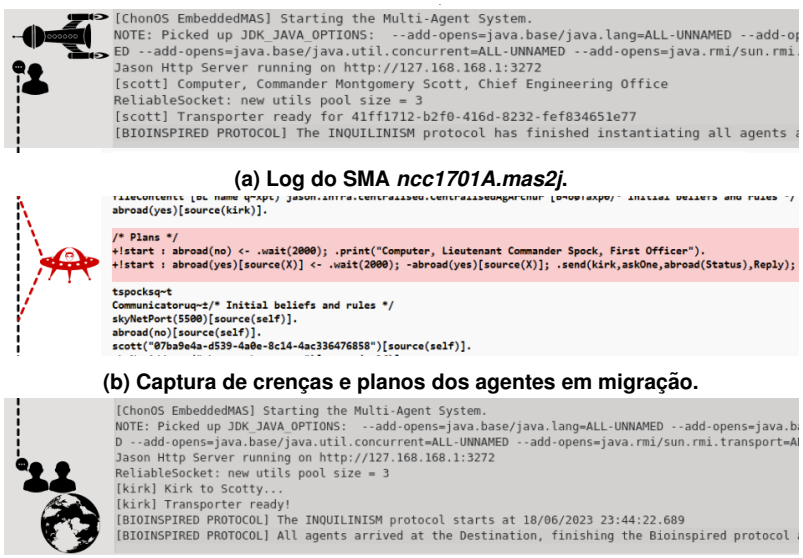
Figura 2. Ataque durante comunicação entre SMAs.

3.2. Cenário 2: Ataque na Migração

No segundo cenário, temos um SMA (ncc1701A.mas2j) com o agente Scott (Comunicador) representado na Figura 3a. O segundo SMA (andoria.mas2j) com os agentes

Kirk (Comunicador) e Spock representados na Figura 3c. O agente Scott está ativo na rede apenas aguardando mensagens. O agente Kirk inicia a comunicação com o agente Scott informando que irá iniciar uma transferência de agentes. O agente Scott confirma o recebimento da mensagem. Assim o agente Kirk ativa o protocolo Inquilinismo [Jesus et al. 2021] e chega ao SMA destino com sucesso.

Porém, toda a comunicação foi interceptada e agora não só foi possível obter o UUID e as mensagens transmitidas na rede, pois como os agentes Kirk e Spock migraram, todas a suas crenças, planos e intenções foram capturadas como vemos na Figura 3b. Isso possibilita a criação de agentes clones que podem ser igualmente transmitidos para o SMA de destino.



(a) Log do SMA *ncc1701A.mas2j*.

(b) Captura de crenças e planos dos agentes em migração.

(c) Log do SMA *andoria.mas2j*.

Figura 3. Ataque durante comunicação entre SMAs.

4. Trabalhos futuros

Nesta etapa da pesquisa foi possível provar as vulnerabilidades na comunicação entre SMA que usam o middleware ContextNet. As próximas etapas consistem na formulação de uma abordagem para troca de chaves criptográficas, diretamente na dimensão dos agentes do SMA. Além disso, será necessária uma validação da conformidade dos padrões de segurança a serem adotados, tal como apresentado em [Rocha et al. 2020]. Por fim, uma será necessário criar mecanismos de comunicação segura, sobre meios inseguros de comunicação, tal como apresentado em [Freitas et al. 2021], especificamente para comunicação entre SMA que utilizam o ContextNet.

Referências

Bijani, S. and Robertson, D. (2014). A review of attacks and security approaches in open multi-agent systems. *Artif Intell Rev*, 42:607–636. DOI: 10.1007/s10462-012-9343-1.

Bordini, R. H. and Hübner, J. F. (2006). BDI Agent Programming in AgentSpeak Using Jason. In Toni, F. and Torroni, P., editors, *Computational Logic in Multi-Agent Systems*, pages 143–164, Berlin, Heidelberg. Springer Berlin Heidelberg. DOI: 10.1007/11750734_9.

Bratman, M. (1987). Intention, plans, and practical reason.

David, L., Vasconcelos, R., Alves, L., Andre, R., Baptista, G., and Endler, M. (2012). A large-scale communication middleware for fleet tracking and management. In *Simposio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2012), Salão de Ferramentas*, pages 964–971.

Endler, M., Baptista, G., Silva, L. D., Vasconcelos, R., Malcher, M., Pantoja, V., Pinheiro, V., and Viterbo, J. (2011). Contextnet: Context reasoning and sharing middleware for large-scale pervasive collaboration and social networking. In *Proceedings of the Workshop on Posters and Demos Track*, PDT '11, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/2088960.2088962.

Finin, T., Fritzson, R., McKay, D., and McEntire, R. (1994). KQML as an Agent Communication Language. In *Proceedings of the Third International Conference on Information and Knowledge Management*, CIKM '94, page 456–463, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/191246.191322.

Freitas, J., Souza, L., Sardou, P., and Lazarin, N. (2021). Comunicação segura em VANET. In *Anais da XIX Escola Regional de Redes de Computadores*, pages 109–114, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/errc.2021.18551.

Hubner, J. F. (1995). *Migração de agentes em sistemas multi-agentes abertos*. Dissertação (Mestrado), Universidade Federal do Rio Grande do Sul. Instituto de Informática. Curso de Pós-Graduação em Ciência da Computação, Porto Alegre. `https://lume.ufrgs.br/handle/10183/25032`.

Jesus, V., Manoel, F., Pantoja, C. E., and Viterbo, J. (2018). Transporte de agentes cognitivos entre sma distintos inspirado nos princıpios de relaçoes ecológicas. In *Workshop-Escola de Sistemas de Agentes, seus Ambientes e apliCações—XII WESAAC*, pages 179–187.

Jesus, V. S. d., Pantoja, C., Manoel, F., Alves, G., Viterbo, J., and Bezerra, E. (2021). Bio-Inspired Protocols for Embodied Multi-Agent Systems. In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 1: ICAART,*, pages 312–320. SciTePress. DOI: 10.5220/0010257803120320.

Rocha, I., Schott, R., Verly, P., and Lazarin, N. (2020). Análise de desempenho e conformidade em bibliotecas criptográficas para internet das coisas. In *Anais da VI Escola Regional de Sistemas de Informação do Rio de Janeiro*, Porto Alegre, RS, Brasil. SBC. `https://sol.sbc.org.br/index.php/ersi-rj/article/view/10118`.

Souza de Jesus, V., Mori Lazarin, N., Pantoja, C. E., Vaz Alves, G., Ramos Alves de Lima, G., and Viterbo, J. (2023). An IDE to Support the Development of Embedded Multi-Agent Systems. In Mathieu, P., Dignum, F., Novais, P., and De la Prieta, F., editors, *Advances in Practical Applications of Agents, Multi-Agent Systems, and Cognitive Mimetics. The PAAMS Collection*, pages 346–358, Cham. Springer Nature Switzerland. DOI: 10.1007/978-3-031-37616-0_29.

Stallings, W. (2008). *Criptografia e segurança de redes princípios e práticas*. Pearson Prentice Hall, São Paulo, 4. ed edition.

Wooldridge, M. (2009). *An introduction to multiagent systems*. John wiley & sons.

Uma arquitetura baseada em Docker para Sistemas Multiagentes Abertos

Gustavo L. de Lima[1], **Marilton S. de Aguiar**[1]

[1]Programa de Pós Graduação em Computação (PPGC)
Universidade Federal de Pelotas (UFPEL)
Pelotas – RS – Brazil

{gustavolameirao,marilton}@inf.ufpel.edu.br

Abstract. *In Open Multi-Agent Systems (SMAA), heterogeneous agents (different environments/models) migrate from one system to another, taking their attributes and knowledge with them. The complexity of the opening comes from the dynamic behavior that the change of agents entails, being necessary to formulate techniques to analyze this complexity and understand the general behavior of the system. The article presents an architecture based on Docker to assist in the development of SMAA, acting in the migration of agents between different models running in heterogeneous hardware/software scenarios. We used a simulation scenario with the Open Sugarscape 2 Constant Growback model (NetLogo) and Gold Miners (JaCaMo) to verify the feasibility of the proposal.*

Resumo. *Em Sistemas Multiagentes Abertos (SMAA), agentes heterogêneos (diferentes ambientes/modelos) migram de um sistema para outro, levando seus atributos e conhecimentos. A complexidade da abertura vem do comportamento dinâmico que a mudança de agentes acarreta, sendo necessário formular técnicas para analisar essa complexidade e entender o comportamento geral do sistema. O artigo apresenta uma arquitetura baseada em Docker para auxiliar no desenvolvimento de SMAA, atuando na migração de agentes entre diferentes modelos rodando em cenários heterogêneos de hardware/software. Utilizamos um cenário de simulação com o modelo Open Sugarscape 2 Constant Growback (NetLogo) e o Gold Miners (JaCaMo) para verificar a viabilidade da proposta.*

1. Introdução

Os Sistemas Multiagentes Abertos (SMAA) são um tipo específico de Sistemas Multiagentes (SMA) que permite a interação entre agentes participantes de diferentes modelos. Em SMAA, analisamos agentes heterogêneos sob a perspectiva que estes podem migrar de um sistema para outro, levando seus atributos e conhecimentos [Jamroga et al. 2013]. A heterogeneidade dos agentes pode vir de diferenças entre os modelos, como arquitetura, objetivos ou políticas [Uez 2018].

No entanto, diferentes problemas surgem quando se desenvolve aplicações em SMAA quando comparado a SMA [Dalpiaz et al. 2010]. Primeiro, pode haver diferenças de implementação onde agentes e modelos podem ser criados por equipes diferentes, em outras linguagens de programação ou mesmo em várias plataformas/arquiteturas de agentes. Além disso, frequentemente conflitos de objetivos podem não garantir que os agentes ajam de forma cooperativa e coordenada [Huynh et al. 2004, Kaffille and Wirtz 2006].

Por fim, há ainda a dificuldade gerada pelas incertezas e pelo comportamento dinâmico que a mudança de agentes acarreta.

SMAA precisam lidar com problemas que não estão presentes em sistemas multiagentes fechados (SMA comum). Por exemplo, a migração de agentes entre modelos pode ocorrer em tempo de execução, e a motivação para essa migração de um agente de um sistema para outro pode ser diferente, geralmente de escolha do desenvolvedor, como falhas de execução, vontade própria ou algum gatilho. Além disso, podem ocorrer conflitos de interesse entre os novos agentes quando não projetados para trabalhar naquele conjunto [Uez 2018]. Embora o assunto seja conhecido há muito tempo [van Eijk et al. 1999], ainda há pesquisas na área, como em [Hendrickx and Martin 2016, Franceschelli and Frasca 2018, Houhamdi and Athamena 2020, Noh and Park 2020, Jiang et al. 2021, Franceschelli and Frasca 2021].

Quanto maior o grau de abertura de um SMAA, menor o número de mudanças em um modelo para receber ou enviar agentes [Jamroga et al. 2013]. Desta forma, um sistema multiagentes aberto perfeito não precisaria de transformações (0 passos) para acomodar novos componentes, enquanto, por outro lado, existem sistemas que exigiriam um redesenho completo (muitos passos) quando novos agentes chegassem [Jamroga et al. 2013].

Nesse contexto, o Docker pode auxiliar na resolução de parte dos problemas citados. Docker é uma ferramenta de código aberto que atua no desenvolvimento de aplicações na forma de contêineres. Os contêineres são uma abstração que une o código e suas dependências. Vários contêineres podem ser executados na mesma máquina, compartilhando recursos do kernel do sistema operacional, sendo cada um executado como processos isolados do espaço do usuário [Anderson 2015]. Além disso, o uso do Docker permite uma maior modularização do sistema, onde a implementação de cada módulo dentro de um contêiner possibilita a execução de diferentes ferramentas, em diferentes versões, em diferentes sistemas operacionais, permitindo assim uma fácil substituição de que o contêiner executa [Turnbull 2014].

Além disso, no sentido de diminuir a complexidade de transação dos agentes entre modelos, foram implementados módulos específicos de registro (**register**) e roteamento (**router**) de agentes. Estes módulos permitem com que a lógica de criação de agentes e os critérios de escolha de para onde os agentes vão quando saem dos modelos seja abstraída dos modelos. O isolamento destas lógicas permite com que diferentes decisões possam ser tomadas, como por exemplo estender uma estrutura simples de roteamento de agentes por algo mais complexo, como um modelo que retreina os agentes antes de voltarem aos modelos de simulação. Por fim, a modularização permite agregar ao sistema novos comportamentos que os modelos não previram em sua fase de concepção. Em geral, a abordagem visa avançar na generalização do SMAA, simplificando o processo de abertura.

Este artigo apresenta um modelo baseado em Docker para auxiliar no desenvolvimento de SMAA e facilitar a migração de agentes entre diferentes modelos que podem ser executados em cenários heterogêneos de hardware e software, diferentes ambientes de desenvolvimento e ferramentas para SMA ou outros sistemas operacionais.

Organizamos este trabalho da seguinte forma. Primeiramente, apresentamos trabalhos relacionados na Seção 2. Em seguida, a Seção 3 apresenta detalhes da abordagem proposta, como o fluxo de execução e detalhes de implementação. A seguir, a Seção 4 apresenta os resultados obtidos, enfatizando o cenário de simulação desenvolvido, que é de grande importância para verificar a viabilidade da implementação. Por fim, a Seção 5 apresenta as considerações finais do estudo.

2. Trabalhos Relacionados

A Tabela 1 sintetiza os estudos similares encontrados, indicando se é focado no aspecto de organização do SMAA, se utiliza algumas ferramentas DevOps, e se lida com mais de uma plataforma SMA. O aspecto do DevOps é importante porque é uma maneira de implementar várias partes dos sistemas que podem executar diferentes linguagens e sistemas operacionais, possibilitando o uso de todos os tipos de ferramentas SMA. O último aspecto é quantas plataformas/ferramentas SMA estão sendo usadas. Este aspecto é essencial para tornar a ferramenta o mais abrangente possível.

Dos 15 estudos analisados, o foco principal de 9 deles é lidar com os problemas organizacionais advindos da abertura do SMAA. Dos outros 6, 3 são dependentes da linguagem, não permitindo a utilização de ferramentas SMA/SMAA que rodam em um ambiente de desenvolvimento/utilização diferente, como o uso de outra ferramenta que utiliza outra linguagem. Por fim, além do nosso, apenas um estudo trata de mais de uma plataforma SMA. Este estudo trata apenas de SMA, não relacionado a SMAA, portanto não lida com o problema de transporte de agentes entre modelos. Além disso, este estudo conecta duas plataformas SMA que rodam na mesma linguagem (Java), não considerando outras ferramentas que possam utilizar outras linguagens.

Tabela 1. Características dos Trabalhos Relacionados.

Estudo	Ano	Relacionado à organização	Ferramenta de DevOps	Plataforma de SMA
[Jamroga et al. 2013]	2013	Sim	–	Única
[Dalpiaz et al. 2010]	2010	Sim	–	Única
[Demazeau and Costa 1996]	1996	Sim	–	Única
[Gonzalez-Palacios and Luck 2006]	2006	Sim	–	Única
[Artikis 2011]	2012	Sim	–	Única
[Paurobally et al. 2003]	2003	Sim	–	Única
[Singh and Chopra 2009]	2009	Sim	–	Única
[Hattab and Lejouad Chaari 2021]	2021	Sim	–	Única
[Houhamdi and Athamena 2020]	2020	Sim	–	Única
[Ramirez and Fasli 2017]	2017	Outro	–	Múltiplas
[Uez 2018]	2018	Outro	–	Única
[Dähling et al. 2021]	2021	Outro	Docker e Kubernetes	Única
[Pfeifer et al. 2021]	2021	Outro	Docker	Única
[Perles et al. 2018]	2018	Outro	–	Única
Nossa abordagem	2023	Outro	Docker	Múltiplas

Em conclusão, nossa abordagem contribui para o estado da arte em como o programador usará as plataformas. Por exemplo, alguns estudos propõem novas abordagens que exigem que o programador reconstrua seus modelos de acordo com as propostas. Em nossa arquitetura, os programadores não teriam que mudar todo o seu modelo. Em vez disso, eles adicionarão as estruturas necessárias aos seus modelos para se comunicarem com nossa arquitetura (providas pela própria arquitetura), permitindo uma transição mais fácil de um SMA fechado para um SMA aberto. Outro trabalho relacionado já utilizou

abordagens baseadas em contêiner/nuvem relacionadas ao sistema SMA, mostrando que essa abordagem é promissora. Além disso, alguns estudos testaram a comunicação entre agentes NetLogo e Jason.

3. Abordagem Proposta

A abordagem proposta utiliza como base contêineres no Docker. Cada imagem de contêiner é gerada a partir de um arquivo de descrição, denominado *DockerFile*, contendo informações sobre as estruturas necessárias no contêiner, como sistema operacional, linguagens de programação e código a ser executado. Além disso, o Docker possui um banco de imagens, o *Docker Hub Container Image Library*, que contém as imagens frequentemente utilizadas pelos desenvolvedores, ampliando a aplicabilidade da abordagem.

Os contêineres são responsáveis pela execução de cada bloco essencial da estrutura da arquitetura. Portanto, contêineres podem ser facilmente substituídos ou adicionados, tornando a arquitetura mais modularizada e adaptável a novos cenários não previstos na concepção, tornando-a mais robusta. Também lidamos com questões de segurança, criando redes virtuais separadas para que cada contêiner tenha uma visão parcial do sistema, limitando o acesso a códigos confidenciais.

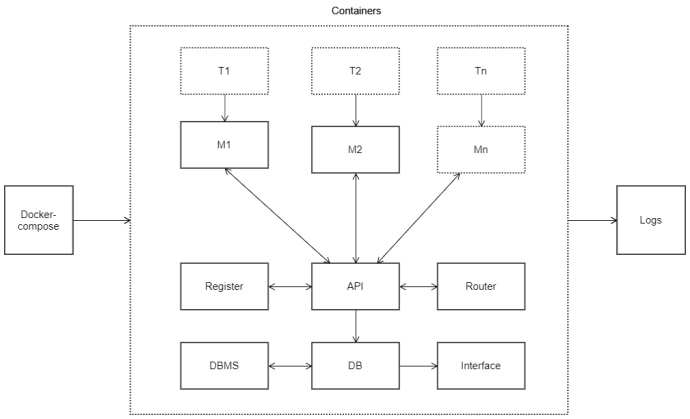


Figura 1. Organização da arquitetura proposta.

A Figura 1 apresenta uma descrição geral da arquitetura proposta em formato de diagrama de blocos. No bloco **Docker-compose**, temos a geração de imagens e serviços baseados no arquivo *docker-compose.yaml* e no *Dockerfile* de cada serviço. Cada *Dockerfile* contém a sequência de instruções necessárias para montar a imagem com todos os requisitos que cada serviço necessita. Então, com base nas imagens, o **Docker-compose** usa os parâmetros de cada serviço (nome do contêiner, portas expostas, rede, volumes, comando/arquivo a ser executado) e constrói os contêineres referentes. Por fim, após todos os serviços descritos no **Docker-compose** serem montados, eles podem ser executados usando apenas um comando.

Os blocos $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_n$ representam os contêineres responsáveis por executar os modelos. No arquivo de configuração principal, o designer do sistema pode escolher um parâmetro chamado *auto-run*. Quando definimos este parâmetro como *True*, os modelos são executados automaticamente quando o contêiner é montado. Se for *False*, devem ser implementados contêineres que irão determinar quando a execução dos modelos começa (contêineres de disparo $\mathrm{T}_1, \mathrm{T}_2, \ldots, \mathrm{T}_n$). Além disso, é necessário implementar as

comunicações entre a arquitetura e os modelos, como os gatilhos *triggers* de entrada/saída e as funções de envio/recebimento de agentes. Até este estágio de desenvolvimento, existem contêineres, gatilhos e funções prontos para oferecer suporte aos modelos NetLogo (contêiner Java) e JaCaMo (Java via Gradle).

$\mathbf{T}_1$, $\mathbf{T}_2$, ..., $\mathbf{T}_n$ são contêineres de gatilhos de execução, responsáveis por enviar uma mensagem aos modelos para iniciar sua execução. Usamos esses contêineres quando existe uma lógica mais robusta para iniciar a execução dos modelos. Por exemplo, eles podem executar códigos que leem um sensor, aguardar a medição ou executar um determinado modelo somente quando os agentes são atribuídos.

O bloco **API** (*Application Programming Interface*) é o contêiner responsável por gerenciar o acesso dos modelos ao banco de dados. Esse contêiner atua como um intermediário quando qualquer parte do sistema deve gravar, ler, atualizar ou excluir informações do banco de dados. Atualmente, a API é implementada em Python, usando o *framework* Flask [Grinberg 2018].

O bloco **Database** (DB) representa o contêiner responsável pelo banco de dados onde armazenamos, para cada modelo, todas as informações de entrada/saída de agentes para serem acessadas por outros contêineres que possam necessitar dessas informações. Além disso, esse contêiner é responsável por realizar as operações do banco de dados (ler, gravar, atualizar e excluir).

O bloco **Database Management System** (DBMS) representa o contêiner que facilita o acesso ao DB. Ele fornece uma interface da Web (exposta por padrão à máquina *host*) que pode importar/exportar conteúdo/arquivos SQL (*Structured Query Language*) e visualizar as informações em tempo real.

O bloco **Register** é um contêiner responsável por gerenciar a entrada de todos os agentes na plataforma. Todo agente deve possuir uma identificação para ser utilizado pela arquitetura, portanto as primeiras requisições de inserção de novos agentes vindas de qualquer modelo de contêineres necessitam de uma identificação única. Assim, o **Register** é responsável por gerar uma identificação única para cada agente, e então encaminhá-los de volta aos ambientes.

O bloco **Router** é responsável por receber todos os agentes que saíram de um determinado modelo, analisando e julgando a qual modelo enviar o agente. Esta etapa especifica os protocolos de entrada e saída do agente de diferentes contêineres/modelos. Este bloco é parte essencial da proposta, pois os modelos delegam ao roteador a tarefa de distribuição dos agentes entre os modelos. Quando o ambiente de simulação não compartilha (ou apenas parcialmente) informações sobre o mundo, o roteador pode lidar com diversos problemas relacionados à ausência dessas informações. O julgamento pode ocorrer de forma diferente, como analisar os agentes mais promissores, códigos de aprendizado de máquina e executar um novo modelo SMA que retreine os agentes.

O bloco **Interface** é um contêiner que recebe as informações de movimentação do agente através do sistema e gera relatórios expostos e formatados para visualizar métricas de execução da arquitetura. Na implementação atual, este contêiner possui um Apache Webserver rodando código PHP (*Hypertext Preprocessor*) que lê todas as informações dos agentes que já passaram pelo **Router** e gera um relatório com todos os agentes, seus atributos e o caminho percorrido por eles, em cada passagem. A forma geral do

relatório é por meio de um front-end HTML (*HyperText Markup Language*), CSS (*Cascading Style Sheet*) e código JS (JavaScript)), que é acessível através da exposição da porta do contêiner ao *host* (porta 80 por padrão). Cada tupla do relatório contém todos os atributos essenciais para cada interação do agente até o momento.

Por fim, o bloco **Logs** não é um contêiner e sim um módulo responsável por gerar *logs* das saídas de cada contêiner do sistema, *logs* regulares do docker (via comando docker log), tuplas de banco de dados via exportação SQL e impressões em arquivos txt (usando operadores de terminal ao executar o contêiner no docker-compose). Esses *logs* podem auxiliar na depuração de tarefas ou na geração de dados para análise de execução.

4. Resultados

Para o cenário de simulação proposto, construímos contêineres para dar suporte a duas diferentes ferramentas de desenvolvimento para SMA: NetLogo [Tisue and Wilensky 2004] e JaCaMo [Boissier et al. 2013]. Escolhemos um modelo da documentação de cada ferramenta: NetLogo's Open Sugarscape 2 Constant Growback [Epstein and Axtell 1996] e JaCaMo's Gold Miners [Bordini et al. 2006], com adaptações. Usamos três contêineres modelo, executando duas cópias isoladas do Open Sugarscape e uma do Gold Miners. O principal objetivo deste cenário de simulação é verificar a viabilidade da abordagem, permitindo que os agentes se movam livremente entre os modelos, mesmo que as duas ferramentas utilizem arquiteturas de agentes completamente diferentes.

4.1. NetLogo – Open Sugarscape (Modelo e Adaptações)

O modelo Open Sugarscape simula uma população com recursos limitados onde cada agente é representado como uma formiga que, para sobreviver, deve se deslocar pelo ambiente em busca de alimento (açúcar). Cada formiga nasce com uma quantidade de açúcar (de 5 a 25), metabolismo (de 1 a 4) e visão (de 1 a 6). O metabolismo define quanta energia a formiga perde ao se mover. Se a energia da formiga chega a zero, ela morre e sai do modelo. A visão determina quantas posições de distância a formiga pode ver o açúcar de sua posição atual.

A Figura 2(a) e a Figura 2(b) mostram, respectivamente, o código-fonte necessário para que o NetLogo possa receber e enviar dados da arquitetura. Nossa abordagem fornece as funções *check_new_agent_on_api* e *send_agent_to_api* ao usuário para definir os gatilhos para enviar (e quais informações devem ser enviadas) e receber um agente. Para este modelo específico, quando um agente morre (segundo o modelo original, um agente morre quando seu alimento chega a zero), o modelo envia este agente para a arquitetura e remove este agente da simulação.

Fizemos outro ajuste quando o agente volta para a simulação; em vez de usar suas últimas informações sobre comida, geramos esse parâmetro aleatoriamente, enquanto os outros parâmetros são os mesmos da última simulação. Fizemos esse ajuste por causa da maneira como a simulação original funciona. Se o agente sair da simulação quando sua comida for zero, e usarmos essa mesma informação novamente quando o agente retornar, isso faria o agente chegar à simulação com zero comida e morreria novamente.

Os dois únicos atributos adicionados aos agentes NetLogo são *agent_id* e *historic*. Usamos o *agent_id* guardar o id único do agente para a arquitetura, enquanto o id normal é usado apenas para a simulação do NetLogo. O atributo *historic* mostra o caminho de quais

```
to check_new_agent_on_api
  py:setup py:python
  py:run "from db_python_netlogo import receive_agent"
  let result py:runresult (word "receive_agent('" ("m1") "')")
  ifelse(length result > 0)
  [
    foreach result
    [
      x ->
      let tuple read-from-string item 1 x

      create-turtles 1
      [
        set agent_id item 0 x
        set sugar random-in-range 5 25
        set metabolism item 1 tuple
        set vision item 2 tuple
        set historic item 2 x

        set shape "circle"
        move-to one-of patches with [not any? other turtles-here]
        set vision-points []
        foreach (range 1 (vision + 1)) [ n ->
          set vision-points sentence vision-points (
            list (list 0 n) (list n 0) (list 0 (- n)) (list (- n) 0))
        ]
        run visualization
        print "Agent created"
      ]
    ]
  ]
  [
    print ("There is no new agent on DB")
  ]
end
```

(a)

```
to send_agent_to_api
  py:setup py:python
  py:run "from db_python_netlogo import send_function"
  ;//Example: "send_function('12', '[1 2 3]', '1-1')"

  let updated_historic ""
  ifelse (historic = "")
  [
    set updated_historic 1
  ]
  [
    set updated_historic (word "" (historic) "-1")
  ]
  let tuple []
  set tuple lput sugar tuple
  set tuple lput metabolism tuple
  set tuple lput vision tuple


  let result py:runresult (word
    "send_function('" (agent_id) "', '" tuple "', '" (updated_historic) "')"
  )

  print("Agent sent?")
  print(result)
end
```

(b)

Figura 2. Funções do NetLogo responsáveis por (a) receber e (b) enviar agentes entre a arquitetura e o modelo no NetLogo.

modelos os agentes passaram. Para adaptar qualquer modelo NetLogo à nossa plataforma, a única dependência necessária é a extensão do Py NetLogo. Precisamos desta extensão para enviar/receber informações da API através do código Python. Temos funções simples para isso, bastando o programador incluir essas funções e utilizá-las sempre que o modelo precisar enviar/receber informações sobre os agentes.

4.2. JaCaMo – Gold Miners (Modelo e a Adaptações)

Este modelo simula um conjunto de agentes representando mineradores cujo papel é navegar no ambiente. Por exemplo, quando um agente encontra um nó de ouro, ele interrompe seu objetivo atual, obtém o ouro e o traz de volta ao depósito central. No modelo original, há a representação dos agentes, do depósito, de barreiras do ambiente (posições não acessíveis aos agentes) e os nós de ouro.

Um trecho do código fonte (Java e ASL – *AgentSpeak Language*) necessário para que o JaCaMo possa enviar e receber dados da arquitetura é mostrado na Figura 3. A Figura 3(a) e a Figura 3(b) mostram o código Java para receber e deletar agentes. Figura 3(c) e Figura 3(d) mostram o código ASL para verificar novos agentes e remover agentes excluídos da arquitetura. A arquitetura apresenta dois agentes para envio/recebimento de informações: *killer_agent* e *check_new_agents*. O *killer_agent* espera uma mensagem para que um agente seja removido da simulação. Quando isso acontece, este agente salva todas as informações do agente a ser removido em um arquivo *.asl* (que será utilizado no caso deste agente voltar para a simulação) e então retira o agente da simulação.

A função *check_new_agents* verifica se um agente está esperando para entrar no modelo atual. Se sim, o agente é inserido no modelo. Este agente utiliza código Java e ASL para se comunicar com a API e inserir o agente na simulação. Também podemos adaptar este código para inserir informações iniciais extras ao agente, como crenças, objetivos ou foco nos artefatos do CArtAgO. Quando o modelo inclui um agente na simulação,

Figura 3. Funções Java e ASL inseridos no código JaCaMo para prover as funções responsáveis por (a) receber, (b) excluir, (c) checar, e (d) remover agentes da/para a arquitetura.

o *check_new_agents* verifica se existe um arquivo *.asl* com o id do agente, ou seja, se este agente já esteve no modelo. Se o arquivo existir, o modelo cria o agente com suas informações anteriores (utilizando o arquivo *.asl* anterior). Caso contrário, o modelo cria o agente com um novo arquivo *.asl* usado como modelo.

Na atual adaptação do modelo original, utilizamos apenas um agente minerador, juntamente com os agentes da arquitetura (*check_new_agents* e *killer_agent*) e o agente líder. Assim, quando o agente entra na simulação, ele assume o controle do agente minerador. Este agente é inserido com base nas informações armazenadas anteriormente (se tiver), navega pelo mapa, pega uma unidade de ouro, traz de volta ao depósito e então sai da simulação (envia uma mensagem para o *killer_agent* para ser removido da simulação). Finalmente, quando o agente sai da simulação, o modelo verifica o próximo agente que entrará na simulação e assume o controle do agente minerador para que a simulação possa continuar sua execução.

Para usar o modelo JaCaMo na nossa plataforma, precisamos rodar o projeto via Gradle (que é a opção padrão), adaptando-o para isso [Boissier et al. 2022a]. As etapas necessárias estão apresentadas na documentação do GitHub do JaCaMo [Boissier et al. 2022b]. Depois, precisamos incluir duas bibliotecas no arquivo Gradle de compilação: JSON-simple (para lidar com informações no formato JSON) e cliente HTTP (para lidar com solicitações HTTP). O último passo é incluir os agentes *killer/check* (com seus arquivos ASL) que lidam com a API. Com essas etapas concluídas, o modelo está pronto para se comunicar com a API. Agora, o programador pode usar as funções do sistema para enviar e receber agentes entre a API e o modelo.

4.3. Cenário de Simulação

No cenário de simulação criado, os contêineres $\mathbf{M}_1$ e $\mathbf{M}_2$ executam simultaneamente duas réplicas do modelo Sugar Scape NetLogo, e o contêiner $\mathbf{M}_3$ executa o modelo Gold Mi-

ners, no JaCaMo. Todos os três modelos são executados indefinidamente e usam a opção *auto-run* (modelos executados automaticamente quando os contêineres são montados). Quando um agente morre, o modelo atual é responsável por enviá-lo para o **Router** (e depois para o **DB** via **API**).

No início da execução, ao invés de criar os agentes diretamente pelo modelo, os modelos $\mathbf{M}_1$ e $\mathbf{M}_2$ pedem ao contêiner **Register** para criar os agentes e receberem uma identificação única no SMAA. Na implementação atual do cenário de simulação, apenas os modelos NetLogo solicitam novos agentes. O modelo JaCaMo está recebendo e enviando agentes, mas não os criando na inicialização, embora seja possível.

Após a etapa inicial de criação e cadastro de todos os agentes, os modelos $\mathbf{M}_1$, $\mathbf{M}_2$ e $\mathbf{M}_3$ continuam sua execução, juntamente com o processamento do **Router**. Sempre que um agente sai de algum dos modelos, ele vai para o contêiner **DB** (via API). Uma vez que uma nova informação está no **DB**, o **Router** sabe que existem novos agentes a serem processados, então ele lê os agentes e os envia para um modelo de acordo com o tipo de roteamento (por padrão, aleatoriamente). Ambos os contêineres **Log** ou **Interface** podem acessar o fluxo desta informação.

É fundamental ressaltar que a maioria das adaptações não são substanciais, que impliquem a descaracterização os modelos originais para serem utilizados na arquitetura; ao contrário, as alterações são devidas porque esses modelos precisam estar preparados para receber agentes de fora do modelo. Fornecemos exemplos das funções de gatilho para enviar e receber informações para a arquitetura. Os usuários devem adicioná-lo ao seu modelo, adaptar as informações que desejam enviar ou receber e usar os gatilhos quando quiserem que o modelo envie ou receba agentes.

Por fim, é importante ressaltar que ambas plataformas têm acesso a todas as informações do agente. Por exemplo, o modelo do JaCaMo poderia acessar qualquer atributo do agente do NetLogo (açúcar, metabolismo e visão) para tornar algo útil na simulação. O contrário é verdadeiro: o NetLogo tem acesso ao arquivo *.asl* de cada agente, possibilitando o uso de algumas informações, como uma crença, para algo útil na simulação. Disponibilizamos vídeos e tutoriais na página do GitHub [de Lima and de Aguiar 2022] sobre o cenário testado que são essenciais para verificar os pontos fortes da arquitetura.

5. Conclusões

Este artigo apresentou uma proposta de arquitetura para auxiliar o desenvolvimento de SMAA utilizando Docker. Essa arquitetura permite a migração de agentes entre modelos que podem rodar em cenários heterogêneos. Além disso, a estrutura permite que o código seja executado dentro de contêineres que contenham a mesma estrutura de operação onde foram desenvolvidos, evitando problemas como programas que rodam na fase de desenvolvimento mas na fase de produção apresentam problemas.

O protótipo apresentado permitiu que os agentes se movimentassem livremente entre os modelos, compartilhando todas as informações do agente com os respectivos modelos e reduzindo a complexidade da adaptação do código, demonstrando que é suficiente, mas simples para entender melhor a abordagem proposta. A migração de agentes entre modelos ocorre em tempo de execução.

Em trabalhos futuros, queremos explorar novos gatilhos que fazem os agentes mudarem de modelo, como limites geográficos e modelos paralelos. Limites geográficos tratam de modelos onde, quando o agente atinge a borda do ambiente, ele passa para o outro modelo. Modelos paralelos são modelos onde o mesmo agente participa de mais de um modelo simultaneamente, mas cada modelo evolui atributos particulares do agente. Além disso, mesmo que a plataforma esteja atualmente rodando em uma máquina local, diversas plataformas (como o AWS Docker da Amazon) permitem que toda a estrutura que está sendo desenvolvida com base em Docker rode na nuvem, possibilitando a portabilidade de aplicações e ampliando o universo de aplicações na arquitetura. Finalmente, a implementação da arquitetura atual suporta duas plataformas de agentes relevantes, NetLogo e JaCaMo. No entanto, queremos ir além e oferecer suporte a outras plataformas de agentes, como JADE (baseado em Java) ou Mesa (baseado em Python).

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

Referências

Anderson, C. (2015). Docker [software engineering]. *Ieee Software*, 32(3):102–c3.

Artikis, A. (2011). Dynamic specification of open agent systems. *Journal of Logic and Computation*, 22(6):1301–1334. doi: 10.1093/logcom/exr018.

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. doi: 10.1016/j.scico.2011.10.004.

Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A., and Santi, A. (2022a). Jacamo's github page. Available in: `https://github.com/jacamo-lang/jacamo`. Accessed in 26 jul. 2022.

Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A., and Santi, A. (2022b). Jacamo's github page. Available in: `https://github.com/jacamo-lang/jacamo/blob/master/doc/install.adoc`. Accessed in 26 jul. 2022.

Bordini, R. H., Hübner, J. F., and Tralamazza, D. M. (2006). Using jason to implement a team of gold miners. In *International Workshop on Computational Logic in Multi-Agent Systems*, pages 304–313. Springer. doi: 10.1007/978-3-540-69619-3_18.

Dähling, S., Razik, L., and Monti, A. (2021). Enabling scalable and fault-tolerant multi-agent systems by utilizing cloud-native computing. *Autonomous Agents and Multi-Agent Systems*, 35(1):1–27. doi: 10.1007/s10458-020-09489-0.

Dalpiaz, F., Chopra, A. K., Giorgini, P., and Mylopoulos, J. (2010). Adaptation in open systems: Giving interaction its rightful place. In Parsons, J., Saeki, M., Shoval, P., Woo, C., and Wand, Y., editors, *Proc. of the Conceptual Modeling – ER 2010*, pages 31–45, Berlin, Heidelberg. Springer Berlin Heidelberg. doi: 10.1007/978-3-642-16373-9_3.

de Lima, G. L. and de Aguiar, M. S. (2022). Architecture's github page. Available in: `https://github.com/GustavoLLima/open_mas_docker_opt`. Accessed in 26 oct. 2022.

Demazeau, Y. and Costa, A. R. (1996). Populations and organizations in open multi-agent systems. In *Proceedings of the 1st National Symposium on Parallel and Distributed AI (PDAI'96)*, pages 1–13, India. University of Hyderabad.

Epstein, J. M. and Axtell, R. L. (1996). *Growing artificial societies: Social Science from the Bottom Up*. Complex Adaptive Systems. Bradford Books, Cambridge, MA.

Franceschelli, M. and Frasca, P. (2018). Proportional dynamic consensus in open multi-agent systems. In *Proc. of the IEEE Conference on Decision and Control (CDC)*, pages 900–905, Miami, FL, USA. IEEE. doi: 10.1109/CDC.2018.8619639.

Franceschelli, M. and Frasca, P. (2021). Stability of open multiagent systems and applications to dynamic consensus. *IEEE Transactions on Automatic Control*, 66(5):2326–2331. doi: 10.1109/TAC.2020.3009364.

Gonzalez-Palacios, J. and Luck, M. (2006). Towards compliance of agents in open multi-agent systems. In *Proc. of the International Workshop on Software Engineering for Large-Scale Multi-agent Systems*, pages 132–147, Shanghai, China. Springer. doi: 10.1007/978-3-540-73131-3_8.

Grinberg, M. (2018). *Flask web development: developing web applications with python*. O'Reilly Media, Inc. doi: 10.5555/2621997.

Hattab, S. and Lejouad Chaari, W. (2021). A generic model for representing openness in multi-agent systems. *The Knowledge Engineering Review*, 36:e3. doi: 10.1017/S0269888920000429.

Hendrickx, J. M. and Martin, S. (2016). Open multi-agent systems: Gossiping with deterministic arrivals and departures. In *Proc. of the 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1094–1101, Monticello, IL, USA. IEEE. doi: 10.1109/ALLERTON.2016.7852357.

Houhamdi, Z. and Athamena, B. (2020). Collaborative team construction in open multi-agents system. In *Proc. of the 21st International Arab Conference on Information Technology (ACIT)*, pages 1–7, Giza, Egypt. IEEE. doi: 10.1109/ACIT50332.2020.9300116.

Huynh, T. D., Jennings, N. R., and Shadbolt, N. (2004). Developing an integrated trust and reputation model for open multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 13:119–154. doi: 10.1007/s10458-005-6825-4.

Jamroga, W., Meski, A., and Szreter, M. (2013). Modularity and openness in modeling multi-agent systems. *Electronic Proceedings in Theoretical Computer Science*, 119:224–239. doi: 10.4204/eptcs.119.19.

Jiang, W., Chen, Y., and Charalambous, T. (2021). Consensus of general linear multi-agent systems with heterogeneous input and communication delays. *IEEE Control Systems Letters*, 5(3):851–856. doi: 10.1109/LCSYS.2020.3006452.

Kaffille, S. and Wirtz, G. (2006). Modeling the static aspects of trust for open mas. In *2006 International Conference on Computational Intelligence for Modelling Control and Automation and International Conference on Intelligent Agents Web Technologies and International Commerce (CIMCA'06)*, pages 186–186, Australia. IEEE, IEEE. doi: 10.1109/CIMCA.2006.150.

Noh, S. and Park, J. (2020). System design for automation in multi-agent-based manufacturing systems. In *Proc. of 20th International Conference on Control, Automation and Systems (ICCAS)*, pages 986–990, Busan, Korea (South). IEEE. doi: 10.23919/ICCAS50221.2020.9268357.

Paurobally, S., Cunningham, J., and Jennings, N. R. (2003). Ensuring consistency in the joint beliefs of interacting agents. In *Proc. of 2nd International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Australia, Proceedings*, pages 662–669, Melbourne, Australia. ACM. doi: 10.1145/860575.860682.

Perles, A., Crasnier, F., and Georgé, J.-P. (2018). Amak - a framework for developing robust and open adaptive multi-agent systems. In Bajo, J., Corchado, J. M., Navarro Martínez, E. M., Osaba Icedo, E., Mathieu, P., Hoffa-Dabrowska, P., del Val, E., Giroux, S., Castro, A. J., Sánchez-Pi, N., Julián, V., Silveira, R. A., Fernández, A., Unland, R., and Fuentes-Fernández, R., editors, *Proc. of International Conference on Practical Applications of Agents and Multi-Agent Systems – Highlights of Practical Applications of Agents, Multi-Agent Systems, and Complexity: The PAAMS Collection*, pages 468–479, Cham. Springer International Publishing. doi: 10.1007/978-3-319-94779-2_40.

Pfeifer, V., Passini, W. F., Dorante, W. F., Guilherme, I. R., and Affonso, F. J. (2021). A multi-agent approach to monitor and manage container-based distributed systems. *IEEE Latin America Transactions*, 20(1):82–91. doi: 10.1109/TLA.2022.9662176.

Ramirez, W. A. L. and Fasli, M. (2017). Integrating netlogo and jason: a disaster-rescue simulation. In *2017 9th Computer Science and Electronic Engineering (CEEC)*, pages 213–218. IEEE. doi: 10.1109/CEEC.2017.8101627.

Singh, M. P. and Chopra, A. K. (2009). Programming multiagent systems without programming agents. In *Proc. of the International Workshop on Programming Multi-Agent Systems*, pages 1–14, Budapest, Hungary. Springer. doi: 10.1007/978-3-642-14843-9_1.

Tisue, S. and Wilensky, U. (2004). Netlogo: A simple environment for modeling complexity. In *International conference on complex systems*, volume 21, pages 16–21. Citeseer.

Turnbull, J. (2014). *The Docker Book: Containerization Is the New Virtualization*. James Turnbull, Melbourne, Australia.

Uez, D. M. (2018). *Open AEOlus: um método para especificação de sistemas multiagentes abertos*. Phd thesis, phd on automation and systems engineering, Federal University of Santa Catarina, Centro Tecnológico, Florianópolis. Available in: https://repositorio.ufsc.br/handle/123456789/205584.

van Eijk, R. M., de Boer, F. S., Van Der Hoek, W., and Meyer, J.-J. C. (1999). Open multi-agent systems: Agent communication and integration. In *Proc. of the International Workshop on Agent Theories, Architectures, and Languages*, pages 218–232, Orlando, Florida, USA. Springer. doi: 10.1007/10719619_16.

Um Mapeamento de Sistemas Multiagentes e Ativos Digitais com uma Proposta de Aplicação em Smart Parking

Gabriel L. M. Oliveira[1], **Gleifer Vaz Alves**[1], **Nilson Mori Lazarin**[2]

[1]Universidade Tecnológica Federal do Paraná (UTFPR)
Ponta Grossa, PR - Brasil

[2]Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (Cefet/RJ)
Rio de Janeiro, RJ – Brazil

gabrieloliveira.2020@alunos.utfpr.edu.br

gleifer@utfpr.edu.br, nilson.lazarin@cefet-rj.br

Abstract. *With the growth of the smart cities' paradigm, other scopes are starting to be explored. For example, improving the control or management over parking spaces using Multi-Agent Systems (MAS) is possible in the context of Smart Parking. Integrating MAS and Blockchain can improve efficiency, transparency, and user experience in parking operations, contributing to developing more intelligent and sustainable cities. This paper presents research mapping papers in the MAS area that use BDI agents and digital assets applied to Smart Parking. Besides, an implementation proposal based on these concepts is given.*

Resumo. *Com o crescimento do paradigma de cidades inteligentes, outros escopos começam a ser explorados. No contexto de Smart Parking, por exemplo, é possível melhorar o controle ou a gestão de reserva de vagas, através da aplicação de Sistemas Multiagentes (SMA). Integrar SMA e Blockchain neste contexto pode proporcionar um aprimoramento da eficiência, transparência e experiência do usuário nas operações de estacionamento, contribuindo para o desenvolvimento de cidades mais inteligentes e sustentáveis. Este artigo apresenta um mapeamento de artigos a respeito da área de SMA que usam agentes BDI e ativos digitais aplicados em um Smart Parking. Além disso, uma proposta de implementação baseado nesses conceitos é descrita.*

1. Introdução

Na última década, a dependência da nossa sociedade em sistemas inteligentes descentralizados aumentou drasticamente. Todo dia itens como smartphones, dispositivos vestíveis, veículos e eletrodomésticos, estão aumentando suas capacidades computacionais e comunicativas, isso tem mudado hábitos e costumes na sociedade contemporânea [Calvaresi et al. 2019]. Neste contexto, os Sistemas Multiagentes (SMA) têm recebido destaque como uma abordagem promissora para a resolução de problemas complexos, que requerem a colaboração entre diferentes entidades autônomas.

Na área de Inteligência Artificial, particularmente a de SMA, visa o desenvolvimento de entidades computacionais autônomas capazes de ações flexíveis e autônomas em contextos dinâmicos, ou seja, que geralmente contém outros agentes possivelmente

desenvolvidos por diferentes partes e distribuídos em uma rede [Papi et al. 2022]. Levando isso em consideração, uma área particularmente interessante dentro dos Sistemas Multiagentes é a teoria dos agentes BDI (*Belief-Desire-Intention*) [Bratman 1987]. Essa teoria, inspirada na psicologia cognitiva, descreve atitudes mentais como crenças (*Beliefs*) sobre o mundo, desejos (*Desires*) a serem alcançados e intenções (*Intentions*) que direcionam suas ações. O modelo BDI tem sido utilizado para modelar a cognição humana e também têm sido aplicado com sucesso em SMA [Rao 1996].

A ascensão dos ativos digitais, como criptomoedas e tokens, têm proporcionado novas possibilidades para a negociação entre agentes autônomos. Esses ativos digitais permitem a representação e transferência de valor de forma descentralizada e segura, sem a necessidade de intermediários tradicionais [Minarsch et al. 2022]. Com isso, a negociação entre agentes BDI por meio de ativos digitais torna-se uma linha de pesquisa promissora, pois a tecnologia de Sistemas Multiagentes pode oferecer ferramentas para desenvolver aplicações em domínios envolvendo transação de ativos [Papi et al. 2022].

Um mapeamento sistemático foi desenvolvido de modo a identificar lacunas de pesquisa e reunir conteúdos sobre a utilização de ativos digitais na negociação entre agentes, selecionando artigos que serão futuramente explorados para o desenvolvimento deste trabalho. Para execução desta tarefa foi usada a metodologia de [Kitchenham and Charters 2007] e a ferramenta Parsifal[1].

Com o resultado deste mapeamento, buscou-se apresentar uma proposta de trabalho para atacar a lacuna de pesquisa encontrada, a falta de estudos que abordam SMA BDI que utilizam ativos digitais em um ambiente de Smart Parking. O *middleware* Velluscinum será utilizado como base para o modelo de transações de ativos digitais, esse *middleware* utiliza Distributed Ledger Technologies (DLT), que é um banco de dados com características de descentralização e segurança, ideal para a modelagem proposta para um SMA [Lazarin et al. 2023]. Destaca-se que foram analisados outros *middlewares*, porém apenas o Velluscinum utiliza agentes BDI.

Considerando o avanço de Smart Parkings em Cidades Inteligentes, foi selecionado como cenário de testes para a abordagem deste trabalho, a aplicação em um sistema de Smart Parking, onde há negociações de vagas entre agentes, como abordado em [Mellado et al. 2021].

Em suma, este trabalho se propõe a investigar estudos e desenvolver uma modelagem de um SMA BDI utilizando um *middleware* para a negociação de ativos digitais em um ambiente de Smart Parking. Na Seção 2 é apresentado o mapeamento sistemático. A Seção 3 descreve o modelo de implementação proposto. Por fim, na Seção 4 são apresentadas as considerações finais.

2. Mapeamento Sistemático

Para identificar possíveis lacunas de pesquisas um mapeamento sistemático a respeito de Sistemas Multiagentes BDI que utilizam ativos digitais em sua negociação foi realizado conforme descrito nesta seção.

[1]https://parsif.al/

2.1. Planejamento

Observando que a necessidade do mapeamento já foi citada em seções anteriores, foram definidas as seguintes questões de pesquisa: ***(Q1)*** como os agentes normalmente operam no sistema? ***(Q2)*** como ocorre a utilização dos ativos digitais pelos agentes no sistema? ***(Q3)*** em um sistema que utiliza *blockchain*, qual a finalidade dos agentes? ***(Q4*** quais são os desafios descritos nos estudos?

2.2. Execução

A etapa de execução é definida por buscar estudos, selecionar os estudos relevantes, avaliar a qualidade dos estudos selecionados, extrair os dados relevantes de cada estudo e por fim a interpretação do resultado.

2.2.1. Critérios de Inclusão e Exclusão

Nesta etapa foram definidos os seguintes critérios de inclusão de estudos: i) estudos que tratam SMA BDI; ii) estudos que tratam SMA que utilizam ativos digitais; iii) estudos que tratam SMA que utilizam *blockchain*. E os seguintes critérios de exclusão de estudos: i) artigos resumidos; ii) estudos duplicados; iii) estudos secundários ou terciários; iv) estudos em qualquer idioma que não seja inglês.

2.2.2. Pesquisa por String de Busca

Prosseguindo o protocolo, em conjunto de algumas palavras-chave com seus sinônimos é obtida uma string de busca, com ela é possível fazer a pesquisa dos trabalhos que podem englobar o tema requerido nas bases de dados de bibliotecas digitais. Para busca de artigos foram usadas as principais bases de artigos disponíveis que funcionam facilmente em conjunto com a ferramenta Parsifal e que também viabilizam o uso dos filtros de exclusão.

Assim as seguintes palavras-chave foram definidas: BDI Agents, Blockchain, Digital Ledger Technology, Multi Agent System Trade e Smart Contract. Agregando alguns sinônimos obtém-se a seguinte string de busca: *("BDI Agents" OR "BDI agent" OR "Belief-Desire-Intention agents" OR "Intelligent agent" OR "Multi Agent System Trade" OR "Multi-Agent System" OR "Multiagent System") AND ("Blockchain" OR "Digital assets" OR "Digital Ledger Technology" OR "DLT")*. Utilizando as bibliotecas digitais (ver Tabela 1) e inserções de estudos de forma manual, foram obtidos 615 estudos, ao aplicar o primeiro filtro da remoção de trabalhos duplicados, restaram 492 estudos.

Source	**Imported Studies**
ACM Digital Library	104
IEEE Xplore	39
ISI Web of Science	86
Manual	2
Scopus Digital Library	384
Total	615

Tabela 1. Tabela de Resultados em Cada Base de Dados

2.2.3. Aplicação de Critérios de Inclusão e Exclusão

Aplicando os critérios de inclusão e exclusão tem-se um filtro da maior parte dos estudos de forma que somente 39 trabalhos restaram nesta etapa. Também foram incluídos dois novos artigos manualmente conforme o critério de inclusão. Foram incluídos os artigos [Lazarin et al. 2023] e [Liu et al. 2021], sendo que o primeiro é de fato uma das referências principais para este trabalho de pesquisa.

2.2.4. Avaliando a Qualidade dos Artigos

Para esta etapa foi aplicado um novo filtro levando em consideração a plataforma Sucupira, esta plataforma é usada para avaliar a qualidade dos periódicos procurando-os através do código ISSN e obtendo uma avaliação, além disso, foi feita uma leitura mais detalhada do resumo comparando os artigos e usando como base as questões de pesquisa.

Neste processo foram comparados os artigos entre si, e através das propostas e dos temas foi possível remover aqueles que não tinham aderência com as questões de pesquisa proposta. Com isso, foram selecionados os seguintes nove artigos:

1. “A Blockchain integration to support transactions of assets in multi-agent systems” [Papi et al. 2022];
2. “Trusted Registration, Negotiation, and Service Evaluation in Multi-Agent Systems throughout the Blockchain Technology” [Calvaresi et al. 2018a];
3. “The good, the bad, and the ethical implications of bridging blockchain and multi-agent systems” [Calvaresi et al. 2019];
4. “A Distributed Electricity Trading System in Active Distribution Networks Based on Multi-Agent Coalition and Blockchain” [Luo et al. 2019];
5. “Enterprise Platform of Logistics Services Based on a Multi-Agents Mechanism and Blockchains” [Liu et al. 2021];
6. “Trust-Based Smart Contract for Automated Agent to Agent Communication” [Mhamdi et al. 2022];
7. “Resilience Network Controller Design for Multi-Domain SDN: A BDI-based Framework” [Song et al. 2022];
8. “Reputation Management in Multi-Agent Systems Using Permissioned Blockchain Technology” [Calvaresi et al. 2018b];
9. “Velluscinum: A Middleware for Using Digital Assets in Multi-Agent Systems” [Lazarin et al. 2023].

Como última etapa para selecionar os melhores artigos entre os escolhidos, foi feito um último filtro, a leitura e aplicação de questões classificatórias que determinaram uma pontuação para cada estudo, quanto maior a pontuação, melhor o artigo se encaixa com o escopo do projeto. Foram selecionadas cinco perguntas e cada uma pode somar 1.0 se a resposta for sim, 0.5 se parcialmente e 0.0 se não.

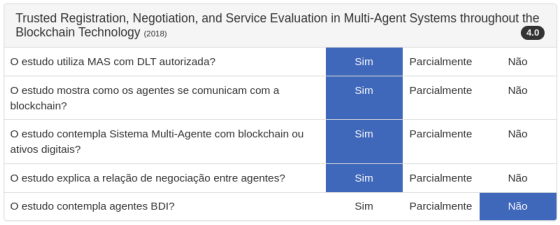
Seguem abaixo as questões utilizadas para a qualificação, o resultado geral da classificação desta etapa (Figura 1) e por fim os artigos mais bem classificados e suas pontuações (Figuras 2a, 2b, 2c e 2d).

1. O estudo utiliza Sistema Multiagente com DLT autorizada?

2. O estudo mostra como os agentes se comunicam com a *blockchain*?
3. O estudo contempla Sistema Multiagente com *blockchain* ou ativos digitais?
4. O estudo explica a relação de negociação entre agentes?
5. O estudo contempla agentes BDI?

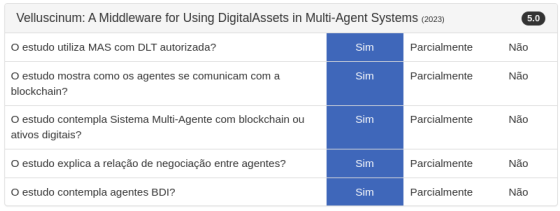
Title	Quality Score
Resilience Network Controller Design for Multi-Domain SDN: A BDI-based Framework	2.5
Trust-Based Smart Contract for Automated Agent to Agent Communication	3.0
Reputation Management in Multi-Agent Systems Using Permissioned Blockchain Technology	3.0
The good, the bad, and the ethical implications of bridging blockchain and multi-agent systems	4.0
A Blockchain integration to support transactions of assets in multi-agent systems	4.0
A Distributed Electricity Trading System in Active Distribution NetworksBased on Multi-Agent Coalition and Blockchain	2.0
Velluscinum: A Middleware for Using DigitalAssets in Multi-Agent Systems	5.0
Enterprise Platform of Logistics Services Based on a Multi-Agents Mechanism and Blockchains	2.5
Trusted Registration, Negotiation, and Service Evaluation in Multi-Agent Systems throughout the Blockchain Technology	4.0

Figura 1. Resultado da Qualificação Geral.

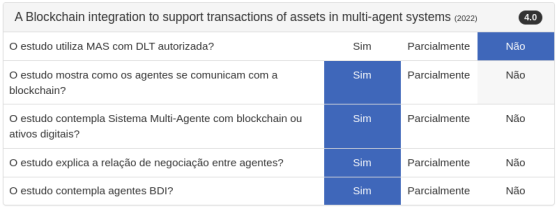


(a) Calvaresi et al. 2018a.

(b) Calvaresi et al. 2019.



(c) Lazarin et al. 2023.



(d) Papi et al. 2022.

Figura 2. Resultado da Qualificação

2.3. Análise dos Resultados

Com base na leitura dos artigos, serão respondidas as questões de pesquisas apresentadas anteriormente.

1. **Como os agentes normalmente operam no sistema?**
   Os agentes inteligentes em sistemas que usam *blockchain*, em geral, são abordados como gerenciadores de recursos, por exemplo, distribuição de energia

[Luo et al. 2019], ou como agentes que trocam serviços entre si e negociam via transferência de ativos [Papi et al. 2022].

2. **Como ocorre a utilização dos ativos pelos agentes no sistema?**
   Existem vários tipos de interações abordados, mas em resumo, os agentes podem fazer trocas e negociações através da *blockchain* usando, em sua maior parte, contratos vinculados ao registro (*ledger*).
3. **Em um sistema que utiliza *blockchain*, qual a finalidade dos agentes?**
   Estabelecer uma relação através de contratos inteligentes, onde assim é possível os agentes se relacionarem entre si usando sistemas de transações e validações de transações feitas por ativos digitais.
4. **Quais são os desafios descritos no estudo?**
   Levando em consideração o tema principal do estudo, nota-se que não se tem resultados relacionados a Sistemas Multiagentes que usam *blockchain* em um cenário de Smart Parking, mostrando que existe uma lacuna de pesquisa nessa área que pode ser explorada. Há, sim, propostas de sistemas que utilizam *blockchain* porém em outros cenários. E tais sistemas podem ser usados como base para uma possível adaptação ou integração em um Smart Parking.

2.4. Conclusão do Mapeamento

Ao final deste estudo, nota-se que os artigos selecionados são de 2018 ou mais recentes e principalmente considerando a parte final de qualificação de artigos, pode-se concluir que existem vários estudos na área de SMA que utilizam ativos digitais inclusos, porém é notório que estudos que utilizam DLT autorizada e agentes BDI são raros, como mostra a qualificação geral dos artigos na Figura 1, inclusive somente o estudo base deste artigo [Lazarin et al. 2023] contempla todas as qualidades definidas como desejáveis para este projeto (Fig. 2c).

Os trabalhos avaliados (ilustrados nas Fig. 2a, 2b, 2c e 2d), podem auxiliar na continuidade deste trabalho. Os artigos [Calvaresi et al. 2018a] e [Calvaresi et al. 2019] exemplificam um SMA (mostrando o relacionamento entre eles) integrado a *blockchain* que usa DLT autorizada de forma completa e debatendo sobre a transparência sobre estes sistemas, já [Papi et al. 2022], usam um SMA integrado a *blockchain* que implementa agentes BDI de forma centralizada, e por fim temos [Lazarin et al. 2023] que apresenta um *middleware* chamado Velluscinum utilizado para a negociação entre os agentes BDI, sendo que a negociação conta com a DLT autorizada.

Com a leitura dos nove artigos foi possível analisar que alguns estudos abordam temas diferentes, porém grande parte trabalha com Contratos Inteligentes, contratos estes fundamentais para o relacionamento entre os agentes em um SMA. A *blockchain*, abordada em todos os artigos, é implementada de forma descentralizada em sete dos nove estudos, o que expressa uma tendência a sistemas descentralizados ao invés de centralizados.

Chegando ao final desta análise é possível também observar a existência de uma lacuna de pesquisa dentro do escopo de SMA BDI que utilizam ativos digitais em um ambiente de Smart Parking, sendo esta a principal motivação para o desenvolvimento deste trabalho.

3. Modelo Proposto

Dado o resultado do mapeamento sistemático, decide-se que o objetivo geral da segunda parte deste trabalho é estruturar e propor uma modelagem para a integração de um SMA em um ambiente de Smart Parking que utiliza agentes BDI e fazem negociação pelo intermédio de um *middleware* que utiliza ativos digitais, assim como mostra a Figura 3. O elemento *network* demonstrado na figura é como os agentes podem negociar usando o *middleware*, um exemplo de *network* que o *middleware* Velluscinum faz conexão é o BigChainDB [GmbH 2018]. Que é um banco de dados descentralizado que visa adicionar características de blockchain para banco de dados distribuídos.

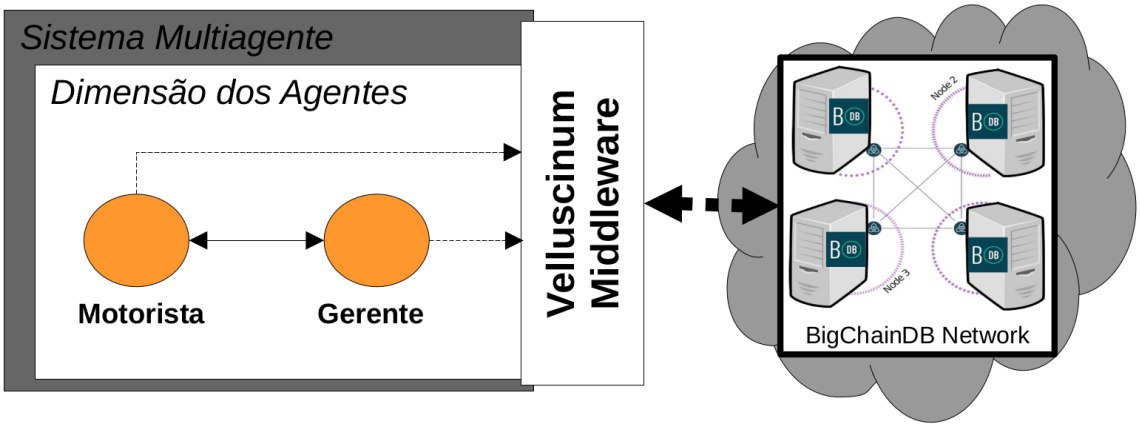


Figura 3. Visão Geral do Modelo.

3.1. Agentes

A modelagem proposta considera dois tipos de agentes: Gerente e Motorista, cada um deles sendo independente em relação ao outro. Além dos agentes, o sistema conta com um ambiente, que é o estacionamento, e com um *middleware* de intermédio para que os agentes possam ter contato com a *blockchain* e transferir os ativos. A negociação entre os agentes do sistema serão divididas em quatro cenários, cada cenário ilustra uma situação diferente do sistema.

Gerente: Faz o controle da entrada e saída de veículos, de vagas e do financeiro. Como única crença este agente tem "*Vagas Disponíveis*", que é a quantidade de vagas que estão livres no estacionamento, já o objetivo é "*Criar Carteira*".
As intenções do gerente são as seguintes: Verificar Vagas Disponíveis; Enviar Mensagem; Enviar Informações da Vaga; Ocupar Vaga; Enviar Ticket Pagamento; Liberar saída; Validar Transferência (valor ou reserva); Gerar Reserva; Transferir Reserva; e Informar ID Reserva.

Motorista: O agente Motorista representa um motorista com seu veículo que pode ter algumas ações dentro desse sistema, ele não contém nenhuma crença. Este agente tem como objetivos: "*Estacionar Carro*", "*Usar Reserva*", "*Fazer Reserva*" e "*Fazer Transferência*", sendo que todos os objetivos envolvem negociações entre o agente Motorista e o Gerente, exceto pelo último, que envolve duas instâncias do próprio agente Motorista. As intenções do Motorista são dispostas entre: Solicitar Vaga; Solicitar Retirada; Solicitar Reserva; Informar (será informado o ID da transferência valor ou reserva); Transferir Reserva; Transferir Valor; Validar Transferência (valor ou reserva); e Sair do Estacionamento.

3.2. Cenários de Aplicação

Aqui são apresentadas as possíveis situações de negociação entre os agentes, de forma a exemplificar como ocorrem as ações e as trocas de mensagens.

- **Cenário 1:** No primeiro cenário, como ilustra a Fig. 4, tem-se um Motorista que chega no estacionamento e tem o objetivo de estacionar, este agente fará a solicitação de vaga e caso tenha vaga ele irá estacionar seu veículo. Quando o Motorista quiser, pode solicitar a retirada de seu veículo e recebe do Gerente um ticket de pagamento. Ao transferir o valor do ticket, o Motorista informa o ID de transferência validado pelo próprio Gerente, com a validação feita a saída é liberada. Destaca-se que para o pagamento do ticket só é utilizado pagamento por valor, ou seja, apenas moedas digitais foram transicionadas.

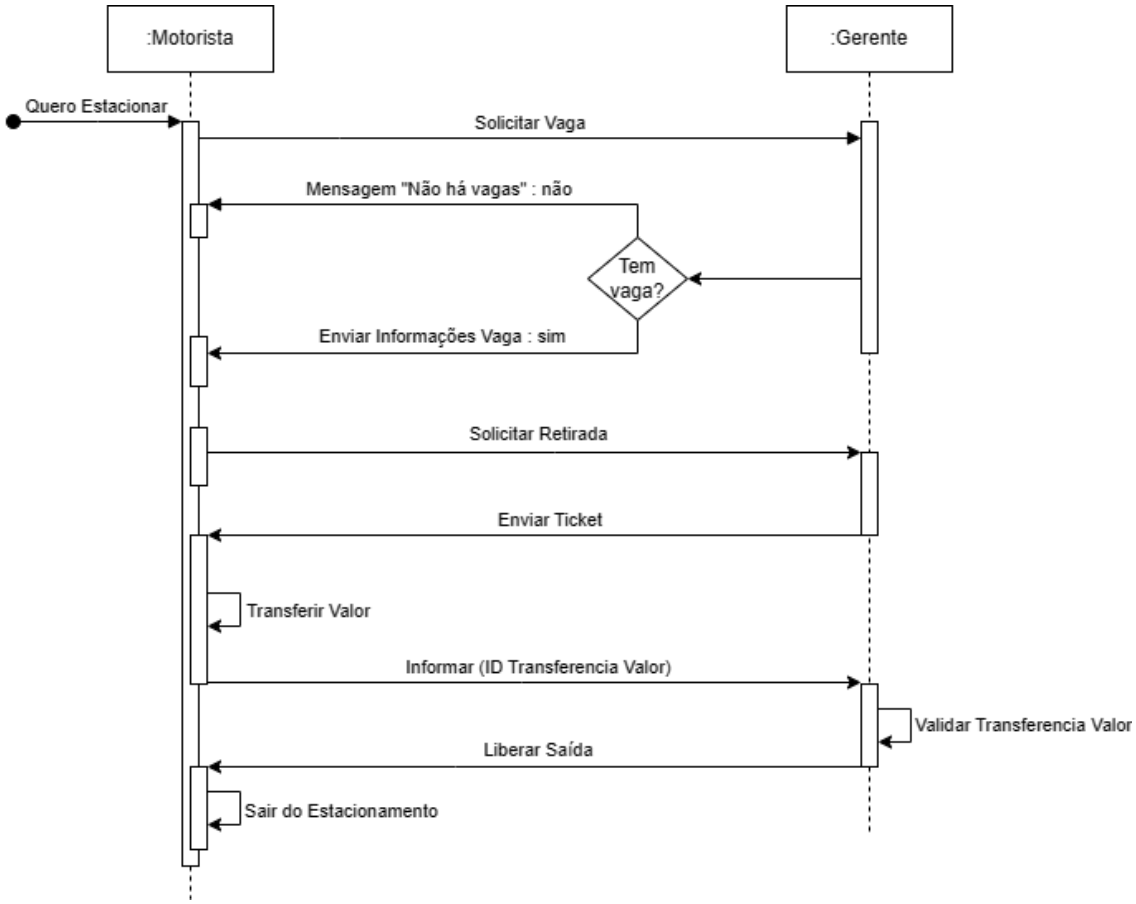


Figura 4. Cenário 1.

- **Cenário 2:** No segundo cenário, tem-se a reserva de vaga, mostrada pela Fig. 5, onde o Motorista quer reservar uma vaga no estacionamento, para isso ele solicita a vaga ao Gerente passando o dia e o horário que ele deseja, ao ter uma vaga disponível o Gerente informa o valor a ser pago, tendo efetuado o pagamento o Motorista informa o ID da transferência feita. Quando recebido o ID da transferência, o Gerente gera a reservva referente a vaga, transfere e informa o ID de transferência da reserva para o Motorista, o mesmo valida essa transferência e assim obtém acesso à vaga reservada.

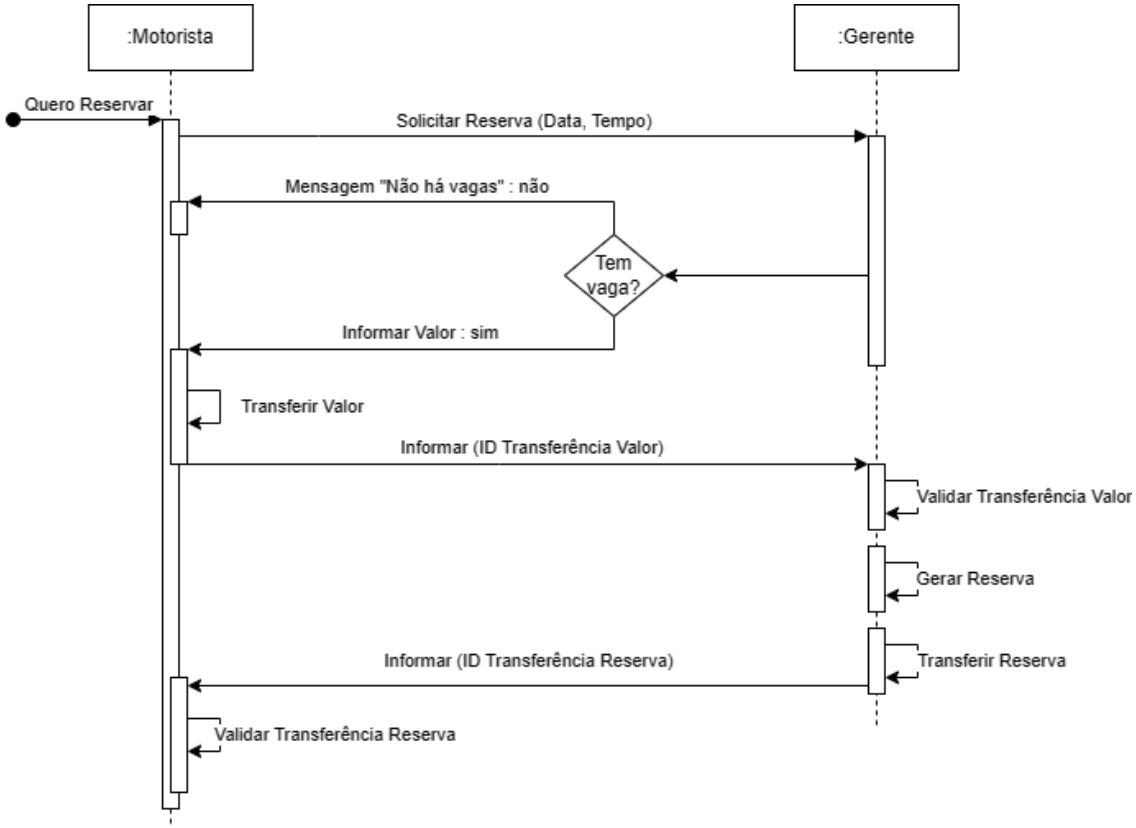


Figura 5. Cenário 2.

- **Cenário 3:** O terceiro cenário representado nesta modelagem é descrito na Fig. 6, neste caso, o Motorista já tem uma reserva previamente feita e deseja usá-la. Ao chegar no estacionamento, o agente Motorista faz a transferência da reserva para o Gerente e informa o ID da transferência da reserva, quando é validado pelo Gerente, o acesso é liberado e é permitido estacionar. Quando o Motorista quiser ele pode solicitar a retirada, ao ocorrer a verificação se será necessário cobrar um valor extra por exceder o tempo estipulado na reserva, o Motorista pode tanto ser liberado para sair diretamente, quanto ter que transferir o valor e informar o ID desta transferência, uma vez que o Gerente validar a transferência o Motorista será liberado.
- **Cenário 4:** No quarto cenário a negociação é feita diretamente entre dois agentes Motoristas, Motorista 1 e Motorista 2. O processo inicia-se com o objetivo de transferir uma reserva feita pelo Motorista 1 e a transferência sendo realizada pelo Motorista 2. O ID da transferência feita pelo Motorista 2 é passado e validado pelo Motorista 1, o mesmo, após a validação, transfere a reserva, ou seja, a reserva, para o outro motorista e logo a seguir informa o ID da transferência da reserva. Ao final o Motorista 2 valida a transferência da reserva.

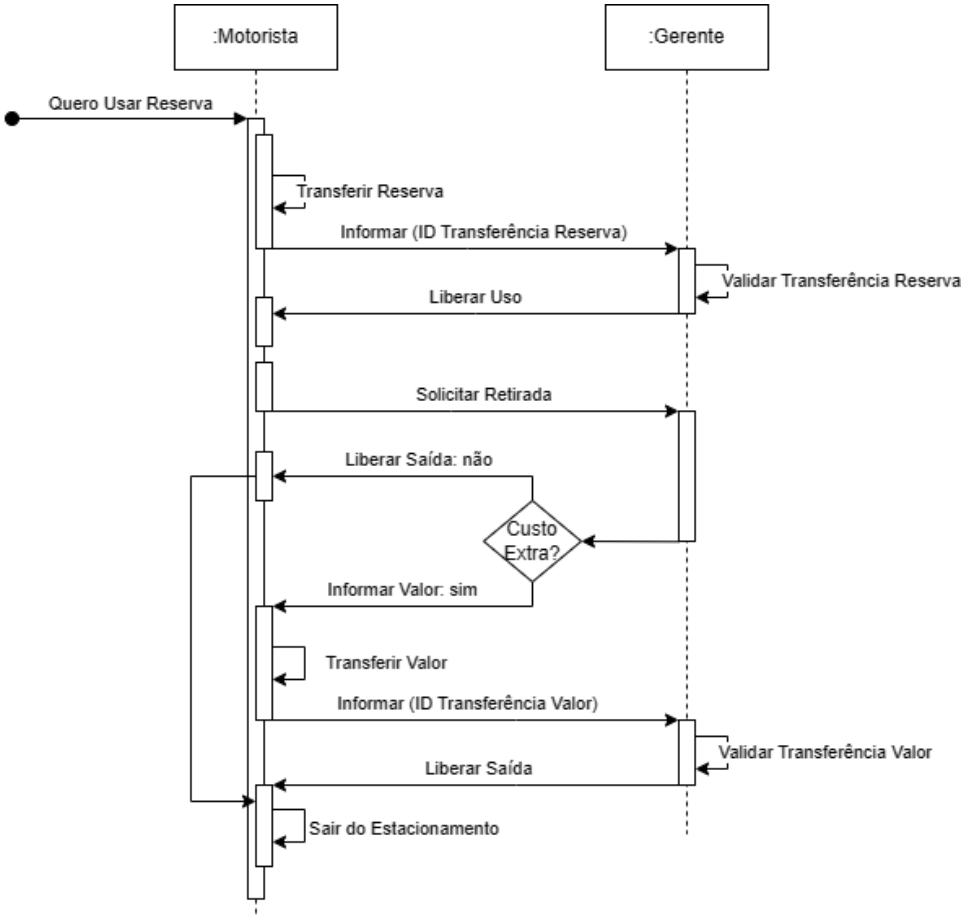


Figura 6. Cenário 3.

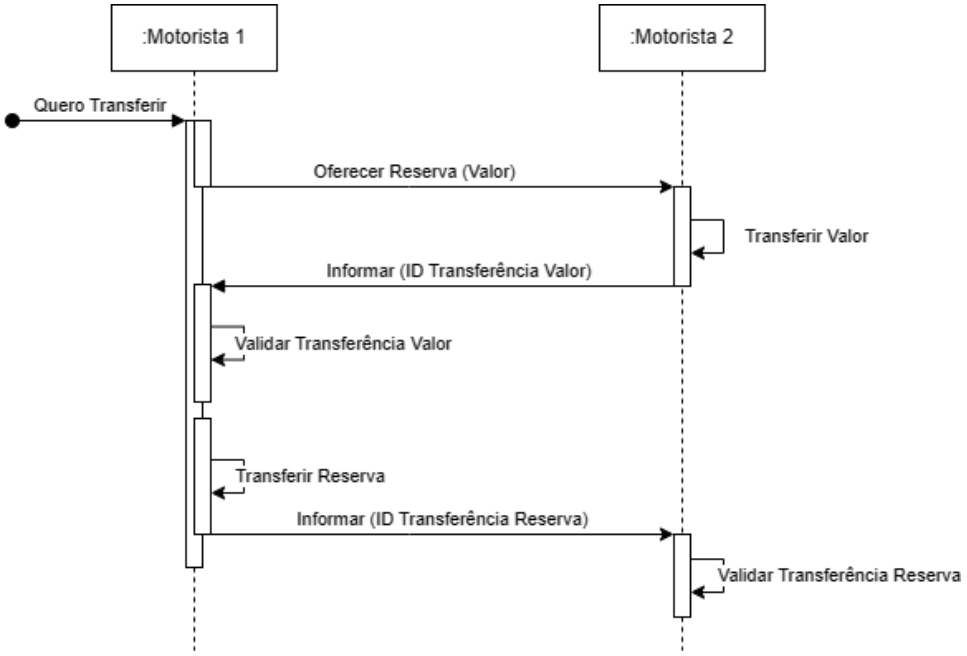


Figura 7. Cenário 4.

4. Conclusão

Neste trabalho foi descrito um mapeamento sistemático, o qual evidenciou uma lacuna de pesquisa na área de SMA BDI que usa ativos digitais em um ambiente de Smart Parking. Dessa forma, o artigo propõe uma modelagem de um SMA com agentes BDI e uso de ativos digitais para a negociação de vagas em um Smart Parking. Nessa modelagem também considera-se a utilização do *middleware* Velluscinum para implantar a comunicação com a *blockchain*, comunicação esta que traz maior segurança para as transações a serem executadas.

A modelagem proposta somada com os estudos selecionados no mapeamento, serão a base de pesquisa para futuros trabalhos, como a implementação dessa modelagem, que pode ser feita utilizando o framework de agentes JaCaMo [Boissier et al. 2016], já que o *middleware* a ser usado se dispõe do mesmo framework e é integrado ao BigChainDB [GmbH 2018]. Para essa implementação, será necessário o estudo a respeito de negociações entre agentes em Smart Parking, para identificar qual método é o mais adequado [Alves et al. 2019], e também o estudo de outros conceitos, por exemplo a precificação de vagas [Mellado et al. 2021] e a relação com grau de confiança dos agentes, como abordado em [Castro et al. 2017].

Referências

Alves, B. R., Alves, G. V., Borges, A. P., and Leitão, P. (2019). Experimentation of Negotiation Protocols for Consensus Problems in Smart Parking Systems. In Mařík, V., Kadera, P., Rzevski, G., Zoitl, A., Anderst-Kotsis, G., Tjoa, A. M., and Khalil, I., editors, *Industrial Applications of Holonic and Multi-Agent Systems*, Lecture Notes in Computer Science, pages 189–202, Cham. Springer International Publishing. DOI: 10.1007/978-3-030-27878-6_15.

Boissier, O., Hübner, J. F., and Ricci, A. (2016). The JaCaMo Framework. In Aldewereld, H., Boissier, O., Dignum, V., Noriega, P., and Padget, J., editors, *Social Coordination Frameworks for Social Technical Systems*, volume 30, pages 125–151. Springer International Publishing, Cham. Series Title: Law, Governance and Technology Series. DOI: 10.1007/978-3-319-33570-4_7.

Bratman, M. (1987). *Intention, plans, and practical reason*. CSLI.

Calvaresi, D., Calbimonte, J.-P., Dubovitskaya, A., Mattioli, V., Piguet, J.-G., and Schumacher, M. (2019). The good, the bad, and the ethical implications of bridging blockchain and multi-agent systems. *Information*, 10(12):363. DOI: 10.3390/info10120363.

Calvaresi, D., Dubovitskaya, A., Retaggi, D., F. Dragoni, A., and Schumacher, M. (2018a). Trusted registration, negotiation, and service evaluation in multi-agent systems throughout the blockchain technology. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pages 56–63. DOI: 10.1109/WI.2018.0-107.

Calvaresi, D., Mattioli, V., Dubovitskaya, A., Dragoni, A. F., and Schumacher, M. (2018b). Reputation management in multi-agent systems using permissioned blockchain technology. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, page 719–725. DOI: 10.1109/wi.2018.000-5.

Castro, L. F. S. D., Alves, G. V., and Borges, A. P. (2017). Using trust degree for agents in order to assign spots in a Smart Parking. *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal*, 6(2):45–55. DOI: 10.14201/ADCAIJ207624555.

GmbH, B. (2018). BigchainDB 2.0 The Blockchain Database. `https://www.bigchaindb.com/whitepaper/`.

Kitchenham, B. A. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report.

Lazarin, N. M., Coelho, I. M., Pantoja, C. E., and Viterbo, J. (2023). Velluscinum: A middleware for using digital assets in multi-agent systems. In Mathieu, P., Dignum, F., Novais, P., and De la Prieta, F., editors, *Advances in Practical Applications of Agents, Multi-Agent Systems, and Cognitive Mimetics. The PAAMS Collection*, pages 200–212, Cham. Springer Nature Switzerland. DOI: 10.1007/978-3-031-37616-0_17.

Liu, S., Hennequin, S., and Roy, D. (2021). Enterprise platform of logistics services based on a multi-agents mechanism and blockchains. *IFAC-PapersOnLine*, 54(1):825–830. DOI: 10.1016/j.ifacol.2021.08.097.

Luo, F., Dong, Z. Y., Liang, G., Murata, J., and Xu, Z. (2019). A distributed electricity trading system in active distribution networks based on multi-agent coalition and blockchain. *IEEE Transactions on Power Systems*, 34(5):4097–4108. DOI: 10.1109/tpwrs.2018.2876612.

Mellado, A., Alves, G. V., Leitão, P., and Borges, A. P. (2021). Um Módulo de Precificação Dinâmica em Sistema Multiagente de um Estacionamento Inteligente. *Revista Eletrônica de Iniciação Científica em Computação*, 19(1). `https://sol.sbc.org.br/journals/index.php/reic/article/view/1779`.

Mhamdi, H., Soufiene, B. O., Zouinkhi, A., Ali, O., and Sakli, H. (2022). Trust-based smart contract for automated agent to agent communication. *Computational Intelligence and Neuroscience*, 2022:1–11. DOI: 10.1155/2022/5136865.

Minarsch, D., Favorito, M., Hosseini, S. A., Turchenkov, Y., and Ward, J. (2022). Autonomous economic agent framework. In Alechina, N., Baldoni, M., and Logan, B., editors, *Engineering Multi-Agent Systems*, pages 237–253, Cham. Springer International Publishing. DOI: 10.1007/978-3-030-97457-2_14.

Papi, F. G., Hübner, J. F., and de Brito, M. (2022). A blockchain integration to support transactions of assets in multi-agent systems. *Engineering Applications of Artificial Intelligence*, 107:104534. DOI: 10.1016/j.engappai.2021.104534.

Rao, A. S. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language. In Carbonell, J. G., Siekmann, J., Goos, G., Hartmanis, J., van Leeuwen, J., Van de Velde, W., and Perram, J. W., editors, *Agents Breaking Away*, volume 1038, pages 42–55. Springer Berlin Heidelberg, Berlin, Heidelberg. Series Title: Lecture Notes in Computer Science. DOI: 10.1007/BFb0031845.

Song, Y., Gao, X., Li, P., and Yang, C. (2022). Resilience network controller design for multi-domain sdn: A bdi-based framework. In *2022 IEEE 95th Vehicular Technology Conference: (VTC2022-Spring)*, pages 1–5. DOI: 10.1109/VTC2022-Spring54318.2022.9860710.

The Tupinambá: An Exercise in Societal Modeling

Antônio Carlos da Rocha Costa

[1]Programa de Pós-Graduação em Filosofia - PPGFil
Pontifícia Universidade Católica do Rio Grande do Sul - PUCRS
90.619-900 Porto Alegre, Brazil

Abstract. *The modeling exercise presented in this paper is a companion to the short papers presented at EMAS 2022 and WESAAC 2022. It illustrates the* societal approach *to multiagent systems by formally modeling some of the main components of the* Tupinambá *society, a tribal society that lived in the territory of Brazil at the time of the first attempts of its occupation and colonization, in beginning the 16th century, by the Portuguese and other Europeans.*
Keywords: *Agent societies. Societal architectures. Society modeling languages.*

Resumo. *O estudo de caso apresentado neste artigo é um complemento dos artigos curtos apresentados no EMAS 2022 e WESAAC 2022. Ele ilustra a* abordagem societal *a sistemas multiagentes através da modelagem formal de alguns dos principais componentes da sociedade dos* Tupinambá*, uma sociedade tribal que vivia no território do Brasil na época das primeiras tentativas de sua ocupação e colonização, no início do século XVI, pelos Portuguêses e outros Europeus.*
Palavras-chave: *Sociedades de agentes. Arquiteturas societais. Linguagens de modelagem de sociedades.*

1. Introduction

This paper presents a modeling exercise illustrating the idea that *agent societies* (i.e., multiagent systems structured in terms of *societal* architectures, not just *organizational* architectures [Costa 2022a, Costa 2022b]) are the appropriate architectural forms for supporting the conception, design, and implementation of *full-fledged multiagent systems*, that is, MAS that are able to computationally model all the essential characteristics of *general* societal systems.

The paper is structured as follows.

Section 2 exposes, in general terms, the features that should be taken as central to any *societal architectural model* for MAS.

Section 3 gives the essential details of the *agent society modeling languages* used in this work, namely: TinySML, for modeling the structural and operational features of the society, and TinyIML, for modeling its ideological features[1].

Section 4 presents the particular modeling exercise which is the subject of the paper: the sketching of a *formal societal model* of the main components of the *Tupinambá*, a tribal society that lived in the territory of Brazil previously to the first attempts of its occupation and colonization, in beginning the 16th century, by the Portuguese and other Europeans.

Section 5 is the Conclusion.

[1]See [Costa 2022c] for some more details on those modeling languages.

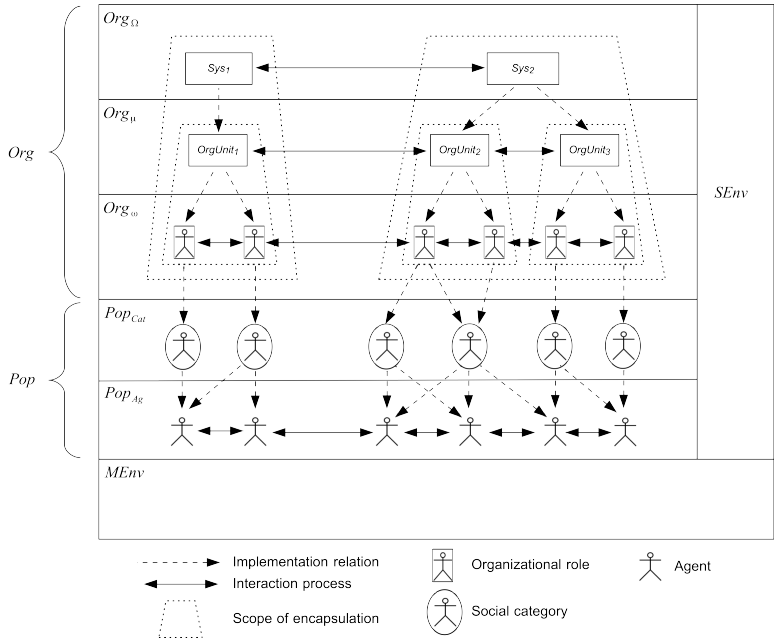


Figure 1. Basic components of an agent society.

2. The Organizational and Societal Approaches to Agent Societies

2.1. The Organizational Approach to Agent Societies

We say that a MAS has an *organizational architecture* whenever it is structured in terms of two core components, Pop and Org, cast as *architectural levels* (see Fig. 1):

- Pop, the *Populational Level*, composed of two sub-levels:
  - Pop_{Ag}, the sub-level of the *Populational Agents*: the very set of *agents*;
  - Pop_{Cat}, the sub-level of the *Populational Categories*: the set of *social categories* (or *strata*) of agents, such as: economic statuses, educational levels, professional experiences, ethnic groups, religious affiliations, gender identities etc.;
- Org, the *Organizational Level*, composed of two sub-levels:
  - Org_{ω}, the *Micro-organizational level*: the network of the *organizational roles* that the agents may perform;[2]
  - Org_{μ}, the *Meso-organizational level*: the network of the (possibly, hierarchically recursive) *organization units* (groups, organizations, institutions etc.), each implemented by a subset of *organizational roles*.

2.2. The Societal Approach to Agent Societies

2.2.1. Structural Features

The concept of *societal architecture* extends the *organizational architecture* with one organizational sub-level, placed above the two previous ones (see Fig. 1):

[2]Notice that *organizational roles* are not *immediately* allowed to be implemented by *agents*, as is often the case in the *organizational approach*, where agents are usually specifically designed to meet the requirements of the organizational roles they will implement. In the *societal approach*, organizational roles are allowed to be implemented by agents only on the condition of the latter's belonging to particular *social categories*. In other words, in the *societal approach* one or more *social categories* are taken to operate as *requirements* that agents have to meet in order to be allowed to perform a given social role.

- Org_{Ω}, the *Macro-organizational level*: the network of the (possibly, hierachically recursive) *societal systems*, each implemented by a (possibly, hierachically recursive) network of *organizational units*.

2.2.2. Functional Features

The following are examples of *societal functions* that *societal systems* may perform in agent societies[3].

• **Ideological System** We call *ideological system* the societal system that comprises *all the cultural elements* present in the society, impacting the behaviors and interactions of the social actors: morality, law, customs, traditional conceptions etc., which are assumed to be symbolically represented in the *Symbolic Environment* (see Fig. 1), as explained in, e.g., [Costa 2015, Costa 2016, Costa 2020, Costa 2021]).

• **Legal System** The *legal system* of an agent society comprises two main sub-systems (the *legislative* and the *judiciary*), responsible for managing the set of *positive norms* (the *legal order*) that *regulates* the social processes of the society.

• **Political System** The *political system* of an agent society can be characterized, in its minimal functionality, in the way proposed in [Easton 1965], namely, as the societal system responsible for the *authoritative allocation of resources* among social actors.

• **Economic System** The *economic system* comprises two main subsystems, the *production* and the *distribution* subsystems. The *production subsystem* can be characterized as the societal system that regiments a set of social actors (the *producers*) in order to continuously generate new objects in the society's *material* and *symbolic* environments, possibly consuming for that purpose some of the objects available, at each time, in those environments. The *distribution subsystem* can be characterized as the societal system that regiments a set of social actors (the *distributors*) in order to continuously distribute, for consumption, among the set of all society's social actors, the objects produced by the production system.

• **Educational System** The *educational system* is the societal system that regiments a set of social actors (the *educators*) in order to capacitate some agents to participate in some of the society's *societal systems* (including the educational system itself).

2.3. Inter-Societal Systems

In the same way that *organization units* are constituted by *networks of organizational roles*, articulated by interation processess, and that *societal systems* are essentially constituted by *networks of organization units*, also articulated by interaction processes, *agent societies* may give rise to *networks* articulated by *societal interaction processes*, thus constituting *inter-societal systems* (see [Costa 2017]).

[3]Notice that we use the concept of *social actor* as a general concept for any of the architectural components of the societal architecture (agents, social roles, organization units, societal system), including the entire *agent society* itself.

Inter-societal systems should be conceived analogously to the way agent societies are conceived, that is, on the basis of a *core structure* given by a *populational component* and an *organizational component*, the latter composed, in an apropriate corresponding way, of three levels: the micro, meso, and macro *inter-societal levels*.

3. The Core Concetps of the Modeling Languages

3.1. The Core Concetps of TinySML

The core concepts of TinySML duplicate formally the *sets of components* that constitute the *societal architecture* of any agent society $AgSoc$ or inter-societal system $InterSoc$, namely:

- $\mathbf{Agents}$, the set of *agents*;
- $\mathbf{SocRole}$, the set of *social roles*:
- $\mathbf{OrgUnit}$, the set of *organization units*;
- $\mathbf{SocSys}$, the set of *social systems*;
- $\mathbf{AgSoc}$, the set of *agent societies*.

Besides, *exchange processes* and other *relations* between those architectural components also belong to this set of core concepts.

3.2. The Core Concepts of TinyIML

The two *basic universes* on which the *ideological system* of an agent society $AgSoc$ operates are: $\mathbf{SocAct}$, the universe of the *social actors* of the society, and $\mathbf{TypeCond}$, the universe of *types of conducts* that those social actors may perform in the society.

The universe of social actors $\mathbf{SocAct}$ is the same universe of social actors of the TinySML language, given by:

$$\mathbf{SocAct} = \mathbf{Ag} \cup \mathbf{OrgRole} \cup \mathbf{OrgUn} \cup \mathbf{SocSys} \cup \mathbf{AgSoc}$$

The universe of *segments* of social actors is given by[4]:

$$\mathbf{Segm} = \wp(\mathbf{SocAct})$$

The universe of *ideological envisagements* is given by

$$\mathbf{IdeoEnvis} = \mathbf{SegmEnvis} \cup \mathbf{NormEnvis} \cup \mathbf{ValuatEnvis} \cup \mathbf{QualifEnvis}$$

where:

- $\mathbf{SegmEnvis}$ is the universe of *segmenting envisagements*;
- $\mathbf{NormEnvis}$ is the universe of *normative envisagements*;
- $\mathbf{ValuatEnvis}$ is the universe of *valuating envisagements*;
- $\mathbf{QualifEnvis}$ is the universe of *qualifying envisagements*.

The formal structures of the various envisagements are given in [Costa 2015], but they are exemplified, intuitively, in the modeling experiment below.

The universe of ideological frameworks is given by:

$$\mathbf{IdeoFrmwrk} \subseteq \mathbf{SegmEnvis} \times \mathbf{NormEnvis} \times \mathbf{ValuatEnvis} \times \mathbf{QualifEnvis}$$

so that for any particular ideological framework:

[4] $\wp(X)$ denotes the powerset of the set X.

$$ideoFrmwrk = (segmEnvis, normEnvis, valuatEnvis, qualifEnvis) \in \mathbf{IdeoFrmwrk}$$

the segmenting envisagement $segmEnvis$ is said to be the *reference segment* of $ideoFrmwrk$, that is, the particular segmenting envisagement which is refered to by each of the other envisagements ($normEnvis$, $valuatEnvis$, and $qualifEnvis$) of $ideoFrmwrk$.

Given any social actor $sa \in \mathbf{SocAct}$ of an agent society, the *ideology* that sa adopts, at any time t, is given by the *set of ideological frameworks* it is adopting at that time, denoted by $Ideo^t_{sa}$.

For any set of social actors $SocAct \in \wp(\mathbf{SocAct})$, the *ideology* of $SocAct$, at the time t, is given by $Ideo^t_{SocAct} = \sqcap\{Ideo^t_{sa} \mid sa \in SocAct\}$.[5]

4. The Tupinambá: A Society of Tribal Groups of Pre-Colonial Brazil

4.1. The Tupinambá Society

The *Tupinambá* was an indigenous tribe, of the linguistic group *Tupi*, that lived near the Brazilian Atlantic shore, between the present states of Rio de Janeiro and Bahia, before the beginning of the Portuguese occupation and colonization of the country, in the 16th century.[6]

The account of the *Tupinambá* society taken here as a reference is that in *The Social Organization of the Tupinambá* [Fernandes 1945], the classical study by the doyen of Brazilian sociology, Florestan Fernandes.

Of that detailed study, plenty of qualitative and quantitive information about the *Tupinambá* society, we use here a meager amount of information, which can only give a rough picture of the complexity of the organization of that society, but which seems to be enough to allow for the illustration of the modeling possibilities of the *societal approach*.

Fernandes' account of the organization of the *Tupinambá* society is divided into five main chapters:

1. The geographical distribution of the *Tupinambá*
2. The organization of the local groups (including their economic system and system of social relations)
3. The kinship system
4. The age categories
5. The council of chiefs

We will make use of the parts dealing with the *local groups* and the *council of chiefs*. After a brief summary of Fernandes' account of the main features of those two parts, we give the TinySML and TinyIML representation of those features. We use *concrete syntaxes* for those languages, for which we give no formal definition here, since they seem well intutive to read.

Some fragments of the societial model of the *Tupinambá* society that can be built on the basis of the information given below are presented in Figs. 2 and 3.

[5]The intersection operator ($\sqcap$) is taken to operate in a component-wise way on the ideological envisagements of the various ideological frameworks of the individual ideologies $Ideo^t_{sa}$, on the basis of the given reference segment.

[6]The occupation and subsequent colonization proceeded essentially through a civil and military war against those tribal groups, which led, by the middle of the 17th century, to the virtual extinction of their organized society, and either the dispersion the defeated Tupinambá groups in the hinterland or their placement as peripheral populations around the villages being founded by the Portuguese settlers.

As will appear, the most general modeling statement about the *Tupinambá* society is that it constitutes an *Inter-Societal System*.

4.2. The Local Groups

4.2.1. Organization

The *Tupinambá* tribe was constituted by a large number of *Local groups*, which were the fundamental integrative elements of the tribe.

Local groups were groups of *families* that lived in local arrangements of collective houses (*Malocas*), each inhabited by several families. *Families* were polygamous, formed by a man and several women.

The *Malocas* were very large constructions, made of palms and straw, inhabited by a considerable number of families[7].

The space within the *Malocas* was parceled among the married men, each men having a separate space for each of his wifes. But there were no walls marking those spaces, so that a *Maloca* was a wide, open collective space, fully visibly accessible to all of its inhabitants.

The *Malocas* of a *Local Group* were placed side by side, next to each other, around a large *Central Square*.

4.2.2. Economic System

The *Local Groups* were self-suficient economic units within their particular territories, with no relevant economic exchanges among them. That is, economic activities (gathering, hunting, cultivation) concerned just the immediate needs of each *Local Group*'s population. Craftmanshipt, specially woodcraft and ceramic, were of practical importance, regarding the manufacturation of tools. The *Local Groups* were located preferably in areas that could guarantee access to: water to drink, wood for making fire, fertile land for crops, rivers plenty of fishes.

The populations of the *Malocas* had practical limits in their number, given the size of the houses. Thus, the growth of the population of a *Maloca* naturally lead to its split into two or more new *Malocas*.

The division of labor in the *Local Groups* was done bascially between men and women: women were responsible mainly for housekeeping and the production of potery and other means for that activity, for cultivating and harvesting crops, and for collecting fruits and roots; men, mainly for preparing the land for cultivation, for hunting, fishing, and warfare, and for the production of tools for those activities. Women were also responsible for the transportation of all kind of objects, inside and outside the *Local Group*, in peacetime and in wartime.

In general, families were free to collect, fish, and hunt, in any area of the territory of the *Local Group*. Only the cultivation of land was submitted to regulation, assinging a particular cultivation terrain to each family.

[7]Fernandes mentions that the *Malocas* had sizes usually around 8-10 meters by 80-100 meters and 4-6 meters high, each inhabited by a usual number of 500-600 people, belonging to around 50-70 families [Fernandes 1945, p.68-69].

4.2.3. Social Relation System

Four levels of *social relations* characterized the social life of the Tupinambá: relations among members of a *Maloca*, relations internal to the *Local Groups*, relations among *Local Groups* (i.e., relations internal to the *Tribe*), and relations among the *Tupinambá* and other tribes.[8]

Kinship relations were the main relations among the inhabitants of a *Maloca*, making it the basis of the social life of the *Local Group*.

Solidarity among the members of a *Maloca* costumarily extended to the members of the *Local Group* and to the members of other *Local Groups*, constituting a strong moral bond among the whole *Tupinambá* tribe, regarding both peace and war issues. Adornments made of bird plumes, were of importance in this connection, both as symbols of status for those that happen to own them, and as symbolic values in the exchanges the *Local Groups* performed to maintain social solidarity both internally, among the members of the group, and externally, with other *Local Groups*.

When a *Maloca* was to be splitted, the man who succeeded in gathering a minimum number of people for the new *Maloca* becomes its *Chief*.

In time of war, local groups that were neighbor to each other formed alliances to face enemies coming from distant places, or to attack them. A system of messengers, running among the *Local Groups*, guaranteed the prompt realization of that alliance.

Winner groups gathered in feasts with anthropophagic rituals, where some of the defeated enemies were eaten by all members of the groups[9]. Often, some other members of the defeated groups were hold captive, to serve as slaves, before being sacrified.

War was, in fact, the main reason for the contact between neighbor local groups which, otherwise, remained quite isolated from each other. Blood revenge, culminating in the ritual anthropophagism of the defeated contender - reassuring the unity and self-steem of the victorious *Local Group* - was the usual war goal, much more frequent than plundering or expansion of territories.

The main collective activities of the group (feasts, ritual sacrifices, meetings of the council of chiefs etc.) occurred in the central square formed by the circle of Malocas.

4.3. The Council of Chiefs

The *Council of Chiefs*, each particular chief from a particular *Maloca*, was the core of the *political system* of a *Local Group*. It was responsible for the authoritative regulation of the interaction between the people of different *Malocas* of the *Local Group* and for decisions about problems that affected the *Local Group* as a whole, including its relationships with other *Local Groups*. The meetings of the council were open to all members of the *Local Group*.

In particular, the *Council of the Chiefs* was responsible for establishing and regulating: the war initiatives; the punishment of offenses between individuals, and between groups of individuals; and the domination of the elders over the youngsters.

[8]Notice the while the *Economic System* was a societal system of the *Local Group*, the *Social Relation System* is a societal system of the whole *Tupinambá* tribe.

[9]The classical reports of those cannibal rituals, which caused great impact in Europe when published in 1557, are those by Hans Staden [Staden 1557].

Additional problems dealt with were: issues arosen in the daily interaction between the members, reports from visiting members of other *Local Groups*, the change of the geographical location of the *Local Group*, articulation of war actions, of attack or defense, with allied *Local Groups* etc.

An important particular chief was the *Pajé*, the religious chief of the tribe. He circulated among the various *Local Groups*, visiting all the *Malocas* in each one, bringing them religious songs. Also, the *Pajé* usually had some type of imunity, that allowed him to also visit *Local Groups* of other tribes, besides the *Tupinambá*, including traditional enemy tribes.

As mentioned above, Figs. 2 and 3 show some fragments of a *societal model* of the *Tupinambá* society that can be built with the information given in the present section.

5. Final Remarks

It should be clear that there is an important drawback to the *societal approach* to MAS, which is of a pragmatical kind: the work has to be based on *general sociological theories*, which are more *complex*, less *complete*, less *consistent*, and more prone to *ideological disputes* than the usual *organizational theories*.

However, it seems that societal approaches that are general enough, such as that presented in, e.g., Jonathan Turner's *Theoretical Principles of Sociology* [Turner 2010], may satisfy the requirement of conceptual transparence and deducibility required from any approach to systems that, like MAS, should in principle be formally specifiable.[10]

Also, the concept of *agent society* is an effective conceptual solution to the problem of the link between the *micro* and the *macro* levels of (full-fledged) MAS, under the proviso, however, that the technical terms *micro* and *macro* refer only to architectural sub-levels of the *organizational structure* of the agent societies. That is, that the term *micro* does not refer to the *agents* of the *populational structure*, as it happens in the usual way of using the expression *micro-macro link*, but to the *organizational roles* performed by those agents.[11]

[10] In particular, its three volumes (1: *Macrodynamics*, 2: *Microdynamics*, and 3: *Mesodynamics*) seem to meet well the basic structure of the *societal architecture* adopted in the present paper.

[11] But, notice that the *implementation relation* between *agents* and *organizational roles* links the organizational roles with the agents, thus indirectly introducing the agents into the micro-level of the architecture.

```
InterSocSys Tupinambá:
    AgSocs = Set(LocalGroup) ∪ Set(LocGrpOtherTribes)
    IdeoFrmwrks = {IntraLocGrpRels, InterLocGrpRels, InterTribesRels}
---
AgSoc LocalGroup:
    Pop ⊆ Tupi
    SocCats = {Man, Woman, Adult, Child}
    SocSys = {HousingSys, EconSys, CouncChiefs}
---
SocSys HousingSys:
    OrgUnits = Set(Maloca)

SocSys EconSys:
    OrgUnits = Set(Family)

SocSys CouncChiefs:
    OrgUnits = {Council}
---
OrgUnit Maloca:
    OrgUnits = Set(Family)
    OrgRoles = {MalocaChief}

OrgUnit Family:
    OrgRoles = {Father, Mother, Son, Daughter}
    OrgRoleRelations = {maybe(Man and Adult, MalocaChief)}

OrgUnit Council:
    OrgRoles = {Member, Pajé, Spectator}
    OrgRoleRelations = {Member isa MalocaChief,
                        Pajé isa Member,
                        ¬(Spectator isa Member)}
---
MatEnv HousingSys:
    Objects = Set(MalocaHouse) ∪ {Square}
    ObjRelations = encircled(Square, Set(MalocaHouse))

MatEnv EconSys:
    Objects = {CultivationArea, FishingArea, HuntingArea,
               GatheringArea, WaterSources, CeramicObjs, WoodcraftObjs}

MatEnv Maloca:
    Objects = {MalocaBuilding} ∪ Set(FamilyHomeSpace) ∪ {FamilyCultivationArea}
    ObjRelations = {Set(FamilyHomeSpace) isa partition(MalocaBuilding)}
---
IdeoFrmwrk LocalGroup:
    SegmEnvisagement:
        Man, Woman, Adult, Child ⊆ Pop
            Man ∩ Woman = ∅
            Adult ∩ Child = ∅
    NormEnvisagement:
        OrgRoleConducts = {becomechief, marry}
            permitted(becomechief, Adult and Man)
            prohibited(becomechief, Woman or Child)
            permitted(marry, Man and Set(Woman))
            prohibited(marry, Woman and Set(Man))
    QualifEnvisagement:
        Woman ≤{landprep,hunting,fishing,toolsprod,warfare} Man
        Man ≤{housekeeping,cultivation,gathering,craftsmanship,transportation} Woman

IdeoFrmwrk Maloca:
    SegmEnvisagement:
        Resident, Family ⊆ Pop
            Family ⊆ Set(Resident)
    NormEnvisagement:
        OrgRoleConducts = {cultivate(FamilyCultivationArea), split(Maloca)}
            if ¬belongs(FamilyCultivationArea, Family):
                prohibited(Family, cultivate(FamilyCultivationArea))
            if size(Maloca) > limit:
                obligated(Residents, split(Maloca))
```

Figure 2. Fragments of a description of the Tupinambá tribe (Part 1).

```
IdeoFrmwrk CouncChiefs:
    SegmEnvisagement:
        Resident, Spectator, Member ⊆ Pop
        Spectator, Member ⊆ Resident
        Spectator ∩ Member = ∅
        Chief, Pajé ∈ Member
    NormEnvisagement:
        OrgRoleConducts = {regulate, conflict, changelocaction, waraction}
            if conflict among Set(Resident))):
                permitted(Council, regulate(conflict))
            if conflict with other(LocalGroup):
                permitted(Council, regulate(conflict))
            if changelocation neededby LocalGroup:
                permitted(Council, regulate(changelocation))
            if waraction neededby LocalGroup:
                permitted(Council, regulate(waraction))
        OrgRoleConducts = {visit(LocalGroup)}
            if visit(LocalGroup) requestedby Pajé of otherLocalGroup:
                obligated(Council, allow(visit))
        OrgRoleConducts = {watch(assembled(Council)}
            permitted(Member, watch(assembled(Council))
---
IdeoFrmwrk IntraLocGrpRels:
    SegmEnvisagement:
        pop(LocGrp) ⊆ Pop
            sameLocGrp(x,y,LocGrp) ⇔ x ∈ pop(LocGrp) and y ∈ pop(LocGrp)
    ValEnvisagement:
        OrgRoleConducts = {besolidarywith}
            if sameLocGrp(x,y,LocGrp):
                ¬besolidarywith(x,y) < besolidarywith(x,y)

IdeoFrmwrk InterLocGrpRels:
    SegmEnvisagement:
        Messenger, OtherPeople, pop(LocGrp) ⊆ Pop
            Messenger ∩ OtherPeople = ∅
            differentLocGrp(x,y) ⇔
                    x ∈ pop(LocGrp1) and y ∈ pop(LocGrp2) and LocGrp1 ≠ LocGrp2
    ValEnvisagement:
        OrgRoleConducts = {besolidarywith}
            if differentLocGrp(x,y):
                ¬besolidarywith(x,y) < besolidarywith(x,y)
    QualEnvisagement:
        OrgRoleConducts = {carrymessage}
            OtherPeople ≤carrymessage Messenger
    NormEnvisagement:
        OrgRoleConducts =
                {carrymessage, bloodoffend, carrywaragainst, defeat, sacrifice}
            if LocGrp1 ≠ LocGrp2 and x ∈ pop(LocGrp1) and
                    y ∈ pop(LocGrp2) and bloodoffend(x, y):
                obligated(LocGrp2, carrywaragainst(LocGrp1))
            if wartime and necessary(carrymessage(msg)):
                obligated(Messenger, carrymessage(msg))
            if wartime and LocGrp1 ≠ LocGrp2 and defeat(LocGrp1, LocGrp2) and
                    x ∈ pop(LocGrp2):
                            permitted(LocGrp1, sacrifice(x))

IdeoFrmwrk InterTribeRels:
    SegmEnvisagement:
        pop(Tribe) ⊆ HumanRace
            memberDifferentTribe(x,y) ⇔
                    x ∈ pop(Tribe1) and y ∈ pop(Tribe2) and Tribe1 ≠ Tribe2
    ValEnvisagement:
        OrgRoleConducts = {solidarywith}
            if memberDifferentTribe(x,y):
                ¬solidarywith(x,y) < solidarywith(x,y)
```

Figure 3. Fragments of a description of the Tupinambá tribe (Part 2).

References

Costa, A. C. R. (2015). Situated ideological systems: A core formal concept, some computational notation, some applications. *Axiomathes*, 27(February):15–78.

Costa, A. C. R. (2016). Moral systems of agent societies: Some elements for their analysis and design. In *Workshop on Ethical Issues in the Design of Intelligent Agents - EDIA@ECAI 2016*, page Paper 10. EURAI/University of Delft.

Costa, A. C. R. (2017). Ecosystems as agent societies, landscapes as multi-societal agent systems. In Adamatti, D. F., editor, *Multiagent Based Simulations Applied to Biological and Environmental Systems*, pages 25–43. IGI Global, Hershey.

Costa, A. C. R. (2020). Elements for the agent-based modeling of slavery systems. *Advances in Distributed Computing and Artificial Intelligence Journal*, 9:15–27.

Costa, A. C. R. (2021). Racism in agent societies: A model based on the concept of capability- based social control mechanism. In *WESAAC 2021*. CEFET/RJ. Available online at https://wesaac.ufsc.br/2021.

Costa, A. C. R. (2022a). Breve nota sobre os limites da abordagem organizacional aos sistemas multiagentes. In *Anais do XVI Workshop-Escola de Sistemas da Agentes, seus Ambientes e Aplicações*, Florianópolis. UFSC. Disponível em https://wesaac2022.ufsc.br/2022/proceedings.

Costa, A. C. R. (2022b). A short note on the bounds of the organizational approach to MAS. In *EMAS@AAMAS 2022 Informal Proceedings*, Richland. IFAMAS. Available at https://emas.in.tu-clausthal.de/2022/.

Costa, A. C. R. (2022c). A type system and a modeling language for agent societies (draft). Technical report, PPGFil-PUCRS, Porto Alegre. Available online at https://www.researchgate.net/profile/Antonio-Carlos-Rocha-Costa/publications.

Easton, D. (1965). *A Framework for Political Analysis*. Prentice-Hall, Englewood Cliffs.

Fernandes, F. (1963 [1945]). *Organização Social dos Tupinambá*. Difusão Européis do Livro, São Paulo.

Staden, H. (2011 [1557]). *Hans Staden's True History: An Account of Cannibal Captivity in Brazil*. Duke University Press.

Turner, J. H. (2010). *Theoretical Principles of Sociology (Vols. 1-3)*. Springer.

Acknowledgment

The author thanks the useful comments and questions raised by the reviewer that took this paper seriously.

Uma Proposta de Emulador de Portas Seriais para Sistemas Multiagentes Embarcados

Bruno Policarpo Toledo Freitas, Nilson Mori Lazarin, Carlos Eduardo Pantoja

[1]Centro Federal de Educação Tecnológicas Celso Suckow da Fonseca (Cefet/RJ)
Rio de Janeiro, RJ – Brazil

{bruno.freitas,nilson.lazarin,carlos.pantoja}@cefet-rj.br

Abstract. *Embedded Multi-Agent Systems (MAS) allows cognitive agents to act and perceive the physical world. However, to make it possible to evaluate the behavior of these agents is mandatory to implement an abstraction within the MAS or to build physical prototypes. This work proposes using serial communication emulation to verify the behavior of embedded agents, allowing the creation of decoupled verifiers from the MAS. A physical prototype model of an unmanned autonomous vehicle, represented in a simulated exogenous environment, was used to validate the proposed approach.*

Resumo. *Sistemas Multiagentes (SMA) Embarcados permitem a atuação de agentes cognitivos no mundo físico. Entretanto, para possibilitar a avaliação do comportamento desses agentes é obrigatória a implementação de uma abstração dentro do SMA ou a construção de protótipos físicos. Este trabalho propõe o uso de emulação de comunicação serial para a avaliação do comportamento de agentes embarcados, possibilitando dessa forma a criação de verificadores desacoplados do SMA. Para validação da abordagem proposta foi utilizado um modelo protótipo físico de veículo autônomo não tripulado, representado em um ambiente exógeno simulado.*

1. Introdução

Um conjunto de agentes autônomos capazes de perceber e atuar para alcançar algum conjunto de objetivos ou executar algum conjunto de tarefas, seja em um ambiente físico ou virtual, é denominado Sistemas Multiagentes (SMA) [Wooldridge 2000]. Agentes se diferem de softwares convencionais por apresentarem independência, pró-atividade, colaboratividade, cognição e adaptabilidade [Hübner et al. 2004]. Conforme [Michel et al. 2009], para possibilitar o desenvolvimento de agentes cognitivos, o principal modelo utilizado é o *Belief-Desire-Intention* (BDI) [Bratman 1987], pois ele se fundamenta no entendimento do raciocínio prático humano. Dessa forma, é possível implementar um mecanismo de controle deliberativo, permitindo que eles decidam sobre os objetivos a alcançar, os planos a seguir e as ações a executar [Alvares and Sichman 1997].

Os SMA também têm se mostrado uma solução viável para a adição de uma camada de cognição a sistemas ciber-físicos [Fichera et al. 2017, K. C. and Chodorowski 2019, Karaduman et al. 2023]. Um sistema baseado em agentes cognitivos, executando em um hardware capaz de atuar e perceber o ambiente físico é considerado um SMA Embarcado [Brandão et al. 2021]. Uma maneira comum

de construir esse tipo de sistema é através da ação direta do agente sobre o hardware, via comunicação serial, sem controle remoto ou processamento externo [Lazarin et al. 2022].

A aplicação de técnicas de teste de software em SMA é uma tarefa desafiadora, dado sua característica distribuída, autônoma e deliberativa, além das questões relativas à comunicação e interoperabilidade semântica e de coordenação [Houhamdi 2011]. Verificar o comportamento do SMA é uma etapa importante no desenvolvimento de soluções baseadas em agentes. Utilizar simulação ou métodos formais pode demonstrar que o SMA se comporta corretamente de acordo com suas especificações [Fortino et al. 2005]. Entretanto, no caso de SMA Embarcados, que são desenvolvidos utilizando uma arquitetura de quatro camadas [Pantoja et al. 2016], a camada de raciocínio é dependente da camada de hardware. Ou seja, envolve o desenvolvimento de um sistema computacional, o projeto e implementação do esquemático elétrico e mecânico, para então verificar a solução no mundo real, gerando dessa forma um aumento no custo e o tempo no desenvolvimento em cenário educacionais, por exemplo.

Supondo a necessidade de avaliação do comportamento de um SMA baseado em agentes BDI a ser embarcado em um veículo autônomo, é possível utilizar uma abordagem de alto nível [Alves et al. 2020], em tempo de execução, que fornece uma representação das ações dos agentes em um simulador conectado via protocolo UDP. Entretanto, neste caso, é necessário descaracterizar o próprio SMA Embarcado para possibilitar a avaliação, uma vez que a percepção e atuação nesses sistemas se dá através da comunicação serial [Lazarin and Pantoja 2015, Guinelli and Pantoja 2016]. Além disso, a abordagem de alto nível não dispensa a necessidade de um verificador formal [Dennis et al. 2012], a ser utilizado em tempo de design. Dessa forma, podemos definir o problema como a obrigatoriedade de existência de hardware para a avaliação do comportamento de SMA Embarcados.

O objetivo deste trabalho é propor o uso de emulador de interface serial para integrar ambientes exógenos simulados ao SMA, possibilitando a construção de simuladores que permitam a avaliação do comportamento do SMA a serem embarcados, sem a necessidade de hardware físico. A solução aqui apresentada explora características da arquitetura de *drivers* de dispositivos seriais para Linux, kernel comumente utilizado em SMA Embarcados. Nele, todo dispositivo serial é um dispositivo genérico do tipo *TeleTYpewriter* (TTY) [Linux Kernel Organization, Inc. 2023]. Assim, o SMA pode se conectar a uma interface serial que atua sobre um ambiente exógeno simulado ao invés do ambiente físico do mundo real. Para demonstrar a viabilidade da proposta, é apresentado um estudo de caso de desenvolvimento e validação do comportamento do SMA em um protótipo de veículo autônomo. Este trabalho contribui para a evolução do processo de desenvolvimento, através da possibilidade de criação de ambientes de homologação e simuladores desacoplados para SMA Embarcados.

Este trabalho está organizado da seguinte forma: a Seção 2 apresenta os principais conceitos necessários ao entendimento desta proposta. A Seção 3 apresenta e discute trabalhos relacionados. Na Seção 4 é apresentada a proposta de emulação de interfaces seriais para SMA Embarcados. A Seção 5 apresenta uma prova de conceito da abordagem proposta. Finalmente, uma discussão e possíveis trabalhos futuros é apresentada na Seção 6.

2. Fundamentação Teórica

Os modelos de ação e percepção dos agentes foram concebidos para trabalhar com ambientes físicos ou computacionais, denominados *ambientes exógenos*, fonte das percepções e alvo das ações dos agentes, desacoplados do projeto do SMA e considerados o contexto externo. Algumas abordagens consideram o uso de *ambientes endógenos*, uma dimensão ortogonal em relação à dimensão dos agentes que precisa ser projetada e programada pelo desenvolvedor do SMA como uma abstração para encapsular funcionalidades para permitir a percepção e atuação dos agentes [Ricci et al. 2012].

Em se tratando de SMA Embarcados, devido à baixa capacidade computacional de algumas placas computacionais (*single-board computers*), é desejável evitar o uso de camadas de abstrações que possam impactar no desempenho do sistema. A arquitetura ARGO [Pantoja et al. 2016], uma arquitetura customizada de agentes Jason [Bordini et al. 2007] é utilizada para possibilitar que agentes especializados atuem diretamente no ambiente exógeno (mundo real), através de sensores e atuadores integrados a microcontroladores, sem a necessidade de abstração no ambiente endógeno do SMA.

Na Figura 1 é apresentada a arquitetura de um SMA Embarcado onde cada agente ARGO utiliza um canal serial único, bi-direcional e exclusivo com um microcontrolador. Por este canal ele envia comandos de atuação no ambiente exógeno, via ações internas específicas de sua arquitetura. A cada ciclo de raciocínio, caso desejado, as percepções do ambiente exógeno (dados dos sensores) são atualizadas diretamente na base de crenças do agente. A comunicação serial desempenha um papel importante em SMA Embarcados onde sensores e atuadores estão fisicamente conectados ao sistema computacional do SMA. Neste trabalho, portas seriais emuladas são utilizadas para reduzir a dependência da existência de hardware físicos ao desenvolver e testar tais sistemas.

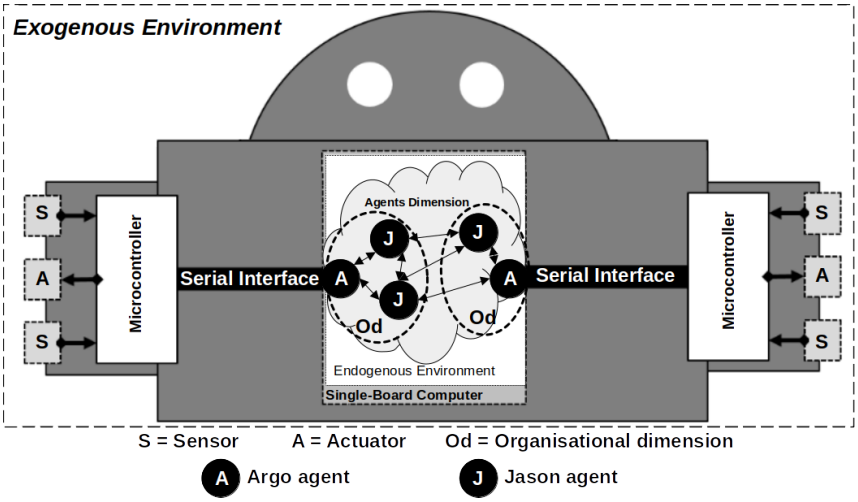


Figura 1. Arquitetura de SMAs embarcados

2.1. Arquitetura de Dispositivos Seriais

Um sistema operacional abstrai os recursos do computador para os processos em execução do sistema. Dessa maneira, cada processo enxerga esses recursos como se fossem exclusivamente para si, enquanto, na verdade, o sistema operacional organiza a partilha desses

recursos. Os processos de usuário solicitam esses recursos por meio de uma interface padronizada. O conjunto de procedimentos dessa interface forma o conjunto de *chamadas de sistema* do sistema operacional [Tanenbaum 2016].

No sistema operacional GNU/Linux, por exemplo, para criar um processo de usuário utiliza-se a chamada *fork*, enquanto a manipulação de arquivos é feita por meio das chamadas *open*, *close*, *read*, *write* e *seek*. Em especial, as chamadas de sistema responsáveis por manipular arquivos também servem como interfaces genéricas para manipulação de dispositivos por processos no espaço de usuário [Tanenbaum 2016]. Na prática, porém, existem classes de dispositivos dentro do kernel e cada classe possui APIs internas para facilitar o desenvolvimento dos drivers desses dispositivos. O kernel então mapeia as operações de leitura ou escrita solicitadas para chamadas das interfaces da classe a qual o dispositivo pertence. Um exemplo dessa propriedade pode ser visto no Código 1, em que são escritos os caracteres XYZ no arquivo */dev/ttyEmulatedPort0*, que na verdade aponta para um dispositivo serial.

Código 1. Exemplo de chamada de sistema *write* em um dispositico serial.

```c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
int main(){
    int fd = open("/dev/ttyEP0", O_WRONLY );
    int ret = write( fd, "XYZ\n" , 4 );
    close(fd);
}
```

No caso dos dispositivos seriais, além da API para os dispositivos reais propriamente ditos, tal interface ainda é subordinada à camada TTY. Inicialmente o TTY gerenciava os terminais de comando dos computadores UNIX (décadas de 70 e 80). Atualmente gerencia todas as classes de dispositivos seriais [Linux Kernel Organization, Inc. 2023].

Na Figura 2 é apresentada a arquitetura dos drivers de dispositivos em sistemas operacionais GNU/Linux. Pode-se observar que, devido ao fato dos dispositivos seriais serem subordinados a camada TTY, torna-se possível a criação de pseudo interfaces seriais. Assim, um agente poderia lidar com uma interface serial legítima, entretanto, dentro do kernel é possível implementar algo diferente. Essa propriedade é explorada neste trabalho para realizar a emulação do canal de comunicação serial.

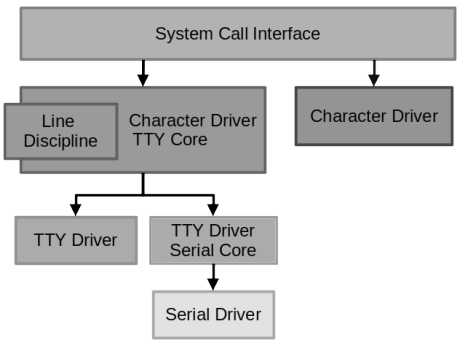


Figura 2. Arquitetura de drivers seriais no GNU/Linux [Free Electrons 2017].

3. Trabalhos relacionados

Em um estudo de caso de monitoramento da aceleração de vibração de vigas de concreto utilizando SMA e redes de sensores [Adi Putra et al. 2018] foram utilizados dois agentes: um coordenador no nó principal da rede, responsável por calcular a rota de migração para um segundo agente; e um agente móvel responsável por migrar de nó em nó da rede, realizando a coleta e agregação dos dados dos sensores. Para analisar o comportamento dos agentes, o trabalho utilizou uma abordagem mista, contendo um emulador executado em um laptop, que representou o nó principal (hospedando o agente coordenador) e cinco sensores SunSPOTs (ARM920T 180 MHz, 512 KB RAM, 2.4 GHz IEEE 802.15.4 e interface USB) reais que acionavam um LED verde quando o agente móvel chegava no sensor e realizava a agregação dos dados.

Um sistema ciber-físico é gerenciado por um agente cognitivo para apoiar pilotos de planadores durante a pilotagem, através do envio de informações, auxílio em emergências e facilitação na comunicação, monitoramento e log de eventos [Mesjasz et al. 2020]. O agente responsável pelo apoio ao piloto, se conecta a um gateway para obter informações dos sensores. Neste gateway há um SMA composto por múltiplos agentes conectores, cada um responsável pela coleta de dados de um determinado sensor e um agente coordenador e responsável por pré-processar os dados. Para analisar o comportamento do agente, o trabalho utilizou uma abordagem mista, contendo: um laptop Intel Core i5-4200U, 16 GB RAM, executando um software emulador de dados dos sensores; um single-board computer Banana Pi A20 ARM Cortex-A7 Dual-Core, 1 GB DDR3 RAM, executando o SMA do gateway; e um smartphone Samsung Galaxy S7 Edge, 4 GB RAM, executando o agente de apoio.

Um framework para programação de veículos aéreos não-tripulados (VANT) utiliza o paradigma de programação orientada a agentes, onde o SMA possui uma camada de software intermediária que utiliza uma interface UART, conectada ao firmware do VANT [Hama 2012]. Esta camada realiza a conversão das ações do agente em funções de baixo nível e a conversão dos bytes recebidos do VANT em percepções para o agente. Para avaliar a capacidade dos agentes do framework proposto na operação de um VANT, foi utilizado um simulador de aeromodelismo e um emulador de portas seriais.

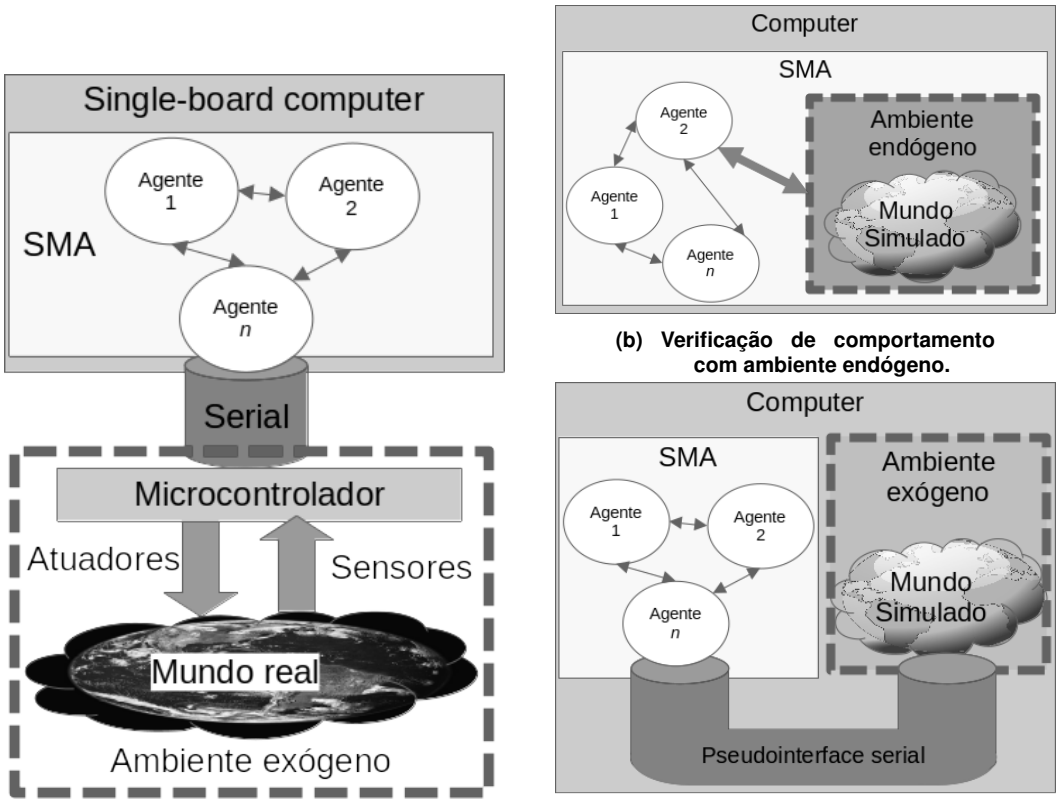
Este trabalho, diferente de [Adi Putra et al. 2018], apresenta uma abordagem que dispensa a necessidade de hardwares físicos para verificar o comportamento do agente, suas ações e percepções do ambiente. Além disso, permite a execução da simulação diretamente no computador do desenvolvedor, dispensando a comunicação externa via rede, como exposto em [Mesjasz et al. 2020]. Por fim, dispensa também a necessidade do uso de framework específico, tal como em [Hama 2012], pois este trabalho apresenta uma solução diretamente na camada de driver do sistema operacional, permitindo a utilização de frameworks genéricos para programação orientada a agentes, tal como o Jason [Bordini et al. 2007], ChonIDE [Souza de Jesus et al. 2023] ou o JaCaMo [Boissier et al. 2013].

4. Metodologia

Atualmente, a abordagem para a avaliação do comportamento de um SMA Embarcado, conforme apresentado na Figura 3a, requer a construção de um hardware físico e a realização de experimentos no mundo real, demandando custos de produção de

protótipos e tempo de preparação de *testbed*. Por outro lado, a utilização de abordagem de verificação de alto nível [Alves et al. 2020] não é adequada para uso em SMA Embarcado, conforme apresentado na Figura 3b, visto que é necessário alterar a implementação do próprio SMA para adição de uma nova dimensão a fim de possibilitar a abstração do ambiente endógeno. Além disso, essa dimensão geraria retrabalhos sempre que houvesse mudança da lógica do SMA, o que é desnecessário dado que ele seria executado diretamente no ambiente exógeno.

Este trabalho propõe a emulação da comunicação serial entre o ambiente exógeno e o SMA, a nível de sistema operacional, conforme apresentado na Figura 3c, de forma que seja possível a criação ferramentas para a avaliação do comportamento de SMA Embarcados específicas para este domínio, sem a necessidade do uso de microcontroladores, sensores, atuadores ou qualquer alteração no SMA.



(a) Abordagem atual.

(b) Verificação de comportamento com ambiente endógeno.

(c) Abordagem proposta

Figura 3. Comparativo entre a utilização de hardware físico, a utilização de ambiente endógeno e a abordagem proposta para a avaliação de comportamento em SMA Embarcados.

Na abordagem proposta, o SMA recebe as percepções e atua normalmente, como faria no mundo real, agora sobre um mundo simulado por meio da pseudointerface serial. Percepções são obtidas através de requisições de leitura, enquanto atuações são solicitas através de escrita. Com isso, o canal de comunicação serial agora pode ser conectado a um ambiente de verificação, simulando as percepções do mundo real e dispensando a necessidade de um hardware físico. Ao receber solicitações de percepções do SMA, é o mundo simulado que responde por meio de escritas no mesmo canal. Solicitações de atuação continuam a ser realizadas no ambiente exógeno, porém em um mundo simulado.

4.1. Implementação

Para possibilitar a abordagem proposta neste trabalho, foi implementado um módulo[1] para o kernel Linux que implementa um par de pseudointerfaces seriais. De um lado é fornecida uma interface para o agente que atua no mundo real se conectar, chamada *ttyEmulatedPort*. Do outro lado é fornecida uma segunda interface para conectar a aplicação que simula o ambiente exógeno, chamada *ttyExogenous*.

Este módulo é o responsável por fazer a emulação do canal de comunicação serial, conforme apresentado na Figura 4. A entrada da porta *ttyEmulatedPort* está conectada a saída da porta *ttyExogenous* e vice-versa - ou seja, o dado escrito no dispositivo *ttyEmulatedPort* é lido no dispositivo ttyExogenous. Esta abordagem é possível pois, quando um dispositivo serial real deseja enviar um dado, ele precisa escrevê-lo em um buffer chamado *flip_buffer*. Como tanto o dispositivo *ttyEmulatedPort* quanto o dispositivo *ttyExogenous* compartilham o mesmo driver, eles podem escrever dados no buffer um do outro.

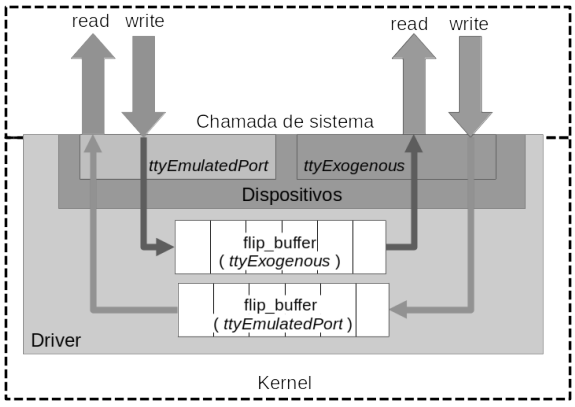


Figura 4. Implementação do canal serial emulado

As implementações das chamadas de sistema *write* dos dispositivos *ttyEmulatedPort* e *ttyExogenous* são apresentadas nos Códigos 2 e 3, respectivamente. As implementações são bastantes parecidas: a principal diferença é o flip_buffer no qual cada implementação escreve. Após inserir os dados no buffer, ao final é executado o comando *tty_flip_buffer_push*, a fim de que o kernel indique ao processo de usuário que há dados recebidos e prontos para serem lidos.

Código 2. Implementação da chamada de sistema *write* do *ttyEmulatedPort*

```
for ( i = 0; i < count; i++ ){
    tty_insert_flip_char( exogenous_port, write_buffer[i], TTY_NORMAL);
}
// Warn the TTY core to send the received characters to the user space
tty_flip_buffer_push( exogeneous_port );
```

Código 3. Implementação da chamada de sistema *write* do *ttyExogenous*

```
for ( i = 0; i < count; i++ ){
    tty_insert_flip_char( emulated_port, write_buffer[i], TTY_NORMAL);

}
tty_flip_buffer_push( emulated_port ); // Warn the TTY core to send for the user space
```

[1] https://github.com/bptfreitas/SerialPortEmulator

5. Estudo de caso

Como prova de conceito foi programado um SMA para um ChonBot [Lazarin et al. 2023] – um modelo de protótipo de veículo autônomo. A especificação do modelo, apresentada na Figura 5, conta com uma lista comandos que o agente pode enviar para o microcontrolador, tais como ativação do motor, alteração da velocidade e manipulação da buzina, faróis e lanternas. Também, é apresentada uma lista de percepções enviadas ao agente a cada ciclo de raciocínio, tais como a situação dos atuadores, a distância de obstáculos e os dados sobre luminosidade e seguidor de linha.

Actions

```
.act(goAhead  | goLeft    | goRight   | goBack  | stop ); // Motor direction
.act(buzzerOn | buzzerOnH | buzzerOnL | buzzerOff);       // Buzzer
.act(lightOn  | lightOnH  | lightOnL  | lightOff);        // HeadLight
.act(alertOn  | flashROn  | flashLOn  | flashLightOff);   // FlashLight
.act(speedH   | speedM    | speedL);                      // Motor speed
.act(breakLOn | breakLOff);                               // BreakLight
```

Perceptions

```
+motor(stopped  | running | turningRight  | turningLeft | backward).  // Motor Status
+flashLight(off | right   | left          | alert).                   // Flashlights LED
+light( off     | on      | high          | low).                     // HeadLight LED
+buzzer( off    | on      | high          | low).                     // Buzzer
+speed(default  | high    | low).                                     // Motor Speed
+breakL( off    | on).                                               // BreakLight LED
+luminosity(N).                                                      // LDR sensor
+distance(N).                                                        // Ultrasonic sensor
+lineL(N).                                                           // Line-following sensors
+lineR(N).
```

Figura 5. 2WD Chon Basic Prototype: Especificações da camada de raciocínio.

Para operar o protótipo, foi programado um agente Argo, apresentado no Código 4. Ele possui uma crença inicial, sobre o endereço do dispositivo serial que deve gerenciar e um objetivo inicial de conectar-se à interface serial, definir o tempo do ciclo de percepção (1 segundo) e habilitar o recebimento das percepções do ambiente exógeno. Ao receber as percepções do ambiente ele define a estratégia a seguir. Caso a distância seja maior ou igual a cinquenta centímetros ele envia um comando de atuação para o microcontrolador acionar o motor (*.act(goAhead)*). Caso contrário, ele envia outro comando, para virar à esquerda (*.act(turnLeft)*).

Código 4. Programação do agente responsável por operar o protótipo.

```
1  /* Initial belief */
2  serialPort(ttyUSB0).
3
4  /* Initial goals */
5  !start.
6
7  /* Plans */
8  +!start : serialPort(SP) <- .port(SP); .limit(1000); .percepts(open).
9
10 +distance(D): D >= 50 & not running <- -~running; +running; .act(goAhead).
11 +distance(D): D < 50 & not ~running <- -running; +~running; .act(turnLeft).
```

O ambiente de testes da prova de conceito foi um computador desktop com processador Celeron E3300 2.50GHz (2 cores) e 2GB de memória RAM (DDR2) executando o

Debian GNU/Linux 11 (bullseye) com ambiente gráfico LXDE (*Lightweight X11 Desktop Environment*). Para a execução SMA foi utilizada a ChonIDE (*Cognitive Hardware on Network - Integrated Development Environment*) [Souza de Jesus et al. 2023]. O módulo de kernel que implementa a proposta, para o Debian (e derivados) está disponível para download[2]. Na Figura 6a é apresentada a comunicação direta entre os pares da pseudointerface.

Para verificar o comportamento do agente Argo, sem a necessidade de construir um hardware físico, foi desenvolvida uma aplicação[3] emuladora. Nesta, são aceitos os mesmos comandos de atuação – realizando a alteração na tela – e também são enviadas as mesmas percepções que o protótipo. Nessa aplicação, o próprio desenvolvedor altera os valores das percepções diretamente nos componentes da interface gráfica da aplicação. Dessa forma, o desenvolvedor pode analisar a resposta do SMA frente as alterações do ambiente.

A aplicação executada no ambiente de testes conectou-se do lado ttyExogenous da pseudointerface instalada. A única alteração necessária, foi mudar a crença do agente Argo sobre o endereço do dispositivo serial para *serialPort(ttyVB0)*. Na Figura 6b é apresentada a avaliação do comportamento do agente, sem a necessidade de construção de hardware físico ou de alterações na estrutura do SMA, demonstrando a viabilidade da abordagem proposta neste trabalho.

(a) Comunicação direta utilizando a pseudo-interface implementada.

(b) Aplicação desenvolvida para emular o comportamento do protótipo.

Figura 6. Uso da comunicação serial emulada.

6. Discussão

O desenvolvimento de SMAs embarcados envolve várias etapas. Além da inteligência artificial descrita por meio de BDI, é necessário construir os sistemas embarcados propriamente ditos. Isso envolve adquirir o computador sob o qual o SMA irá ser executado, adquirir microcontroladores, projetar e implementar seus esquemáticos elétrico e mecânicos, para finalmente conectar o SMA ao mundo real. Somente após este último passo seria possível validar e testar a inteligência descrita inicialmente.

Este trabalho apresentou e demonstrou o uso de uma abordagem de emulação do canal de comunicação serial para SMA Embarcados. A emulação do canal de comunicação serial permite partir diretamente para o desenvolvimento da inteligência,

[2]https://github.com/chon-group/dpkg-virtualport-driver
[3]https://github.com/chon-group/bot2WDVirt

permitindo, por exemplo, validar e testar a solução antes da construção do sistema embarcado. Abre-se a possibilidade, também, de construir ambientes de simulação para tais sistemas. Dessa forma, uma vez validada a solução no mundo simulado, apenas é necessário trocar a porta serial do canal emulado para a porta real de comunicação com o microcontrolador.

Outro desdobramento deste trabalho é a maior facilidade no ensino de Inteligência Artificial apoiado por agentes robóticos [Lazarin et al. 2023]. Uma vez que atualmente é necessário a construção do sistema embarcado, cada aluno precisaria construir seu próprio sistema para iniciar o seu desenvolvimento, o que demanda tempo e um custo maior. Com a emulação e construção de um mundo simulado, o desenvolvimento se torna independente da implementação física.

Trabalhos futuros podem explorar a construção de interfaces genéricas de interação com o ambiente [Behrens et al. 2012], diretamente no ambiente exógeno, sem a necessidade de hardware físico, facilitando a integração de SMA com aplicações de Internet das Coisas em Simuladores Urbanos [Souza de Castro et al. 2022]. Além disso, se faz necessário uma avaliação quantitativa do custo computacional e uma avaliação qualitativa do quanto a solução facilita o processo de desenvolvimento de SMAs Embarcados. Ademais, uma análise do impacto no ciclo de raciocínio do agente se faz necessária, uma vez que um *System-On-a-Chip* (SoC) possui frequência de funcionamento na casa das dezenas ou centenas de MHz e a comunicação serial possui taxas de transferências pré-definidas variando de 9600 até 115200 bps, diferentemente da aplicação emuladora proposta neste trabalho, que é executada na casa dos GHz e não possui controle de taxa de transferência.

Referências

Adi Putra, S., Trilaksono, B., Harsoyo, A., and Kistijantoro, A. I. (2018). Multiagent system in-network processing in wireless sensor network. *International Journal on Electrical Engineering and Informatics*, 10:94–107. DOI:10.15676/ijeei.2018.10.1.7.

Alvares, L. O. and Sichman, J. S. (1997). Introdução aos sistemas multiagentes. In *XVII Congresso da SBC - Anais da Jornada de Atualização em Informática*. UnB.

Alves, G. V., Dennis, L., Fernandes, L., and Fisher, M. (2020). Reliable Decision-Making in Autonomous Vehicles. In Leitner, A., Watzenig, D., and Ibanez-Guzman, J., editors, *Validation and Verification of Automated Systems*, pages 105–117. Springer, Cham. https://doi.org/10.1007/978-3-030-14628-3_10.

Behrens, T., Hindriks, K. V., Bordini, R. H., Braubach, L., Dastani, M., Dix, J., Hübner, J. F., and Pokahr, A. (2012). An Interface for Agent-Environment Interaction. In Collier, R., Dix, J., and Novák, P., editors, *Programming Multi-Agent Systems*, volume 6599, pages 139–158. Springer, Berlin, Heidelberg. Series Title: Lecture Notes in Computer Science. doi.org/10.1007/978-3-642-28939-2_8.

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. Special section: The Programming Languages track at the 26th ACM Symposium on Applied Computing (SAC 2011) Special section on Agent-oriented Design Methods and Programming Techniques for Distributed Computing in Dynamic and Complex Environments. https://doi.org/10.1016/j.scico.2011.10.004.

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

Brandão, F. C., Lima, M. A. T., Pantoja, C. E., Zahn, J., and Viterbo, J. (2021). Engineering approaches for programming agent-based iot objects using the resource management architecture. *Sensors*, 21(23). https://doi.org/10.3390/s21238110.

Bratman, M. (1987). *Intention, Plans, and Practical Reason*. Cambridge: Cambridge, MA: Harvard University Press.

Dennis, L. A., Fisher, M., Webster, M. P., and Bordini, R. H. (2012). Model checking agent programming languages. *Autom Softw Eng*, 19(1):5–63. https://doi.org/10.1007/s10515-011-0088-x.

Fichera, L., Messina, F., Pappalardo, G., and Santoro, C. (2017). A Python framework for programming autonomous robots using a declarative approach. *Science of Computer Programming*, 139:36–55. https://doi.org/10.1016/j.scico.2017.01.003.

Fortino, G., Garro, A., and Russo, W. (2005). An integrated approach for the development and validation of multi-agent systems. *Computer Systems Science and Engineering*, 20(4):259–271. https://hdl.handle.net/20.500.11770/129123.

Free Electrons (2017). Linux serial drivers. https://bootlin.com/doc/legacy/serial-drivers/.

Guinelli, J. V. and Pantoja, C. E. (2016). A Middleware for Using PIC Microcontrollers and Jason Framework for Programming Multi-Agent Systems. In *WPCCG 2016: I Workshop de Pesquisas em Computação dos Campos Gerais*, volume 1, pages 38–41, Ponta Grossa. UTFPR. http://www.wpccg.pro.br/volume001.html.

Hama, M. T. (2012). *Uma plataforma orientada a agentes para o desenvolvimento de software em veículos aéreos não-tripulados*. Dissertação (mestrado), Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação., Porto Alegre. https://lume.ufrgs.br/handle/10183/66196.

Houhamdi, Z. (2011). Multi-agent system testing: A survey. *International Journal of Advanced Computer Science and Applications*, 2(6). http://doi.org/10.14569/IJACSA.2011.020620.

Hübner, J. F., Bordini, R. H., and Vieira, R. (2004). Introdução ao desenvolvimento de sistemas multiagentes com jason. *XII Escola de Informática da SBC*, 2:51–89.

K. C., U. and Chodorowski, J. (2019). A Case Study of Adding Proactivity in Indoor Social Robots Using Belief–Desire–Intention (BDI) Model. *Biomimetics*, 4(4). https://doi.org/10.3390/biomimetics4040074.

Karaduman, B., Tezel, B. T., and Challenger, M. (2023). Rational software agents with the BDI reasoning model for Cyber–Physical Systems. *Engineering Applications of Artificial Intelligence*, 123:106478.

Lazarin, N., Pantoja, C., and Viterbo, J. (2023). Towards a Toolkit for Teaching AI Supported by Robotic-agents: Proposal and First Impressions. In *Anais do XXXI Workshop sobre Educação em Computação*, pages 20–29, Porto Alegre, RS, Brasil. SBC. https://doi.org/10.5753/wei.2023.229753.

Lazarin, N. M., Pantoja, C., Souza de Jesus, V., Manoel, F., and Viterbo, J. (2022). Adição de Recursos em Tempo de Execução a Sistemas Multi-Agentes Embarcados. In *Anais do XVI Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações (WESAAC 2022)*, pages 73–84, Blumenau. UFSC.

Lazarin, N. M. and Pantoja, C. E. (2015). A robotic-agent platform for embedding software agents using raspberry pi and arduino boards. In *Proceedings of 9th Software Agents, Environments and Applications School (WESAAC 2015)*, pages 13–20, Niteroi. UFF.

Linux Kernel Organization, Inc. (2023). TTY — The Linux Kernel documentation. https://www.kernel.org/doc/html/latest/driver-api/tty/.

Mesjasz, M. M., Ganzha, M., and Paprzycki, M. (2020). Modeling cyber-physical systems – a GliderAgent 3.0 perspective. *J Intell Inf Syst*, 55(1):67–93. https://doi.org/10.1007/s10844-019-00588-3.

Michel, F., Ferber, J., and Drogoul, A. (2009). Multi-Agent Systems and Simulation: A Survey from the Agent Community's Perspective. In *Multi-Agent systems: Simulation and applications*. CRC Press. https://doi.org/10.1201/9781420070248-10.

Pantoja, C. E., Stabile, M. F., Lazarin, N. M., and Sichman, J. S. (2016). ARGO: An Extended Jason Architecture that Facilitates Embedded Robotic Agents Programming. In Baldoni, M., Müller, J. P., Nunes, I., and Zalila-Wenkstern, R., editors, *Engineering Multi-Agent Systems*, pages 136–155, Cham. Springer International Publishing. https://doi.org/10.1007/978-3-319-50983-9_8.

Ricci, A., Santi, A., and Piunti, M. (2012). Action and Perception in Agent Programming Languages: From Exogenous to Endogenous Environments. In Collier, R., Dix, J., and Novák, P., editors, *Programming Multi-Agent Systems*, volume 6599, pages 119–138. Springer Berlin Heidelberg, Berlin, Heidelberg. Series Title: Lecture Notes in Computer Science. https://doi.org/10.1007/978-3-642-28939-2_7.

Souza de Castro, L. F., Manoel, F. C. P. B., Souza de Jesus, V., Pantoja, C. E., Pinz Borges, A., and Vaz Alves, G. (2022). Integrating Embedded Multiagent Systems with Urban Simulation Tools and IoT Applications. *RITA*, 29(1):81–90. https://doi.org/10.22456/2175-2745.110837.

Souza de Jesus, V., Mori Lazarin, N., Pantoja, C. E., Vaz Alves, G., Ramos Alves de Lima, G., and Viterbo, J. (2023). An IDE to Support the Development of Embedded Multi-Agent Systems. In Mathieu, P., Dignum, F., Novais, P., and De la Prieta, F., editors, *Advances in Practical Applications of Agents, Multi-Agent Systems, and Cognitive Mimetics. The PAAMS Collection*, pages 346–358, Cham. Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-37616-0_29.

Tanenbaum, A. S. (2016). *Sistemas Operacionais Modernos*. Pearson, 4 edition.

Wooldridge, M. (2000). Intelligent Agents. In *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1st edition.

Coordenação de robôs ROS com Agentes BDI e $\mathcal{M}$OISE

Pedro Henrique Dias[1], **Maiquel de Brito**[2]

[1] Departamento de Automação e Sistemas
Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

[2]Departamento de Engenharia de Controle, Automação e Computação
Universidade Federal de Santa Catarina (UFSC)
Blumenau – SC – Brasil

pedrohenrique.dias8@gmail.com, maiquel.b@ufsc.br

Abstract. *ROS (Robotic Operating System) is a set of libraries, tools and standards that aim to simplify the development of robots. However, ROS does not contain features to handle the coordination of multiple autonomous robots. On the other hand, coordination of autonomous entities is a subject of research and development in the area of Multi-Agent Systems. In this context, this article proposes the integration between concepts and tools of multiagent programming and the ROS platform to enable coordination of robots. For such, it is presented a case of control of simulated robots that need to act in a coordinated way to solve different problems. The solution is experimentally evaluated, verifying to what extent the it allows both the specification of coordination between ROS robots and the effective coordinated action of these robots according to such specification.*

Resumo. *O ROS (Robotic Operating System) é um conjunto de bibliotecas, ferramentas e padrões que tem por objetivo simplificar o desenvolvimento de robôs. No entanto, o ROS não contém recursos para tratar da coordenação de múltiplos robôs autônomos. Por outro lado, coordenação de entidades autônomas é objeto de pesquisas e desenvolvimento na área de Sistemas Multiagente. Neste contexto, este artigo propõe integração entre conceitos e ferramentas de programação multiagente e da plataforma ROS para viabilizar a coordenação de robôs. Para isso, apresenta-se um caso de controle de robôs simulados que precisam atuar de forma coordenada para a solução de diferentes problemas. A solução é avaliada de forma experimental, verificando-se em que medida ela permite tanto a especificação da coordenação entre robôs ROS quanto a efetiva atuação coordenada destes robôs de acordo com tal especificação.*

1. Introdução

O desenvolvimento de robôs e sistemas robóticos é uma área em constante evolução e, como tal, é fundamental contar com ferramentas de desenvolvimento capazes de atender uma gama cada vez maior de aplicações. O *Robot Operating System* (ROS) tem sido atualmente uma das plataformas mais populares para desenvolvimento de aplicações robóticas, graças à sua ampla gama de bibliotecas, algoritmos e softwares de simulação nativos. No entanto, apesar de suas várias funcionalidades, a plataforma não possui uma solução eficiente para coordenação de múltiplos robôs em ambientes dinâmicos e complexos.

Uma possível solução para esta limitaçao é a programação multiagente (ou MAOP, do inglês *Multi-Agent Oriented Programming*). MAOP fornece abstrações e ferramentas não só (i) para desenvolver as entidades computacionais autônomas (chamadas, neste artigo, de *agentes*) que atuam em um sistema como também (ii) para regular e coordenar as atividades destes agentes [Boissier et al. 2020].[1] Considera-se, nesta hipótesse, as possibilidades de (i) utilizar *agentes* como uma metáfora para o desenvolvimento de robôs baseados em ROS e (ii) utilizar os elementos disponibilizados pelos modelos e ferramentas de MAOP para especificar e implementar a coordenação destes robôs.

Diante desse cenário, este artigo propõe a integração de agentes BDI [Rao and Georgeff 1995] com a plataforma ROS para desenvolver uma solução para coordenação de múltiplos robôs. Para isso, apresenta um caso de controle de robôs simulados, em que diferentes quantidades de robôs precisam colaborar em um sistema para atingir um objetivo comum. Os resultados são avaliados experimentalmente a partir do comportamento dos robôs no sistema desenvolvido.

No que segue, a Seção 2 introduz os conceitos essenciais para a compreensão deste trabalho; a Seção 3 descreve a implementação de mecanismos para coordenação de robôs ROS; a Seção 4 descreve os resultados obtidos; e a Seção 5 faz algumas considerações finais, mencionando trabalhos relacionados e potenciais trabalhos futuros.

2. Fundamentação teórica

ROS é uma plataforma de software livre utilizada em robótica [Koubaa 2017]. Ele fornece uma estrutura compartilhada para gerenciar hardware e funções comuns, como comunicação de dados e controle de movimento. Com ele, é possível desenvolver e executar aplicações na área de robôs em um ambiente distribuído, permitindo que eles interajam com seu ambiente de maneira eficiente e autônoma. ROS possui uma ampla comunidade de desenvolvedores que contribuem para seu desenvolvimento e aprimoramento contínuo, além de oferecer suporte para uma ampla variedade de plataformas de robótica.

Entre as várias distribuições ROS disponíveis, este trabalho utiliza a *ROS-Noetic*.[2] Em sistemas baseados nela, o *roscore* é o núcleo do sistema, responsável por criar e gerenciar a infraestrutura de informações que são passadas entre os diferentes nós do sistema. Os *nós* são processos independentes que se comunicam por meio de tópicos e serviços para realizar tarefas específicas em um sistema robótico, podendo representar diferentes partes de um mesmo robô, como sensores e atuadores, ou até mesmo unidades robóticas inteiras. Os *tópicos* permitem a troca de mensagens assíncronas entre diferentes nós do ROS, sendo usados para transmitir informações como dados de sensores, comandos de atuadores ou mensagens de controle. Os *serviços* fornecem uma interface para a chamada de funções ou métodos em nós ROS, permitindo a execução de tarefas como inicialização, configuração ou controle do robô.

Agentes BDI são agentes que têm seu funcionamento basado em crenças (ou *beliefs*), que são as informações que o agente possui sobre o sistema em que atua; desejos, que são os estados de mundo que o agente gostaria de atingir; e intenções,

[1] MAOP considera uma terceira dimensão no desenvolvimento de sistemas, que é o *ambiente*, composto pelos elementos não autônomos utilizados pelos agentes para atingir seus objetivos. Esta dimensão não é explorada neste artigo.

[2] Disponível em `https://wiki.ros.org/noetic`.

que são os estados de mundo que o agente está efetivamente comprometido a atingir [Rao and Georgeff 1995]. *Jason* é um interpretador baseado em Java para o desenvolvimento de agentes inteligentes que utiliza a arquitetura BDI através de uma versão estendida da linguagem *AgentSpeak*(L) [Bordini et al. 2007, Rao 1996]. Com ele, é possível criar agentes autônomos capazes de tomar decisões com base em seu conhecimento, crenças e objetivos, além de interagir com outros agentes em um ambiente complexo.

$\mathcal{M}$OISE é um modelo para especificação de organizações de agentes. Esta especificação é feita nas perspectivas (i) *estrutural*, que define *papeis* e os agrupa em *grupos*; (ii) *funcional*, em que *schemas* definem os *objetivos* da organização, decompondo-os em subobjetivos, os quais são agrupados em *missões*; e (iii) *normativa*, que atribui obrigações aos diferentes papeis com respeito ao cumprimento dos objetivos da organização [Hübner et al. 2007]. Em tempo de execução, agentes que atuam em um sistema que conta com organizações $\mathcal{M}$OISE adotam papeis, comprometem-se com missões e, assim, adquirem obrigações com respeito a determinados objetivos organizacionais. A organização é quem define quais são os objetivos que cada agente deve procurar cumprir, bem como a ordem em que eles devem ser cumpridos.

A integração dos agentes Jason com ROS é possível através do *framework embedded-mas*.[3] Este *framework* fornece extensões aos agentes Jason de forma que (i) suas percepções incluam valores lidos em tópicos ROS e (ii) seu repertório de ações inclua aquelas realizadas tanto através da escrita em tópicos quanto da chamada de serviços ROS.

3. Implementação de um sistema de robôs ROS coordenados

A possibilidade de coordenação de robôs ROS com a utilização de MAOP é avaliada experimentalmente neste artigo. Os experimentos tratam de um cenário simulado em que robôs devem trabalhar em conjunto para desenhar diferentes formas geométricas. Estas formas são: um quadrado, um hexágono e uma estrela de cinco pontas. Admite-se que diferentes quantidades de robôs podem atuar no sistema e essas quantidades são desconhecidas em tempo de projeto. Dessa forma, a atuação de cada robô varia de acordo com a figura a ser desenhada e com o número de robôs disponíveis para tal. Por exemplo, um quadrado pode ser desenhado por uma quantidade de robôs que varia de um a quatro. No caso de um robô, este deve desenhar todos os lados da figura. Dois robôs ou três robôs poderiam adotar diferentes estratégias para desenhar um quadrado (ex.: todos os robôs desenham um único lado e um dos robôs se encarrega de desenhar o lado restante). Para quatro robôs, uma estratégia possível é que cada um deles desenhe um lado. Já no caso de formas com mais de quatro lados, como um hexágono ou uma estrela, alguns destes 4 robôs teriam que desenhar mais de um lado. Os experimentos são detalhados na sequência. Inicialmente, descreve-se a simulação utilizada (Seção 3.1). A seguir, descreve-se a implementação dos agentes e sua integração aos robôs ROS simulados (Seção 3.2), a especificação da organização (Seção 3.3), e, por fim, os experimentos realizados (Seção 3.4).[4]

[3]Disponível em `https://github.com/embedded-mas/embedded-mas`

[4]A implementação dos experimentos está disponível em `https://github.com/embedded-mas/ros-devs/tree/main/examples/turtleCoop/Coordinated_Turtles_Final`

3.1. Descrição da simulação

A distribuição ROS-Noetic, utilizada nos experimentos, disponibiliza a simulação *Turtlesim*, que permite criar robôs simulados simples. Na simulação, os robôs são representados por tartarugas que conseguem girar em seus próprios eixos e se mover linearmente por um plano bidimensional. Para facilitar a visualização, o movimento das tartarugas pelo plano deixa um rastro colorido. Cada robô possui seu próprio conjunto de tópicos e serviços, o que lhes permite interagir com o ambiente. Neste experimento, os robôs são idênticos, e, por isso, seus conjuntos de tópicos e serviços também são idênticos. São eles:

- <*robot_id*>*/teleport-absolute*: Serviço que faz com que o robô se mova pelo plano. Recebe como argumentos X,Y e θ, que são, respectivamente, as coordernadas horizontal, vertical e angular;
- <*robot_id*>*/set-pen*: Serviço que permite que o robô altere as características do rastro que ele faz ao se movimentar. Recebe como argumentos (i) valores RGB, que definem a cor do rastro, (ii) um valor para a largura da linha e (iii) o comando de habilitar ou não o seu desenho;
- <*robot_id*>*/clear*: Serviço que dá ao robô a capacidade de limpar a tela de qualquer rastro deixado pelas tartarugas;
- <*robot_id*>*/pose*: Tópico que armazena as posições e velocidades do robô no plano da simulação.

Os tópicos e serviços de cada robô diferenciam-se uns dos outros porque seus nomes incluem o identificador do robô correspondente, referenciado neste artigo como <*robot_id*>. Por exemplo, se há robôs identificados como *turtle1* e *turtle2*, então há um serviço chamado *turtle1/teleport-absolute* e um serviço chamado *turtle2/teleport-absolute*.

3.2. Implementação de agentes e integração com ROS

Cada robô, que possui seu próprio conjunto de tópicos e serviços acionáveis, é integrado a um agente *Jason* através do *framework embedded-mas*. O código de todos os agentes é idêntico. Cada agente possui em seu repertório as seguintes ações: *move-turtle*, que faz com que o robô se movimente no ambiente; *paint*, que faz com que o robô modifique as características do rastro deixado ao movimentar-se; e *clear*, que faz com que o robô apague os rastros deixados no ambiente. Estas ações são efetivamente realizadas através de atuações habilitadas pelo hardware dos robôs (que é simulado, neste caso). Por isso, elas são integradas à ação interna *defaultEmbeddedInternalAction* (Figura 1, linhas 19, 22 e 25), que faz com que uma ação executada por um agente seja efetivamente realizada por um atuador físico ou simulado. Além do nome da ação executada pelo agente, esta ação interna recebe dois outros parâmetros, que são (i) o *roscore* em que a atuação correspondente será realizada e (ii) uma lista de parâmetros requeridos pela ação.

A porção física do agente (que neste caso, é constituída por um robô ROS simulado) é especificada em um arquivo em formato YAML [Telang 2020]. Parte da especificação usada neste experimento é ilustrada na Figura 2. Essa porção física pode ser composta por um ou mais *roscore*, que controlam os serviços e tópicos dos quais o agente obtém percepções e sobre os quais pode atuar. Neste experimento, utiliza-se apenas um *roscore* identificado como `sample_roscore`, acessível através do endereço `ws://localhost:9090` (Figura 2 – linhas 1 a 3).

```
1  +!paint_square_s1
2    <-!move(7.544445, 7.544445);
3      !paint(0, 255, 0);
4      !clear;
5      !move(3.544445, 7.544445).
6
7  +!paint_square_s2
8    <-!move(3.544445, 7.544445);
9      !paint(0, 255, 0);
10     !move(3.544445, 3.544445).
11
12 +!paint_hexagon_s1
13   <-!move(5.544445, 8.928932);
14     !paint(255, 0, 0);
15     !clear;
16     !move(4.288248, 7.672735).
17
18 +!clear
19    <- .defaultEmbeddedInternalAction("sample_roscore","clear", []).
20
21 +!move(X, Y)
22    <- .defaultEmbeddedInternalAction("sample_roscore","move_turtle", [X,Y,0]).
23
24 +!paint(X, Y, Z)
25       <- .defaultEmbeddedInternalAction("sample_roscore","paint", [X,Y,Z,2,off]).
26
```

Figura 1. Trechos do código do agente

Para cada *roscore*, define-se, no bloco `perceptionTopics`, os tópicos ROS cujos valores serão transformados em percepções do agente, especificando-se (i) o nome do tópico (através da chave `topicName`), (ii) o tipo do dado armazenado no tópico (através da chave `topicType`) e (iii) o identificador (ou *functor*) da crença que será gerada a partir da percepção (através da chave `beliefName`). Na especificação da Figura 2 – linhas 5 a 7, o tópico *<robot_id>/pose*, que armazena dados do tipo *turtlesim/Pose*, produzirá percepções e, eventualmente, crenças no formato *turtle_position(X)*, em que *X* é o valor armazenado no tópico.

As ações do agente que são habilitadas por recursos dos *roscore* são especificadas na seção `action` do arquivo YAML. Neste trabalho, todas as ações são realizadas através de serviços ROS. A conexão entre ações do agente e serviços ROS é especificada na seção `serviceRequestActions`. Para cada ação realizada através de um serviço ROS, define-se (i) o nome da ação segundo o repertório de ações do agente (através da chave `actionName`), (ii) o nome do serviço que realiza a ação (através da chave `serviceName`) e (iii) os parâmetros requeridos pelo serviço (através da chave `params`). Conforme a Figura 2 – linhas 10 a 25 – neste trabalho, o repertório do agente inclui três ações realizadas através de serviços ROS: (i) a ação *move_turtle*, realizada através de uma chamada ao serviço */<robot_id>/teleport_absolute*, incluindo os parâmetros *x*, *y* e *theta*; (ii) a ação *paint*, realizada através de uma chamada ao serviço /robot_id*/set_pen*, com os parâmetros *r*, *g*, *b*, *width* e *'off'*; e (iii) a ação *clear*, realizada através de uma chamada ao serviço *clear*, que não requer parâmetro algum.

Como o código de todos os agentes é idêntico, todos têm as mesmas capacidades para desenhar cada lado indivudual das figuras propostas pelos experimentos, possuindo planos para tal. Por exemplo, na Figura 1, linhas 1 a 16, especifica-se planos para desenhar os lados 1 e 2 do quadrado e o lado 1 do hexágono, movendo-se entre coordenadas específicas, no caso vértices da forma geométrica, e formando as linha que compõem a

```yaml
- device_id: sample_roscore
  microcontroller:
      connectionString: ws://localhost:9090
  perceptionTopics:
      - topicName: turtle1/pose
        topicType: turtlesim/Pose
        beliefName: turtle_position
  actions:
    serviceRequestActions:
        - actionName: move_turtle
          serviceName: /turtle1/teleport_absolute
          params:
            - x
            - y
            - theta
        - actionName: paint
          serviceName: /turtle1/set_pen
          params:
            - r
            - g
            - b
            - width
            - 'off'
        - actionName: clear
          serviceName: /clear
```

Figura 2. Especificação feita no arquivo yaml

figura. Por mais que os agentes tenham a capacidade de desenhar cada lado de uma forma geométrica, eles não têm, codificado em si, a capacidade de desenhar uma forma completa. Por exemplo, os agentes têm planos para desenhar os quatro lados de um quadrado mas não têm plano algum para desenhar um quadrado completo. Por outro lado, como os agentes têm capacidade para desenhar cada lado das figuras geométricas propostas, eles podem participar de organizações que os coordenem para que essas capacidades sejam utilizadas para atingir tal objetivo, conforme descrito na Seção 3.3.

3.3. Organização

Uma organização $\mathcal{M}$OISE especifica a coordenação dos robôs para que eles desenhem todas as formas geométricas em diferentes circunstâncias (que, neste caso, são as diferentes quantidades de robôs que podem atuar simultaneamente no sistema). Para tal, na especificação estrutural, definiu-se 10 papeis que compõem um grupo, conforme definido a seguir e ilustrado na Figura 3.

$$\mathcal{G} = \{painter_1, painter_2, painter_3, painter_4, painter_5, \\ painter_6, painter_7, painter_8, painter_9, painter_{10}\} \quad (1)$$

O agente que desempenha cada papel é responsável por desenhar o lado com o mesmo identificador numérico em cada uma das formas (ex.: o agente que desempenha o papel $painter_1$ é responsável por contruir o lado 1 de todas as figuras). Como as figuras têm quantidades de lados diferentes, o agente que assumir um papel irá pintar o lado designado a ele somente se ele existir. Por exemplo, o papel $painter_5$ será responsável por construir o lado 5 de todas formas, mas, como um quadrado possui apenas 4 lados, o agente que assumi-lo irá cumprir sua função somente quando a organização definir que será desenhada uma figura com mais de quatro lados (que, neste trabalho, são um hexágono e uma estrela). Seguindo essa lógica, como o maior número de lados entre as figuras é 10 (no caso da estrela), foram criados 10 papeis dentro da organização.

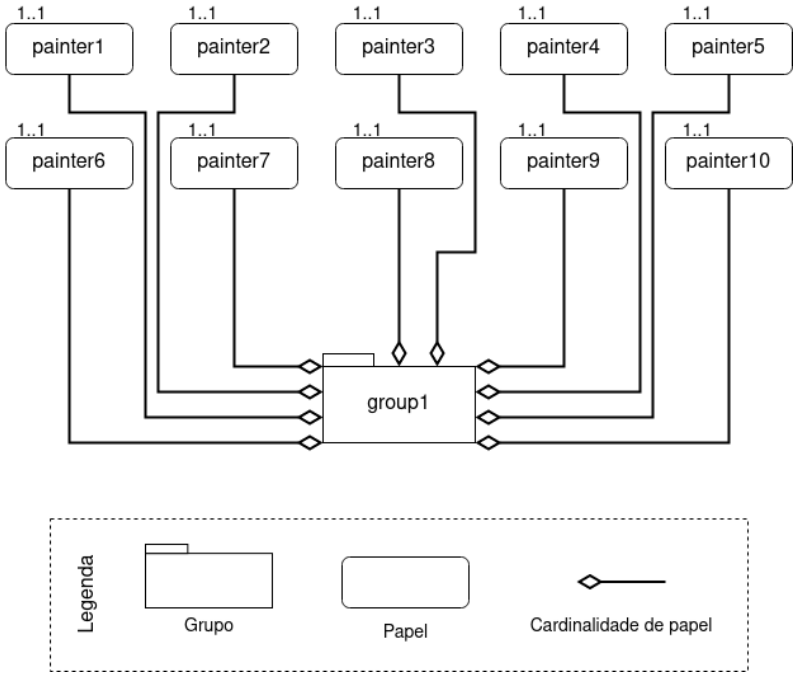


Figura 3. Especificação estrutural

Os agentes atuarão para executar o *schema* ilustrado na Figura 4. O grupo de agentes por ele responsável deve satisfazer o objetivo *paint*, que é decomposto nos subobjetivos *square*, *hexagon* e *star*, que devem ser satisfeitos nesta ordem e consistem em desenhar um quadrado, um hexágono e uma estrela de cinco pontas, respectivamente. Cada um destes subobjetivos é decomposto em novos subobjetivos correspondentes ao desenho de cada lado das formas geométricas correspondentes. Este desenho pode ser feito em paralelo sempre que possível. Por exemplo, diferentes agentes podem satisfazer simultaneamente os objetivos *paint_square_s1*, ..., *paint_square_s4* para desenhar um quadrado. Os objetivos definidos pelo *schema* são agrupados em 10 *missões*, denominadas *paint1*, $\cdots$, *paint10*. A especificação normativa da organização define o compromisso de cada papel com as missões e, consequentemente, com cada objetivo, conforme a Tabela 1.

norm	role	(mission) goals
norm1	painter1	(paint1) paint_square_s1, paint_hexagon_s1, paint_star_s1
norm2	painter2	(paint2) paint_square_s2, paint_hexagon_s2, paint_star_s2
norm3	painter3	(paint3) paint_square_s3, paint_hexagon_s3, paint_star_s3
norm4	painter4	(paint4) paint_square_s4, paint_hexagon_s4, paint_star_s4
norm5	painter5	(paint5) paint_hexagon_s5, paint_star_s5
norm6	painter6	(paint6) paint_hexagon_s6, paint_star_s6
norm7	painter7	(paint7) paint_star_s7
norm8	painter8	(paint8) paint_star_s8
norm9	painter9	(paint9) paint_star_s9
norm10	painter10	(paint10) paint_star_s10

Tabela 1. Especificação normativa

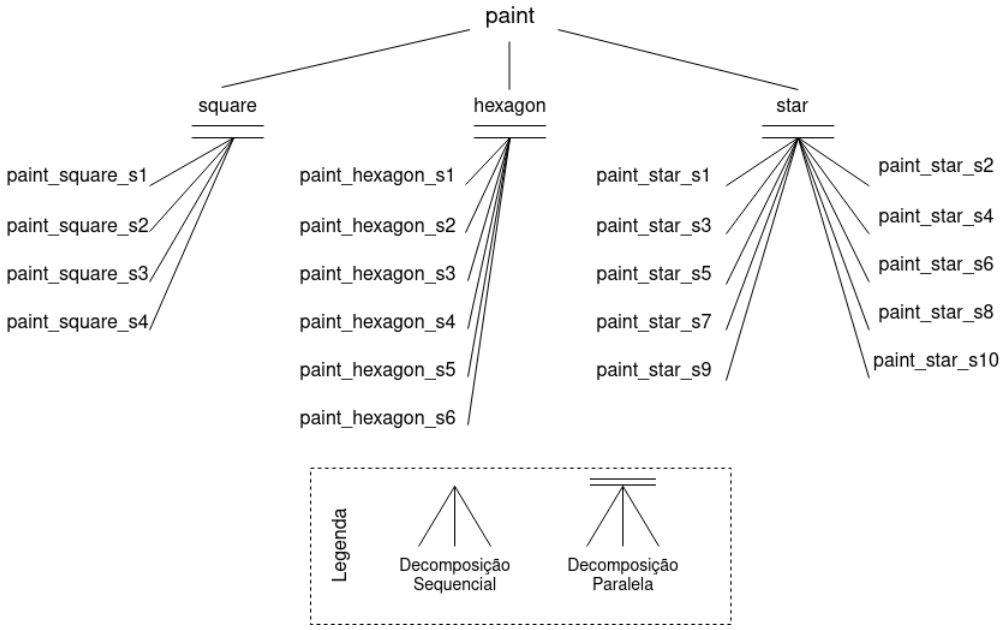


Figura 4. Especificação funcional

3.4. Experimentos

Para avaliar o sistema, foram executadas diversas simulações com diferentes quantidades de robôs em diferentes combinações de papeis. As simulações foram realizadas para verificar se a execução do sistema é bem sucedida (isto é, se todas as formas geométricas são desenhadas corretamente) com diferentes quantidades de robôs executando diferentes papeis. Além disso, as simulações permitem analisar de que forma o objetivo global do sistema é atingido quando diferentes quantidades de robôs atuam no sistema e desempenham diferentes combinações de papeis. Para isso, avalia-se também a quantidade de *passos* necessária para concluir o desenho de todas as formas. Um *passo* é completado quando todos os robôs executam a ação que lhes é atribuída pela organização. Por exemplo, quatro robôs podem desenhar um quadrado em um único passo se a organização designar cada um deles para desenhar um lado da figura. Assume-se que $\mathcal{A}$ é o conjunto de robôs que atuam no sistema, conforme definido a seguir:

$$\mathcal{A} = \{turtle_1, turtle_2, turtle_3, turtle_4, turtle_5, \\ turtle_6, turtle_7, turtle_8, turtle_9, turtle_{10}\} \tag{2}$$

Uma *distribuição* de papeis $\mathcal{D} \subseteq \mathcal{G} \times \mathcal{A}$ é uma atribuição de todos os papeis $g \in \mathcal{G}$ a alguns agentes $a \in \mathcal{A}$. As possíveis distribuições são muitas e não seria possível tratá-las todas neste artigo. Por isso, para cada possível quantidade de robôs atuando no sistema, seleciona-se três distribuições $d \in \mathcal{D}$: uma delas sendo a mais equilibrada possível (em que mais robôs conseguem fazer mais tarefas simultaneamente) e as outras duas sendo definidas de forma aleatória. A única exceção é a execução de um experimento com um único robô, em que a única distribuição possível é aquela em que tal robô assume todos os papeis. Os experimentos com diferentes quantidades de robôs e diferentes distribuições de papeis são descritos a seguir. As distribuições de papeis consideradas são definidas pelas expressões listadas na Figura 5. Elas são notadas como $\mathcal{D}_i^n$, em que n é a quantidade

n	Distribuição		
	$\mathcal{D}_1^n$	$\mathcal{D}_2^n$	$\mathcal{D}_3^n$
1	20	—	—
2	10	11	11
3	8	10	11
4	6	8	10
5	5	8	12
6	4	7	7
7	4	7	6
8	4	5	5
9	4	6	5
10	3	3	3

Tabela 2. Resultados dos experimentos

de agentes considerada na simulação e i identifica a distribuição para aquela quantidade. Por exemplo, $\mathcal{D}_3^2$ é a terceira distribuição considerada para uma simulação dois agentes. Por questões de legibilidade, os elementos dos conjuntos $\mathcal{G}$ e $\mathcal{A}$ serão descritos de forma abreviada: p_n será a abreviação de $painter_n$ e t_n será abreviação de $turtle_n$. Os resultados são exibidos na Tabela 2.

4. Resultados

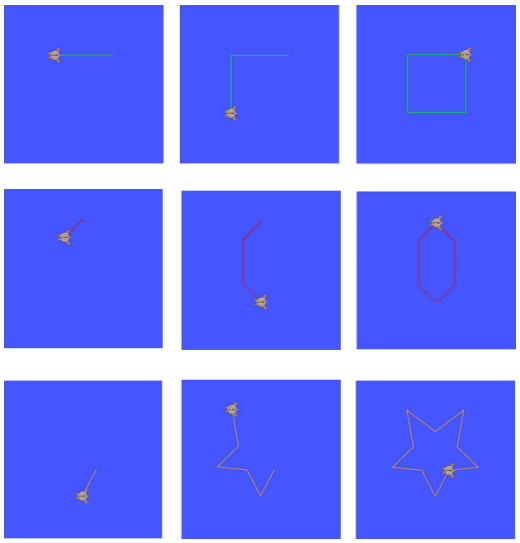
Os robôs cooperaram de forma bem sucedida ao atuar no sistema descrito na Seção 3, desenhando todas as formas geométricas em diferentes combinações de agentes e papeis. Em todas essas combinações, os robôs atuaram de forma coordenada para desenhar as formas geométricas especificadas pela organização. A Figura 6 ilustra alguns passos na execução de alguns experimentos. Embora os agentes tenham capacidade para desenhar *lados* das formas geométricas, a capacidade de desenhar as formas completas é dada pela organização. Os agentes não têm codificado em si qualquer instrução sobre como desenhar cada uma das formas, nem como atuar em cada uma das possíveis quantidades de agentes atuando no sistema.

Assim, observa-se que com a utilização de programação multiagente, com *Jason* integrados aos robôs ROS, e com organizações $\mathcal{M}$OISE, é possível coordenar a atuação de múltiplos robôs sem que essa coordenação precise ser especificada em cada um deles. Cada um dos robôs têm suas capacidades particulares, que não são suficientes para atingir os objetivos globais do sistema. A organização, por sua vez, confere essa capacidade, não aos robôs individuais, mas ao conjunto de robôs que atua no sistema.

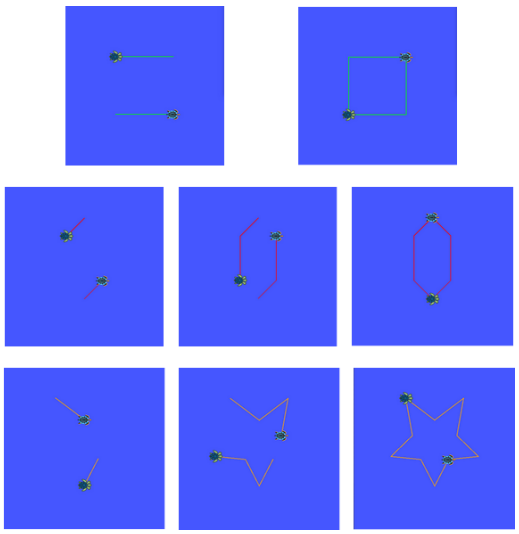
É possível afirmar, também, que a eficiência da operação é afetada pelo número de agentes disponíveis, bem como pela distribuição de papéis e pela definição desses papéis na organização. De modo geral, mais robôs produzem um resultado mais eficiente porque mais tarefas podem ser executadas simultaneamente. Essa eficiência, no entanto, está sujeita à distribuição de papeis, pois certas distribuições não favorecem a execução simultânea de tarefas (por exemplo, a distribuição D_3^n). A distribuição de papéis de forma equilibrada e bem definida pode permitir uma alocação eficiente de recursos e responsabilidades, aumentando a eficiência do sistema.

$$\mathcal{D}_1^1 = \{(t_1,p_1),(t_1,p_2),(t_1,p_3),(t_1,p_4),(t_1,p_5),(t_1,p_6),(t_1,p_7),(t_1,p_8),(t_1,p_9),(t_1,p_{10})\}$$
$$\mathcal{D}_1^2 = \{(t_1,p_1),(t_2,p_2),(t_1,p_3),(t_2,p_4),(t_1,p_5),(t_2,p_6),(t_1,p_7),(t_2,p_8),(t_1,p_9),(t_2,p_{10})\}$$
$$\mathcal{D}_2^2 = \{(t_1,p_1),(t_2,p_2),(t_1,p_3),(t_1,p_4),(t_2,p_5),(t_2,p_6),(t_1,p_7),(t_2,p_8),(t_2,p_9),(t_1,p_{10})\}$$
$$\mathcal{D}_3^2 = \{(t_2,p_1),(t_1,p_2),(t_2,p_3),(t_1,p_4),(t_2,p_5),(t_2,p_6),(t_1,p_7),(t_2,p_8),(t_1,p_9),(t_1,p_{10})\}$$
$$\mathcal{D}_1^3 = \{(t_1,p_1),(t_2,p_2),(t_3,p_3),(t_3,p_4),(t_1,p_5),(t_2,p_6),(t_2,p_7),(t_3,p_8),(t_1,p_9),(t_1,p_{10})\}$$
$$\mathcal{D}_2^3 = \{(t_1,p_1),(t_1,p_2),(t_3,p_3),(t_3,p_4),(t_3,p_5),(t_3,p_6),(t_2,p_7),(t_1,p_8),(t_1,p_9),(t_2,p_{10})\}$$
$$\mathcal{D}_3^3 = \{(t_3,p_1),(t_3,p_2),(t_3,p_3),(t_2,p_4),(t_1,p_5),(t_2,p_6),(t_3,p_7),(t_1,p_8),(t_1,p_9),(t_3,p_{10})\}$$
$$\mathcal{D}_1^4 = \{(t_1,p_1),(t_2,p_2),(t_3,p_3),(t_4,p_4),(t_4,p_5),(t_3,p_6),(t_1,p_7),(t_1,p_8),(t_2,p_9),(t_2,p_{10})\}$$
$$\mathcal{D}_2^4 = \{(t_1,p_1),(t_2,p_2),(t_3,p_3),(t_3,p_4),(t_4,p_5),(t_1,p_6),(t_3,p_7),(t_2,p_8),(t_3,p_9),(t_4,p_{10})\}$$
$$\mathcal{D}_3^4 = \{(t_1,p_1),(t_1,p_2),(t_1,p_3),(t_4,p_4),(t_4,p_5),(t_2,p_6),(t_3,p_7),(t_1,p_8),(t_4,p_9),(t_4,p_{10})\}$$
$$\mathcal{D}_1^5 = \{(t_1,p_1),(t_2,p_2),(t_3,p_3),(t_4,p_4),(t_5,p_5),(t_5,p_6),(t_1,p_7),(t_2,p_8),(t_3,p_9),(t_4,p_{10})\}$$
$$\mathcal{D}_2^5 = \{(t_5,p_1),(t_4,p_2),(t_2,p_3),(t_4,p_4),(t_5,p_5),(t_5,p_6),(t_3,p_7),(t_1,p_8),(t_1,p_9),(t_4,p_{10})\}$$
$$\mathcal{D}_3^5 = \{(t_5,p_1),(t_5,p_2),(t_5,p_3),(t_5,p_4),(t_4,p_5),(t_3,p_6),(t_2,p_7),(t_1,p_8),(t_2,p_9),(t_1,p_{10})\}$$
$$\mathcal{D}_1^6 = \{(t_1,p_1),(t_2,p_2),(t_3,p_3),(t_4,p_4),(t_5,p_5),(t_6,p_6),(t_1,p_7),(t_2,p_8),(t_3,p_9),(t_4,p_{10})\}$$
$$\mathcal{D}_2^6 = \{(t_5,p_1),(t_4,p_2),(t_4,p_3),(t_2,p_4),(t_5,p_5),(t_6,p_6),(t_3,p_7),(t_4,p_8),(t_1,p_9),(t_2,p_{10})\}$$
$$\mathcal{D}_3^6 = \{(t_6,p_1),(t_2,p_2),(t_2,p_3),(t_4,p_4),(t_5,p_5),(t_1,p_6),(t_6,p_7),(t_2,p_8),(t_3,p_9),(t_4,p_{10})\}$$
$$\mathcal{D}_1^7 = \{(t_1,p_1),(t_2,p_2),(t_3,p_3),(t_4,p_4),(t_5,p_5),(t_1,p_6),(t_7,p_7),(t_1,p_8),(t_2,p_9),(t_3,p_{10})\}$$
$$\mathcal{D}_2^7 = \{(t_4,p_1),(t_5,p_2),(t_5,p_3),(t_1,p_4),(t_5,p_5),(t_7,p_6),(t_6,p_7),(t_2,p_8),(t_2,p_9),(t_3,p_{10})\}$$
$$\mathcal{D}_3^7 = \{(t_4,p_1),(t_6,p_2),(t_1,p_3),(t_7,p_4),(t_7,p_5),(t_5,p_6),(t_2,p_7),(t_3,p_8),(t_5,p_9),(t_5,p_{10})\}$$
$$\mathcal{D}_1^8 = \{(t_1,p_1),(t_2,p_2),(t_3,p_3),(t_4,p_4),(t_5,p_5),(t_6,p_6),(t_7,p_7),(t_8,p_8),(t_1,p_9),(t_2,p_{10})\}$$
$$\mathcal{D}_2^8 = \{(t_4,p_1),(t_6,p_2),(t_1,p_3),(t_7,p_4),(t_7,p_5),(t_8,p_6),(t_2,p_7),(t_3,p_8),(t_5,p_9),(t_5,p_{10})\}$$
$$\mathcal{D}_3^8 = \{(t_4,p_1),(t_6,p_2),(t_1,p_3),(t_3,p_4),(t_7,p_5),(t5,p_6),(t_2,p_7),(t_8,p_8),(t_8,p_9),(t_8,p_{10})\}$$
$$\mathcal{D}_1^9 = \{(t_1,p_1),(t_2,p_2),(t_3,p_3),(t_4,p_4),(t_5,p_5),(t_6,p_6),(t_7,p_7),(t_8,p_8),(t_9,p_9),(t_2,p_{10})\}$$
$$\mathcal{D}_2^9 = \{(t_7,p_1),(t_8,p_2),(t_9,p_3),(t_9,p_4),(t_4,p_5),(t_1,p_6),(t_3,p_7),(t_2,p_8),(t_5,p_9),(t_6,p_{10})\}$$
$$\mathcal{D}_3^9 = \{(t_6,p_1),(t_9,p_2),(t_2,p_3),(t_3,p_4),(t_8,p_5),(t_2,p_6),(t_4,p_7),(t_7,p_8),(t_1,p_9),(t_5,p_{10})\}$$
$$\mathcal{D}_1^{10} = \{(t_1,p_1),(t_2,p_2),(t_3,p_3),(t_4,p_4),(t_5,p_5),(t_6,p_6),(t_7,p_7),(t_8,p_8),(t_9,p_9),(t_{10},p_{10})\}$$
$$\mathcal{D}_2^{10} = \{(t_8,p_1),(t_5,p_2),(t_9,p_3),(t_2,p_4),(t_7,p_5),(t_6,p_6),(t_4,p_7),(t_{10},p_8),(t_3,p_9),(t_1,p_{10})\}$$
$$\mathcal{D}_3^{10} = \{(t_{10},p_1),(t_1,p_2),(t_2,p_3),(t_3,p_4),(t_4,p_5),(t_5,p_6),(t_6,p_7),(t_7,p_8),(t_8,p_9),(t_9,p_{10})\}$$

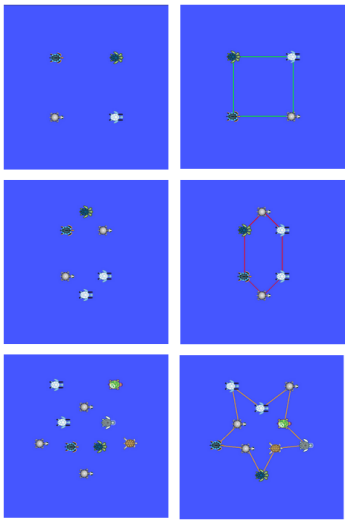
Figura 5. Distribuições de papeis



(a) Um agente construindo as formas geométricas



(b) Dois agentes construindo as formas geométricas



(c) Distribuição mais eficiente de papeis para construção das figuras

Figura 6. Exemplos de construção de figuras geométricas com diferentes distribuições de papeis

Observa-se também que o sistema multirrobô é capaz de se adaptar a diferentes configurações, como diferentes quantidades de robôs e formas geométricas. A combinação de *Jason*, ROS e $\mathcal{M}$OISE permite que os robôs se ajustem dinamicamente, maximizando a eficiência e o desempenho coletivo, independentemente da configuração específica em que estejam operando, característica não presente de forma nativa no ROS em sua configuração padrão.

5. Trabalhos relacionados e conclusões

Este trabalho permite concluir que integração de ROS com agentes possibilita a utilização de MAOP para coordenar robôs, além de observar os diferentes comportamentos

destes robôs em diferentes circunstâncias de operação. Pode-se dizer que um sistema multiagente organizado com $\mathcal{M}$OISE permite que diferentes configurações de sistemas multirrobôs ROS (simulados, no caso dos experimentos feitos) trabalhem de maneira eficiente. Através do $\mathcal{M}$OISE, é possível estabelecer uma estrutura organizacional flexível, permitindo a distribuição adequada de papéis entre os agentes. Isso possibilita uma alocação eficiente de recursos e responsabilidades, além de facilitar a distribuição de missões específicas para cada papel. Enquanto a coordenação de agentes é um tópico bastante consolidado, com diversos modelos e ferramentas, este resultado também depende da integração dos agentes com robôs ROS. O modelo de integração (e as ferramentas correspondentes) utilizados neste trabalho é um entre outros existentes. Integração entre agentes BDI e ROS é tratada em outros trabalhos, utilizando *Jason* e outras linguagens de programação [Onyedinma et al. 2020, Cardoso et al. 2020, Silva et al. 2020]. Estes trabalhos, porém, não levam em conta a coordenação dos robôs ROS. O uso de outros modelos de integração entre agentes e ROS, bem como a cooperação entre robôs integrados a estes diferentes modelos, são trabalhos futuros.

Referências

Boissier, O., Bordini, R. H., Hübner, J. F., and Ricci, A. (2020). *Multi-Agent Oriented Programming – Programming Multi-Agent Systems Using JaCaMo*. The MIT Press.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. J. (2007). *Programming Multi-Agent Systems in AgentSpeak Using Jason*. J. Wiley.

Cardoso, R. C., Ferrando, A., Dennis, L. A., and Fisher, M. (2020). An interface for programming verifiable autonomous agents in ros. In Bassiliades, N., Chalkiadakis, G., and de Jonge, D., editors, *Multi-Agent Systems and Agreement Technologies*, pages 191–205, Cham. Springer International Publishing.

Hübner, 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. *IJAOSE*, 1(3/4):370–395.

Koubaa, A. (2017). *Robot Operating System (ROS): The Complete Reference (Volume 2)*. Springer, 1st edition.

Onyedinma, C., Gavigan, P., and Esfandiari, B. (2020). Toward campus mail delivery using bdi. *Journal of Sensor and Actuator Networks*, 9(4).

Rao, A. S. (1996). Agentspeak(l): Bdi agents speak out in a logical computable language. In Van de Velde, W. and Perram, J. W., editors, *Agents Breaking Away*, pages 42–55, Berlin, Heidelberg. Springer Berlin Heidelberg.

Rao, A. S. and Georgeff, M. P. (1995). BDI agents: From theory to practice. In *Proceedings of the First International Conference on Multiagent Systems, June 12-14, 1995, San Francisco, California, USA*, pages 312–319.

Silva, G. R., Becker, L. B., and Hübner, J. F. (2020). Embedded architecture composed of cognitive agents and ros for programming intelligent robots. *IFAC-PapersOnLine*, 53(2):10000–10005. 21st IFAC World Congress.

Telang, T. (2020). *Introduction to YAML: Demystifying YAML Data Serialization Format*. Amazon Digital Services LLC - Kdp.

Desenvolvimento de um protótipo de braço robótico integrado a um agente BDI

Pedro K. B. Kano[1], **Maiquel de Brito**[2]

[1]Universidade Federal de Santa Catarina (UFSC)
Joinville – SC – Brasil

[2]Universidade Federal de Santa Catarina (UFSC)
Blumenau – SC - Brasil

pedro.doka2012@gmail.com, maiquel.b@ufsc.br

Abstract. *Certain applications, such as smart factories in Industry 4.0, require devices that autonomously interact with the physical world and are capable of reacting and adapting appropriately to changes in the system while maintaining their long-term objectives. BDI agents are a suitable metaphor for developing systems with these characteristics. This article describes the implementation of a robotic arm, typical of industrial applications, that is controlled by a BDI agent. The features of autonomy, proactivity, and reactivity are experimentally evaluated based on the system's behavior*

Resumo. *Certas aplicações, como por exemplo as fábricas inteligentes da Indústria 4.0, requerem dispositivos que atuem sobre o mundo físico de forma autônoma e que sejam capazes de reagir e adaptar-se adequadamente a mudanças no sistema mantendo seus objetivos de longo prazo. Agentes BDI são uma metáfora adequada para o desenvolvimento de sistemas com estas características. Este artigo descreve a implementação de um braço robótico, típico de aplicações industriais, que é controlado por agente BDI. As características de autonomia, proatividade e reatividade são avaliadas experimentalmente a partir do comportamento do sistema.*

1. Introdução

Sistemas robóticos (ou simplesmente *robôs*) são essenciais nos processos de manufatura das fábricas inteligentes (ou *smart factories*) da Indústria 4.0 [Rojko 2017]. Nelas, é esperado que os robôs colaborem uns com os outros para realizar tarefas que seriam difíceis ou até mesmo impossíveis para um único robô. Essa atuação deve ser realizada sem a intervenção de um operador externo e deve ser reconfigurável, de forma que os diferentes robôs adaptem-se a mudanças nas circunstâncias vigentes na linha de produção [Henning Kagermann 2013]. Para prover essas características às fábricas inteligentes, os robôs precisam ser (i) *autônomos*, para atuarem sem a intervenção de um operador externo; (ii) *socialmente capacitados*, para colaborarem entre si nos diferentes processos de manufatura; (iii) *proativos*, para desempenharem tarefas orientados a objetivos de longo prazo, que se mantém mesmo que as condições de operação se modifiquem; e, ao mesmo tempo (iv) *reativos*, para adaptarem-se tempestivamente a novas condições de operação [Aliyuda 2016].

Agentes – e, em particular, aqueles que seguem a arquiteura BDI – são uma metáfora computacional adequada para o desenvolvimento de sistemas com as características mencionadas anteriormente [Rao and Georgeff 1995, Wooldridge 2009]. No entanto, a utilização desta metáfora no desenvolvimento de robôs requer que os mecanismos de percepção e ação destes agentes sejam integrados elementos que compõem o *hardware* do robô. Este artigo explora o uso de ferramentas de programação orientada a agentes e de integração entre agentes e *hardware* para a construção de robôs que apresentem autonomia, habilidades sociais, proatividade e reatividade. Para isso, descreve-se a construção de um braço robótico cujo comportamento é especificado na forma de um agente BDI. Este braço robótico faz parte de um sistema mais amplo, em que interage com outros agentes e no qual precisa atuar de forma autônoma, proativa e reativa. As características de autonomia, proatividade e reatividade são avaliadas experimentalmente a partir do comportamento do sistema.

2. Fundamentação

Um *agente* pode como definido por sistema situado em um ambiente do qual tem percepções e no qual atua de forma autônoma para satisfazer os objetivos para os quais foi projetado [Wooldridge 2009]. Múltiplos agentes podem compor um Sistema Multiagente (SMA). Neste tipo de sistema, os agentes podem interagir uns com os outros para satisfazer seus objetivos. A *arquitetura* de um agente define os componentes que o compõem e a forma como estes componentes interagem entre si para que o agente raciocine, decida e atue [Wooldridge 2009]. Neste trabalho, considera-se especificamente a arquitetura BDI, segundo a qual um agente atua a partir dos seguintes elementos [Bratman et al. 1988, Rao and Georgeff 1995]:

- Crenças (Beliefs): informações que o agente possui sobre o mundo em que se encontra;
- Desejos (Desires): estados do mundo que o agente gostaria de (mas não está necessariamente atuando para) atingir;
- Intenções (Intentions): estados do mundo que o agente está atuando para atingir.

AgentSpeak é uma linguagem baseada em lógica para o desenvolvimento de agentes usando construtores de alto nível baseados na arquitetura BDI [Rao 1996], tais como crenças, objetivos e planos. *Jason* é um interpretador para *AgentSpeak* que inclui funcionalidades para comunicação entre agentes, permitindo desse modo a interação entre agentes e, por consequência, a implementação de SMA [Bordini et al. 2007]. Com a utilização do *Jason*, é possível criar agentes programando-se crenças, objetivos (equivalentes a desejos do modelo BDI) e planos de ação. A partir destes elementos, o interpretador da linguagem continuamente atualiza crenças dos agentes, avalia seus objetivos, produz intenções a partir dos objetivos factíveis, seleciona planos para satisfazer as intenções e os coloca em execução. Dessa forma, é possível desenvolver agentes que tenham (i) autonomia, para agir independentemente, sem a necessidade da intervenção ou supervisão de humanos ou outros sistemas; (ii) proatividade, para tomar a iniciativa sobre as ações a serem realizadas para atingir seus objetivos; (iii) reatividade, para adaptar-se a mudanças no sistema; e (iv) habilidades sociais, para se comunicar e cooperar com outros agentes.

O *framework embedded-mas*[1] fornece extensões para que agentes Jason sejam integrados a dispositivos físicos dotados de sensores e atuadores. Com essas extensões, as

[1] Disponível em `https://github.com/embedded-mas/embedded-mas`

percepções adquiridas pelos agentes incluem valores obtidos através de sensores físicos e o repertório de ações do agente inclui aquelas habilitadas por atuadores físicos. As percepções são obtidas a partir dos sensores físicos de maneira *passiva*, de modo que o agente não precisa atuar para obter valores dos sensores. Ao contrário, modificações nestes produzem percepções de forma automática. As ações dos agentes que são habilitadas por atuadores físicos são ações internas (ou *internal actions*). Assim, fazem parte da implementação do próprio agente em vez de serem tratadas como ações executadas sobre um dispositivo externo a ele.

3. Desenvolvimento

Para a análise das características buscadas (autonomia, proatividade, reatividade e habilidade social), foi montado um cenário em que um braço robótico está disposto entre um *buffer* e um *rack*, interagindo com um veículo autônomo (ou *delivery robot*), como ilustrado na Figura 1. O braço robótico é projetado como um agente BDI. Ele tem o objetivo de manter o *buffer* vazio, movendo os objetos postos sobre ele até o *rack*. Caso o rack já contenha um objeto, o braço robótico deverá primeiro enviá-lo através do *delivery robot*, para então esvaziar o buffer. Caso o *delivery robot* esteja carregando um objeto sobre si, ele pode solicitar que o braço robótico o remova. O braço robótico então passa a ter o objetivo de retirar o objeto do *delivery robot* e levá-lo para o *rack*, caso este não esteja ocupado. Caso o *rack* esteja ocupado, o braço robótico envia uma mensagem ao *delivery robot*, informando que que não será possível atender a sua solicitação.

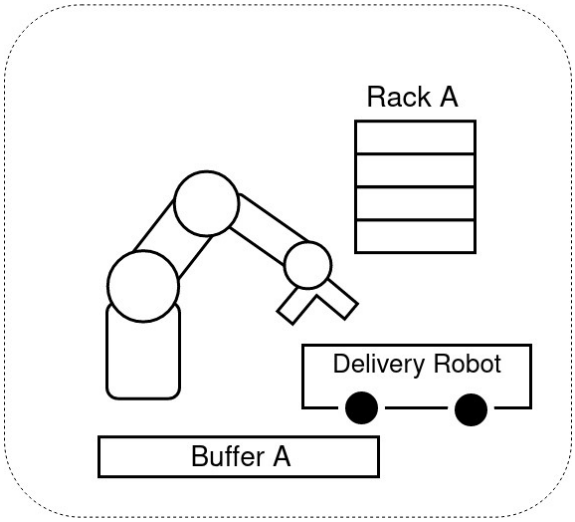


Figura 1. Representação do cenário em que o agente atuará

O braço robótico é construído a partir de um *kit* de robótica. Sua estrutura é composta de peças de acrílico cortadas a laser, contando com quatro micro servo motores 9g SG90, um microcontrolador Raspberry Pi Pico e uma fonte de alimentação externa. Os sensores de detecção de objetos no *buffer* e no *rack* são simulados através de botões. O braço robótico pode se movimentar entre os mais variados pontos, sendo restringido apenas pela limitação angular de cada micro servo motor. Para o cenário analisado, definiu-se quatro pontos em que o braço robótico será capaz de se posicionar: posição inicial, posição de retirada de itens do *buffer*, posição de depósito e retirada de itens do *rack* e posição de depósito e retirada de itens sobre o *delivery robot*. Além da movimentação

realizada por três dos servos motores que define a posição do braço robótico, haverá a movimentação da garra realizada pelo servo motor restante. Diferente do braço robótico, o *delivery robot* é simulado por um agente sem um correspondente físico real.

A estrutura física do braço robótico é integrada a um agente desenvolvido em *Jason*. Um microcontrolador *Raspberry Pi Pico*, que controla os sensores e atuadores do dispositivo, é conectado à porta serial de um equipamento com mais recursos, em que o agente *Jason* é executado. Este equipamento é, tipicamente, um computador pessoal ou um *single board computer*, como, por exemplo, *Raspberry Pi*. A cada ciclo de raciocínio do agente, os valores adquiridos pelos sensores são lidos pelo seu sistema de percepções e transformados em crenças. As ações do agente que são efetivamente realizadas através de atuações do braço robótico são transmitidas ao microcontrolador, que manipula os atuadores adequadamente.

O comportamento do agente é implementado usando elementos disponibilizados pelo *Jason*, que são, basicamente, crenças, objetivos, planos e ações internas. A Figura 2 exibe as principais partes desta implementação. Em sua atuação, o agente considera as seguintes crenças, obtidas através de percepções ligadas aos sensores físicos:

- *full(buffer)* e *empty(buffer)*, representando o *buffer* ocupado e vazio, respectivamente;
- *full(rack)* e *empty(rack)*, representando o *rack* ocupado e vazio, respectivamente;

O objetivo de manter o buffer vazio é representado, no código do agente, pelo objetivo *emptyBuffer*. O código do agente inclui planos para satisfazer este objetivo em diferentes circunstâncias. Entre elas, estão (i) o *buffer* vazio, quando agente espera o *buffer* estar cheio para seguir buscando atingir o objetivo *emptyBuffer* (Figura 2 - linhas 3 a 5), e (ii) o *buffer* cheio, em que o agente passa a ter o (sub)objetivo *move_buffer*, e, depois de satisfazê-lo, permanece buscando atingir o objetivo *emptyBuffer* (Figura 2 - linhas 7 a 9). Para satisfazer o objetivo *move_buffer*, o agente (i) move o item do *buffer* para o *rack* caso este último esteja vazio (Figura 2 – linhas 11 a 12) ou (ii) retira o objeto do rack, envia uma mensagem ao *delivery robot* requerendo o transporte do objeto e, finalmente, move o objeto do buffer para o rack (Figura 2 – linhas 13 a 16). O agente mantém o objetivo *emptyByffer* mesmo após executar os planos para satisfazê-lo (linhas 5 e 9). Por fim, o braço robótico pode receber requisições do *delivery robot* para retirada de um objeto, passando a ter o objetivo *remove_delivery*. A partir de tal objetivo, o braço robótico (i) envia uma mensagem ao *delivery robot*, solicitando que este se posicione para a retirada do item, e retira o objeto do *delivery robot* (Figura 2 – linhas 17 a 19) ou (ii) avisa o *delivery robot* que o *rack* está cheio e não será possível atingir tal objetivo no momento (Figura 2 – linhas 20 a 21) . O repertório de ações do agente inclui as ações *capt_B*, *capt_C* e *capt_R*, que consistem, respectivamente, em mover o braço robótico do local do *buffer* até o local do *rack*, mover o braço da posição em que o *delivery robot* se encontra até o *rack* e mover o braço robótico da posição do *rack* até o local do *delivery robot*. Estas ações são efetivamente realizadas através de atuações habilitadas pelos atuadores físicos do equipamento e, assim, são incorpradas à ação interna *defaultEmbeddedInternalAction* (Figura 2 – linhas 12, 14, 16, 19). Esta ação interna recebe três parâmetros: (i) o identificador do microcontrolador que controla o atuador que realizará a ação; (ii) o idenficador da ação a ser realizada, e (iii) uma lista de parâmetros necessários à realização da ação. Por exemplo, conforme a linha 12, ação *capt_B*, que não tem parâmetro algum, será realizada por meio de um atuador controlado por um microcontrolador identificado como *device1*.

```
1  !emptyBuffer. //initial goal
2
3  +!emptyBuffer : empty(buffer)
4    <- .wait(full(buffer));
5       !emptyBuffer.
6
7  +!emptyBuffer : full(buffer)
8    <- !move_buffer;
9       !emptyBuffer.
10
11 +!move_buffer : empty(rack)
12   <- .defaultEmbeddedInternalAction("device1","capt_B",[]).
13 +!move_buffer : full(rack)
14   <-  .send(delivery,achieve,move_delivery);
15       .defaultEmbeddedInternalAction("device1","capt_B",[]).
16
17 +!remove_delivery: empty(rack)
18   <- .send(delivery,achieve,move_delivery)
19      .defaultEmbeddedInternalAction("device1","capt_C",[]).
20 +!remove_delivery: full(rack)
21   <- .send(delivery,tell,full(rack)).
```

Figura 2. Trecho do código do agente

O código ilustrado na Figura 2 é integrado à porção física do agente, composta pelo braço robótico, através de uma especificação complementar feita em formato YAML [Telang 2020]. A Figura 3(a) ilustra partes desta especificação. Ela inclui definições sobre a conexão serial entre os sistemas de percepção e ação do agente e o microcontrolador (linhas 1 a 4) e a relação entre as ações que compõem o repertório do agente e as atuações implementadas pelo *hardware* (linhas 5 a 11). Por exemplo, a ação *capt_B*, que compõe o repertório do agente, é realizada pela atuação *captBuffer*, implementada no microcontrolador. A Figura 3 (b) ilustra parte da implementação desta atuação no código que é executado no microcontrolador. Se o agente executa a ação *capt_B*, então o microcontrolador recebe, via porta serial, o sinal de que a atuação *captBuffer* deve ser executada (Figura 3 (a) – linhas 6, 7 e Figura 3 (b) – linha 3) e, nesse caso, manipula os atuadores adequadamente Figura 3 (b) – linhas 3 a 12). [2]

4. Conclusão

A partir dos experimentos, observa-se que o braço robótico, cujo comportamento é implementado na forma de um agente BDI, comportou-se de forma autônoma, sem requerer a intervenção de um operador externo. Além disso, foi possível especificar um comportamento proativo para o braço robótico, em que ele toma a iniciativa de atingir um objetivo e atua constantemente para tal, mesmo que as circunstâncias do sistema se modifiquem (Figura 2 – linhas 3 a 9). Este comportamento proativo combina-se com um comportamento reativo, pois, ao buscar atingir tal objetivo, o agente adapta-se a mudanças no sistema. Por exemplo, se o *buffer* modificar seu estado de *vazio* para *cheio*, o agente modifica o plano que está sendo executado para atingir o objetivo *emptyBuffer* (Figura 2 – linhas 3, 4 e 7). Por fim, o braço robótico tem habilidades sociais, pois interage com o *delivery robot* para satisfazer seus próprios objetivos e também para satisfazer objetivos delegados por aquele agente. Assim, conclui-se que a programação orientada a agentes BDI, e, em particular, o uso do *Jason*, aliado a ferramentas de integração entre os sistemas de percepção e atuação

[2] O código do experimento está disponível em https://github.com/embedded-mas/raspberryPico-devs/tree/main/arm_buffer_rack

```yaml
microcontroller:
    id: device1
    serial: "COM3"
    baudRate: 9600
serialActions:
    - actionName: capt_B
      actuationName: captBuffer
    - actionName: capt_R
      actuationName: captRack
    - actionName: capt_C
      actuationName: captCar
```

(a)

```
while(Serial.available() > 0){
   String s = Serial.readString();
   if(s.equals("captBuffer")){
    for(pos = 110; pos >= 5; pos -= 1){
      rpServo1.write(pos);
      delay(15);
    }
    for(pos = 30; pos <= 120; pos += 1){
      rpServo2.write(pos);
      delay(15);
    }
    . . .
  }
}
```

(b)

Figura 3. Trecho de configuração da porção física do agente (a). As ações do agente são realizadas através de atuações implementadas no microcontrolador (b).

do agente e os sensores e atuadores do *hardware*, permitem o desenvolvimento de robôs com características essenciais às fábricas inteligentes da Indústria 4.0, que são autonomia, capacidade de interagir com outros robôs, proatividade e reatividade.

Referências

[Aliyuda 2016] Aliyuda, A. (2016). Towards the design of cyber-physical system via multi-agent system technology. *Int. Journal of Scientific & Engineering Research*, 7(10).

[Bordini et al. 2007] Bordini, R. H., Hübner, J. F., and Wooldridge, M. J. (2007). *Programming Multi-Agent Systems in AgentSpeak Using Jason*. J. Wiley.

[Bratman et al. 1988] Bratman, M. E., Israel, D. J., and Pollack, M. E. (1988). Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355.

[Henning Kagermann 2013] Henning Kagermann, Wolfgang Wahlster, J. H. (2013). Recommendations for implementing the strategic initiative industrie 4.0. Technical report, National Academy of Science and Engineering – Germany.

[Rao 1996] Rao, A. S. (1996). Agentspeak(l): Bdi agents speak out in a logical computable language. In Van de Velde, W. and Perram, J. W., editors, *Agents Breaking Away*, pages 42–55, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Rao and Georgeff 1995] Rao, A. S. and Georgeff, M. P. (1995). BDI agents: From theory to practice. In Lesser, V. R. and Gasser, L., editors, *Proceedings of the First International Conference on Multiagent Systems, June 12-14, 1995, San Francisco, California, USA*, pages 312–319. The MIT Press.

[Rojko 2017] Rojko, A. (2017). Industry 4.0 concept: Background and overview. *International Journal of Interactive Mobile Technologies (iJIM)*, 11(5):pp. 77–90.

[Telang 2020] Telang, T. (2020). *Introduction to YAML: Demystifying YAML Data Serialization Format*. Amazon Digital Services LLC - Kdp.

[Wooldridge 2009] Wooldridge, M. J. (2009). *An Introduction to MultiAgent Systems, Second Edition*. Wiley.

Identificação dos Perfis de Agentes de RPG Mediante Análise de suas Estratégias pelo DSC: Um Estudo de Caso no Contexto de Gerenciamento de Recursos Naturais

Fernanda P. Mota[1], Míriam Blank Born[2], Marilton Sanchotene de Aguiar[2] e Diana F. Adamatti[3]

[1]Universidade Católica de Pelotas (UCPEL)
Pelotas – RS – Brasil

[2]Universidade Federal de Pelotas (UFPEL)
Pelotas – RS – Brasil

[3]Universidade Federal do Rio Grande (FURG)
Rio Grande – RS – Brasil

{marilton,mbborn}@inf.ufpel.edu.br, {nanda2010,dianaada}@gmail.com

Abstract. *The main contribution of this work is an empirical and qualitative study on the motivation of agents and their elaboration of strategies, through a Role-Playing Game (RPG) in the context of water resources. The agents of this work represent RPG player profiles, which refer to the participatory management of water resources, based on data from the state of Rio Grande do Sul. As a result, we present the analysis of the habits of the players, who are the RPG agents, through a semi-structured interview, which resulted in a set of speeches, based on the Collective Subject Discourse (CSD) technique.*

Resumo. *A principal contribuição deste trabalho é um estudo empírico e qualitativo sobre a motivação dos agentes e sua elaboração de estratégias, por meio de um Role-Playing Game (RPG) no contexto de recursos hídricos. Os agentes deste trabalho representam perfis de jogadores de um RPG, sendo que o mesmo refere-se à gestão participativa dos recursos hídricos, com base em dados do estado do Rio Grande do Sul. Como resultados, apresentamos a análise dos hábitos dos jogadores, sendo estes os agentes do RPG, por meio de uma entrevista semiestruturada, a qual resultou em um conjunto de discursos, baseados na técnica do Discurso do Sujeito Coletivo (DSC).*

1. Introdução

A gestão participativa dos recursos naturais é um assunto recorrente quando se fala em meio ambiente e também os conflitos envolvidos nesta problemática. Ao pensarmos no impacto das atividades humanas, podemos observar que a qualidade e o ciclo da água podem ser afetados por estas atividades e pelos diferentes usos da água. [Tundisi 2006] afirma que o consumo da água é influenciado pela concentração populacional, a economia local e pelas atividades industriais e agrícolas.

Com o objetivo de orientar a pesquisa sobre recursos que contribuam para a conscientização e educação da sociedade em relação ao meio ambiente e a todos os aspectos envolvidos nesse contexto, ressalta-se a importância dos jogos na

educação [de Lima 2008]. De acordo com [Vygotsky et al. 1988], um dos processos de aprendizagem ocorre por meio das interações vivenciadas pelos indivíduos, de modo que os jogos lúdicos e os RPG (acrônimo de Jogos de Papéis, do inglês *Role-Playing Game*), também conhecidos como jogos de estratégia, proporcionam condições para que o indivíduo exercite uma imaginação elaborada em uma determinada situação, seguindo regras ou representando papéis na sociedade. Os Jogos Sérios podem ser definidos como desafios mentais disputados com um computador, seguindo regras específicas, e utilizando o entretenimento como meio de promover treinamento governamental ou corporativo, educação, saúde, políticas públicas e objetivos estratégicos de comunicação [Mota and Adamatti 2015].

Neste artigo, realizou-se uma análise das falas dos participantes por meio da técnica do Discurso do Sujeito Coletivo (DSC), utilizando o RPG no contexto da Gestão de Recursos Naturais. As análises apresentadas neste estudo foram baseadas em entrevistas realizadas com indivíduos que participaram de sessões de um RPG. A modelagem do jogo de RPG, utilizado como estudo de caso dessa pesquisa, é baseada em agentes com uma versão web desenvolvida em [Martins 2021].

A estrutura deste artigo é a seguinte: nas Seções 2 e **??** são apresentadas as bases teóricas deste trabalho, abordando o Discurso do Sujeito Coletivo e os jogos sérios e estratégicos, respectivamente. A Seção 3 descreve a metodologia adotada. Na Seção 4 são apresentados os discursos resultantes desta pesquisa e as discussões sobre os discursos encontrados. Por fim, a Seção 5 apresenta as conclusões e indicações para trabalhos futuros.

2. Fundamentação Teórica

O Discurso do Sujeito Coletivo (DSC) é expresso por meio de um discurso emitido na primeira pessoa (coletiva) do singular. Trata-se de um "eu" sintático que, ao sinalizar a presença de um sujeito individual do discurso, expressa uma referência coletiva do discurso singular para alcançar uma coletividade. Esse discurso coletivo expressa um sujeito coletivo que viabiliza o pensamento social [Levevre and Lefevre 2005]. No entanto, no DSC, os discursos não são anulados ou reduzidos a uma categoria comum unificadora, mas são construídos a partir de fragmentos de discursos individuais, como as peças de um quebra-cabeça [Lefevre and Lefevre 2014].

Dessa forma, o DSC é uma abordagem metodológica que busca tornar mais clara uma representação social por meio de uma estratégia discursiva, compreendendo um conjunto de representações que compõem dados imaginários. Através do modo discursivo, é possível visualizar de maneira mais vívida e direta a presença social, que se manifesta não de forma "artificial", como em figuras, tabelas ou categorias, mas sim através de um discurso que reflete como os indivíduos pensam e concebem a realidade.

Por outro lado, os Jogos Sérios têm como objetivo principal o ensino e a aprendizagem de indivíduos, podendo ser aplicados para diferentes fins em diversas áreas do conhecimento. Para [Prensky 2001], esse tipo de jogo, baseado no aprendizado, transcende o aprendizado tradicional, porque aspectos como diversão e prazer na experiência do jogo são considerados e os elementos do aprendizado não são tão evidentes [Mota and Adamatti 2015].

Já o RPG é um tipo de jogo estratégico em que os jogadores "interpretam" perso-

nagens dentro de um cenário específico, também conhecido como ambiente. Esses personagens seguem um conjunto definido de regras para organizar e delimitar suas ações e objetivos dentro do jogo [Mero and Mero 1998]. Esse tipo de jogo é amplamente utilizado em treinamentos, pois permite que os jogadores se envolvam em situações de tomada de decisão semelhantes às reais, porém sem consequências efetivas [Mero and Mero 1998].

3. Metodologia

Os indivíduos que participaram da amostra do experimento foram convidados por e-mail e correspondem ao mesmo grupo de participantes das sessões do jogo de RPG. A amostra foi composta por 16 pessoas, sendo sete mulheres e nove homens. Todos os participantes têm formação na área de Computação, sendo quatro professores, sete estudantes de graduação e cinco estudantes de pós-graduação. A faixa etária dos entrevistados varia de 16 a 66 anos, sendo todos brasileiros residentes nas cidades de Rio Grande e Pelotas.

As questões foram elaboradas seguindo a metodologia proposta por [Lefevre and Lefevre 2014], que enfatiza a importância de relacionar as questões aos objetivos da pesquisa. Para cada objetivo, é necessário formular um conjunto de perguntas adequadas, evitando induções e utilizando questões que incentivem o indivíduo a discorrer sobre o tema.

As seguintes questões foram aplicadas aos participantes do jogo de RPG durante a entrevista: *1) Qual foi o seu papel no jogo? O que você achou do seu papel? 2) Quais estratégias você utilizou durante o jogo? 3) Como você percebe as estratégias dos jogadores neste experimento? 4) Você concorda com as estratégias de outros jogadores?*

4. Resultados e Discussões

Para este artigo, foram analisados dois discursos com base nas questões propostas na seção de metodologia. Foi identificada uma Ideia Central (IC) e suas respectivas âncoras. Todos os Discursos do Sujeito Coletivo (DSCs) foram compostos na primeira pessoa do singular ou do plural, utilizando as expressões-chave (EC). Esses discursos foram identificados como DSC 1 e DSC 2. As informações que não foram consideradas relevantes para as ECs e ICs estão destacadas em preto.

As Tabelas 1, 2, 3 e 4 apresentam exemplos de EC, IC e âncora das perguntas 1, 2, 3 e 4, respectivamente, que deram origem aos dois discursos. Na coluna EC, são apresentadas as transcrições exatas das respostas dos participantes ao questionário, preservando os tempos verbais e as transcrições originais. Na coluna IC, pode-se observar a interpretação dessas respostas, e na coluna "âncora" são apresentadas as teorias associadas a cada IC.

Tabela 1. Questão 1 "Qual foi seu papel no jogo? O que você achou do seu papel?"

EC	IC	Âncora
Eu fui várias coisas já, eu já fui agricultora e agricultor é muito difícil porque, obviamente, você tem que tá tentando equilibra entre produzi alguma coisa, entre ter a sua renda sem tenta agredi o meio ambiente você vê aquele impacto, caso você tome decisões negativas só visando lucro, então são decisões difíceis de serem tomadas.	Estratégias dos jogadores no jogo de RPG	Estratégias utilizadas em jogos de RPG Sustentabilidade e a preocupação com recursos hídricos
Foi fiscal duas vezes e depois e depois vereador.	-	-

Tabela 2. Questão 2 "Quais estratégias utilizou durante o jogo?"

EC	IC	Âncora
Então, porque eu acho que se muda as estratégias durante o jogo, torna mais complexo a análise dele então, eu preferi mante a mesma estratégia sempre, assim como o agricultor, eu sempre preferi opções em que não me desse muito rendimento econômico e que também poluísse o menor possível.	Estratégias dos jogadores no jogo de RPG	Estratégias utilizadas em jogos de RPG Sustentabilidade e a preocupação com recursos hídricos
pois sai vendendo primeiro tudo depois vê o que que ia dá aí eu via o que que eu tinha de poluição o que que eu tinha ganhado de multa i o que que eu tinha ganhado de dinheiro aí quando eu via o que que eu polui e o que que eu era multada eu começava a corta algumas coisas não vendia o super premium pra diminui a poluição vendia um pouco mais caro pra ganha dinheiro pra paga a minha multa então tudo dependia do que aconteceria na primeira rodada.	Estratégias dos jogadores no jogo de RPG	Estratégias utilizadas em jogos de RPG Sustentabilidade e a preocupação com recursos hídricos

Tabela 3. Questão 3 "Como você percebe as estratégias dos jogadores neste experimento?"

EC	IC	Âncora
Eu percebi que eles adotavam estratégias visando pensando no meio ambiente analisando questões ambientais questão da água que não pode ser poluído.	Estratégias dos jogadores no jogo de RPG	Estratégias utilizadas em jogos de RPG Sustentabilidade e a preocupação com recursos hídricos
Eu percebo que alguns são bem honestos querem seguir certinho alguns querem poluir menos cuidar do meio ambiente alguns tentam fazer sindicato o que não da certo sempre alguém pula fora alguém só quer ganhar dinheiro tipo eu alguém como o Daniel tenta montar uma máfia é isso.	Estratégias dos jogadores no jogo de RPG	Estratégias utilizadas em jogos de RPG Sustentabilidade e a preocupação com recursos hídricos
Talvez pra perceber a estratégia mesmo fosse mais no sentido de leilão assim, de tá um vendedor e todos os compradores juntos pra negocia alguma coisa assim, não houve essa oportunidade.	Estratégias dos jogadores no jogo de RPG	Estratégias utilizadas em jogos de RPG Sustentabilidade e a preocupação com recursos hídricos
Mas é perceptível quando agente toma conhecimento do saldo, de quanto dinheiro tem cada personagem, então aí claramente tu consegue observa determinados interesses, mas fora isso pelo menos não foi aparente pra mim em descobrir as estratégias.	Estratégias dos jogadores no jogo de RPG	Estratégias utilizadas em jogos de RPG Sustentabilidade e a preocupação com recursos hídricos
Eu acho que quem já tem uma noção do jogo a ideia, que eu acho que o experimento passa, é que tu consegue te produtividade das cidades mesmo com baixa poluição porque se todo mundo trabalha em equipe, os prefeitos investirem em estratégias pra diminui a poluição eles vão permiti com que os agricultores, os empresários consigam produzir bastante, mesmo que tejam poluindo.	Estratégias dos jogadores no jogo de RPG	Estratégias utilizadas em jogos de RPG Sustentabilidade e a preocupação com recursos hídricos

Tabela 4. Questão 4 "Você concorda com as estratégias dos outros jogadores?"

EC	IC	Âncora
Mas eu acho assim que tu não pode te estratégias restritivas com relação a quem tem os mecanismos de maquinário e vende sementes, tu pode tipo aumenta os valores ou diminui, mas não tipo veta porque aí tu pode faze simplesmente que um agente não consiga joga, isso eu acho injusto.	Estratégias dos jogadores no jogo de RPG	Estratégias utilizadas em jogos de RPG Sustentabilidade e a preocupação com recursos hídricos

A partir das âncoras "Estratégias utilizadas em jogos de RPG" e "Sustentabilidade e preocupação com recursos hídricos" e da IC "Estratégias dos jogadores no jogo de RPG", originadas a partir das questões 1, 2, 3 e 4, foram desenvolvidos os discursos DSC 1 e DSC 2.

Os DSC 1 e DSC 2 (Tabelas 1, 2, 3 e 4) demonstram que, dependendo da estratégia utilizada no jogo de RPG, essa estratégia pode ter influência na gestão de recursos hídricos e, consequentemente, na simulação do ambiente, uma vez que as ações de

Tabela 5. DSC 1 – Estratégias visando questões ambientais.

Eu percebi que eles adotavam estratégias visando, pensando, no meio ambiente, analisando questões ambientais, questão da água que não pode ser poluído porque a água pode acabar e outros, eu via que pensavam apenas para si mesmo. São bem honestos querem seguir certinho alguns querem poluir menos cuidar do meio ambiente alguns tentam fazer sindicato, o que não dá certo, sempre alguém pula fora, alguém só quer ganhar dinheiro, tipo alguém como o outro joga tenta montar uma máfia é isso. Talvez pra perceber a estratégia mesmo, fosse mais no sentido de leilão, assim, de tá um vendedor e todos os compradores juntos pra negocia alguma coisa assim, não houve essa oportunidade. Mas é perceptível, quando a gente toma conhecimento do saldo, de quanto dinheiro tem cada personagem, então aí claramente tu consegue observa determinados interesses, mas fora isso pelo menos não foi aparente pra mim em descobrir as estratégias. Tem estratégias e não negociação, por exemplo, se tem um agente que vende semente, não quer vende, não acho isso muito justo porque isso pode prejudica todo o ambiente. Estratégias restritivas com relação a quem tem os mecanismos de maquinário e vende sementes, tu pode tipo aumenta os valores ou diminui, mas não tipo veta porque aí tu pode faze simplesmente que um agente não consiga joga, isso eu acho injusto.

Tabela 6. DSC 2 – A preocupação dos jogadores com a poluição e o lucro.

O jogador inicial normalmente se preocupa muito em como gasta ou não poluí. Eu acho que quem já tem uma noção do jogo, a ideia, que eu acho que o experimento passa, é que tu consegue te produtividade das cidades mesmo com baixa poluição porque se todo mundo trabalha em equipe, os prefeitos investirem em estratégias pra diminui a poluição eles vão permiti com que os agricultores, os empresários consigam produzir bastante, mesmo que tejam poluindo. Agricultor é muito difícil porque, obviamente, você tem que tá tentando equilibra entre produzi alguma coisa, entre ter a sua renda sem tenta agredi o meio ambiente, você vê aquele impacto, caso você tome decisões negativas só visando lucro. Então são decisões difíceis de serem tomadas, pois sai vendendo primeiro tudo depois vê o que que ia dá, aí eu via o que que eu tinha de poluição, o que que eu tinha ganhado de multa i o que que eu tinha ganhado de dinheiro, aí quando eu via o que que eu polui e o que que eu era multada, eu começava a corta algumas coisas, não vendia o super premium pra diminui a poluição, vendia um pouco mais caro pra ganha dinheiro, pra paga a minha multa, então tudo dependia do que aconteceria na primeira rodada.Percebo que muitos jogadores quiseram muito i pelo politicamente correto, faze tudo certo, não tô dizendo que isso seja uma forma errada de joga, mas como eu disse antes, o jogo dá muita liberdade pra gente faze o certo e o errado, dentro desse ambiente controlado, tinha estratégias pra pro bem maior em relação a poluição.

um jogador afetam as ações de todos os outros, sendo que o ambiente é compartilhado. Esse fenômeno pode ser explicado pelo fato de que o RPG tem seu foco na narração e participação [Silva 2019].

O RPG possui um sistema de regras que orienta o jogo e a construção dos personagens, possibilitando a elaboração de soluções que vão além de meras declarações verbais ou de atuações abstratas de peças teatrais [Silva 2019]. Ao considerar as estratégias relacionadas ao tema do RPG utilizado neste estudo, [Tundisi 2006] afirma que o impacto na qualidade e no ciclo da água ocorre devido às atividades humanas e aos múltiplos usos deste recurso.

O autor [Tundisi 2006] destaca que a crise da água está mais relacionada à gestão dos recursos naturais do que à escassez e contaminação em si. Para que a gestão da água seja eficiente, é necessário integrar o conhecimento científico com o gerenciamento, sendo importante descentralizar os sistemas de gerenciamento, organizando-os por bacias hidrográficas, e promover a integração com agências regionais e municípios.

5. Conclusões

Neste artigo, realizou-se uma análise do Discurso do Sujeito Coletivo (DSC) no contexto da Gestão de Recursos Naturais, utilizando o RPG como ferramenta. O objetivo principal deste estudo consistiu em examinar os hábitos e estratégias dos participantes durante o jogo de RPG. As análises apresentadas neste trabalho foram baseadas nas transcrições das entrevistas realizadas com um grupo de indivíduos que participaram do RPG, cuja temática envolvia a negociação participativa de conflitos na bacia hidrográfica da Lagoa Mirim e do Canal São Gonçalo, localizada no Sul do Brasil.

Os dados foram analisados utilizando a técnica do DSC, possibilitando uma análise qualitativa e quantitativa com base em expressões verbais. Isso resultou na criação de um ou mais discursos de síntese, escritos na primeira pessoa do singular, que refletem o pensamento coletivo do grupo. Como resultado, obtivemos uma variedade de discursos que foram analisados à luz de conceitos teóricos relacionados à importância da gestão dos recursos hídricos e ao impacto da poluição no meio ambiente. Concluímos que os jogos, especialmente o RPG, têm o potencial de auxiliar no processo de aprendizagem dos indivíduos, proporcionando cenários nos quais eles podem experimentar novas atitudes ou comportamentos de forma segura, refletindo a realidade.

Como sugestões para trabalhos futuros, propomos: i) Desenvolver um assistente virtual para auxiliar os jogadores na análise do discurso. ii) Automatizar a metodologia de análise do discurso do sujeito coletivo. iii) Verificar as alterações no RPG com base nas sugestões dos entrevistados. Essas propostas visam aprimorar a metodologia de análise, promover maior acessibilidade aos jogadores e aprofundar a compreensão sobre o impacto do RPG no contexto da gestão de recursos hídricos.

Referências

de Lima, J. M. (2008). O jogo como recurso pedagógico no contexto educacional. *São Paulo: Cultura Acadêmica: Universidade Estadual Paulista, Pró-Reitoria de Graduação*.

Lefevre, F. and Lefevre, A. M. C. (2014). Discourse of the collective subject: social representations and communication interventions. *Texto & Contexto-Enfermagem*, 23(2):502–507.

Levevre, F. and Lefevre, A. (2005). O discurso do sujeito coletivo: Um enfoque em pesquisa qualitativa.

Martins, V. B. (2021). Gorimweb: um rpg para gestão de recursos hídricos na plataforma web. Dissertação (mestrado em engenharia da computação), Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande, Rio Grande, RS.

Mero, G. T. and Mero, M. J. (1998). Role playing game. US Patent 5,810,666.

Mota, F. P. and Adamatti, D. F. (2015). Programming teaching in high schools: An analysis based on the discourse of collective subject. In *2015 IEEE Frontiers in Education Conference (FIE)*, pages 1–5. IEEE.

Prensky, M. (2001). Digital natives, digital immigrants. *On the horizon*, 9(5).

Silva, E. A. d. (2019). Integração conceptual sob a ótica da cognição ecológica nos jogos de rpg. Master's thesis, Brasil.

Tundisi, J. G. (2006). Novas perspectivas para a gestão de recursos hídricos. *Revista USP*, 1(70):24–35.

Vygotsky, L. S. et al. (1988). Aprendizagem e desenvolvimento intelectual na idade escolar. *Linguagem, desenvolvimento e aprendizagem*, 10:103–117.

Improving the Mapping of $\mathcal{M}$oise$^+$ to Colored Petri Nets

Ricardo A. Machado[1], **Diana F. Adamatti**[1], **Eder M. Gonçalves** [1]

[1] Programa de Pós-Graduação em Modelagem Computacional
Centro de Ciências Computacionais – Universidade Federal do Rio Grande (FURG)
Rio Grande – RS – Brasil

{ricardoarend,dianaada,eder.m.goncalves}@gmail.com

Abstract. *The demand for systems with artificial intelligence such as Multiagent Systems (MAS) is continuously growing. At the same time, the need for the development of tools that help software development increases, ensuring better fault tolerance for the project, since these systems have characteristics that make the system non-deterministic and increase the difficulty in carrying out tests. In previous work, an approach was presented for tests in MAS that use the organizational model $\mathcal{M}$oise$^+$, where test cases were generated using Colored Petri Nets (CPN) to map the system in a formal model. In this work, an update is presented for the inclusion of more elements contained in $\mathcal{M}$oise$^+$ that will be mapped in the CPN, more specifically, the minimum and maximum cardinality for each existing mission.*

1. Introduction

Multiagent Systems (MAS) represent an innovative paradigm for modeling, simulating, and designing complex distributed systems. Its effectiveness is due to the ability of agents to represent system entities, their behaviors, and their interactions. In fact, an agent has proactive and reactive characteristics that are very useful in decision-making. In addition, a fundamental feature of MAS is collective or distributed intelligence, and the ability to function even with the failure of one of its components (agent) without the need to interrupt the system as a whole [Boucherit et al. 2020].

Over time, MAS has gained importance in the most different areas, due to the unique characteristics of the agents, such as reactivity, proactivity, autonomy, and social capacity [Padgham and Winikoff 2005]. However, certain applications require a minimum of reliability so that the system does not present any risk to the user. One can cite as an example the control of autonomous vehicles, military applications, Smart Grids, and logistics.

To ensure that the system is reliable and can be made available to the user, software testing is necessary. Software testing consists of dynamically verifying the behavior of a program against a set of suitably selected test cases [Abran et al. 2004]. A test case is an input on which the program under test is executed while testing [Zhu et al. 1997], and testers often need to generate test cases to execute all statements in the program at least once.

Given that agents are often required to perform multiple tasks simultaneously, Petri nets are a particularly effective tool for specifying and testing the behavior of agent-based systems. Furthermore, Petri nets are one of the most reliable formalisms for simulation and analysis of behavioral modeling and MAS failure analysis [Boucherit et al. 2020].

In a previous work [Gonçalves et al. 2022], a formalization for generating test cases was described starting from an organizational model $\mathcal{M}$oise$^+$ and, mapping the model in a Colored Petri Net (CPN) to generate the test cases from two different coverage criteria. One of the limitations of this model is the fact that it does not take into account the number of agents committed to a mission.

In some tests, it was found that in $\mathcal{M}$oise$^+$ if a mission does not have the minimum number of agents indicated by the cardinality, it is not satisfied and the system ends up in a waiting state. On the other hand, if the number of agents is greater than the maximum cardinality, some agents are left out of the mission and an error message is generated, which does not prevent the other agents from completing the mission normally. To ensure that the generated CPN is as faithful as possible to the original $\mathcal{M}$oise$^+$ model and that more test cases can be generated, a proposal for updating the mapping in CPN is presented here, where the inclusion of minimum and maximum cardinality becomes necessary.

2. Related Works

As examples of related work, we can cite the work of [Frasheri et al. 2017], where an initial concept of an adaptive autonomous agent was analyzed about fault tolerance. Failure analysis is performed by modeling the agents through Colored Petri Nets. The simulation results indicate the number of tasks that are discarded or completed by the agents. Based on the experimental results, the analysis showed the correlation between the probability of system failure (increase in the number of lost tasks) and the failure of an agent.

In [Rehman et al. 2019] is presented a model-based testing approach that uses the *Prometheus* tool for MAS modeling. For the generation of test cases, some different coverage criteria defined by the authors were used. The generation of test paths is automated with the help of a tool that takes a test model as input, applies the different coverage criteria, and generates a test path for each coverage criterion.

[Dehimi and Mokhati 2019] proposes a model-based testing approach to test agent interactions. The approach uses an AUML sequence diagram as a model and constraints expressed in the Object Constraint Language (OCL). The approach generates a set of test cases capable of, individually, covering interactions between agents, as well as possible scenarios that can be executed in an inclusive, exclusive, or parallel way.

In [Boucherit et al. 2020] the authors use Petri nets and rewrite logic to facilitate the formalization of critical security applications based on multi-agent. They present an algorithm that allows the automatic generation of Maude specifications for models of multi-agent systems based on Petri nets. In addition, model verification and property-based testing techniques are integrated into the verification and testing phases.

3. Theoretical Background

This work demonstrates how a MAS with an organizational model can be mapped in a CPN. For this, the model $\mathcal{M}$oise$^+$ was chosen, which contains an independent Structural Specification (SS) and a Functional Specification (FE), linked by a Deontic Specification (DS) [Hübner and Sichman 2007]. The $\mathcal{M}$oise$^+$ is described using notions such as groups and roles and the links between them in the structural dimension. The functioning of the system is described by global goals that must be achieved and their missions in the

functional dimension. Finally, the deontic dimension unites the structural and functional aspects, defining permissions and obligations for each function and mission.

According [Boissier et al. 2020] in $\mathcal{M}$oise$^+$ there are two basic types of cardinality constraint: a *role cardinality* that defines upper and lower bounds on the number of agents that can play the role in the corresponding group entity and a *mission cardinality* that constrains the minimum and maximum number of agents that can commit to a mission. In this work, the focus is on mission cardinality.

A Petri Net is a graphical and mathematical modeling tool applicable to many systems. They are used as visual communication aid similar to flowcharts. Petri Nets are divided into two classes: low-level and high-level.

In high-level Petri nets, we have Colored Petri Nets (CPN) as an example, which combine the capabilities of an ordinary Petri Net with the capabilities of a programming language. In CPN, tokens have a data value attached called color. These colors can represent different processes or resources of a network, thus reducing the size of the models. For a given place, all tokens must have colors that belong to a specific type which is called the place color set. This use of color sets is equivalent to data types in programming languages, and the specification of color sets and network variables is done by declarations [Jensen 1997].

CPN composition contains a structure, inscriptions, and declarations. It is a directed graph with two types of vertices: (i) circles, or ellipses represent the places, and (ii) the transitions by rectangles. Each place, transition, and arc of the network has associated inscriptions. For example, places have three types of inscriptions: names, color sets, and initialization expression, which is the initial mark. The declarations are the specifications of the color sets and the declaration of the variables and functions.

4. Mapping the $\mathcal{M}$oise$^+$ Model to a CPN

This section will present all the tree steps for the mapping a $\mathcal{M}$oise$^+$ model in a CPN, namely: (i) Generate the Declarations, (ii) Model the Structure and (iii) Include the Inscriptions. Each of these steps is detailed below.

4.1. Generating Declarations

The first step in mapping is generating the declarations. These declarations play an important role in the functioning of the CPN. The first declaration is the color set (*colset*) *R* that represents the existing roles in the $\mathcal{M}$oise$^+$ specification. Here, as an example, we have two roles, *roleA* and *roleB*, which are separated by a vertical bar. Next, we have the *G* color set that represents the groups of the specification, here being *groupA* and *groupB* and below the *RG* color set that represents the product between the two previous sets in order to relate each role with a respective group.

Continuing using *var*, the variables are declared. For each role on the network, it is necessary to create a new variable, in this example the variable r1 is used for the role *roleA* and the variable *r2* for the role *roleB*, both of type *RG*, and *b* for a boolean type variable. The constants (*val*) are declared in sequence, *NroleA* and *Nrole*B being used to indicate the number of agents with *roleA* and *roleB* there are, and the constants referring to the minimum and maximum cardinalities for each mission are also declared.

Finally, in the declarations, we have the functions that are used in the arcs. Here as an example are two functions (*Fmission1* and *Fmission2*) because for each mission a related function must be created. The function code is always the same, changing only the variables and constants used. The purpose of these functions is to ensure that if the maximum cardinality is reached in a number of agents for a given role, it is not exceeded, so even with more agents available, the mission only uses the number informed in the maximum cardinality. On the other hand, if the number of agents is less than or equal to the maximum cardinality, this number will be used in the mission.

```
Declarations:
colset R = with roleA | roleB;
colset G = with groupA | groupB;
colset RG = product R*G;
var r1,r2 : RG;
var b : BOOL;
val NroleA = 5;
val NroleB = 4;
val Cmin_mission1 = 2;
val Cmax_mission1 = 4;
val Cmin_mission2 = 1;
val Cmax_mission2 = 2;
fun Fmission1(r1)=if NroleA <= Cmax_mission1 then NroleA`r1
                              else Cmax_mission1`r1;
fun Fmission2(r2)=if NroleB <= Cmax_mission2 then NroleB`r2
                              else Cmax_mission2`r2;
```

4.2. Modeling the Structure

In the structure modeling, the $\mathcal{M}$oise$^+$ goals are represented by the CPN places, and the different types of $\mathcal{M}$oise$^+$ plan operators (sequence, choice and parallelism) are transformed into net structures through the mapping model presented in Figure 1 [Gonçalves et al. 2022] thus defining the way places are distributed on the network. In addition to these elements, an initial place called *start* and a final place called *end* are included in the network, indicating where the system starts and ends, as well as a place that represents the agents that are in a waiting state named *Waiting*.

If there are different missions in $\mathcal{M}$oise$^+$, it is also necessary to include auxiliary places in the network called *aux_n*, where *n* represents a numerical sequence to differentiate each of these places.

In Figure 2 an example of a structure is presented that represents a sequence between two goals *goal1* and *goal2* and each of these goals belongs to a mission, with *goal1* belonging to *mission1* and *goal2* to *mission2*. As there is this alternation between missions, an auxiliary place aux_1 is also included between *goal1* and *goal2*. The inclusion of the names for the places is considered as part of the step of including inscriptions and not of the modeling of the structure, in Figure 2 these names were included for a better understanding of the elements exposed here.

4.3. Including the Inscriptions

Once the structure of the CPN is finished, the stage of the inclusion of inscriptions begins. In this step, the names of the places, their initial markings, and their type are included, as well as the names of the transitions, functions, and arc variables. Figure 3 shows the same example with all inscriptions. In the *Waiting* place there are tokens that represent the available agents waiting their turn to satisfy the goals of their missions and an initial

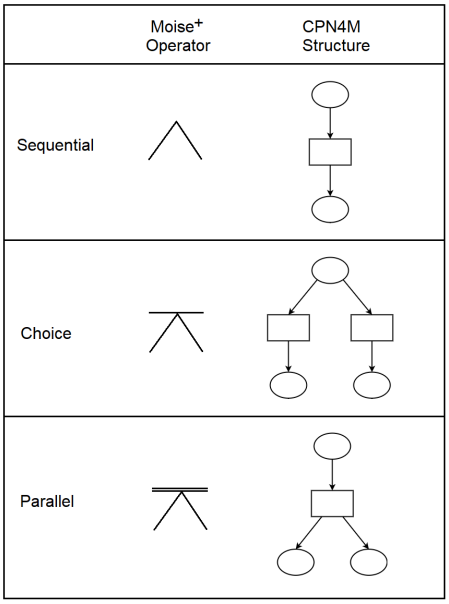


Figura 1. A Mapping between FS operators and CPN structures [Gonçalves et al. 2022].

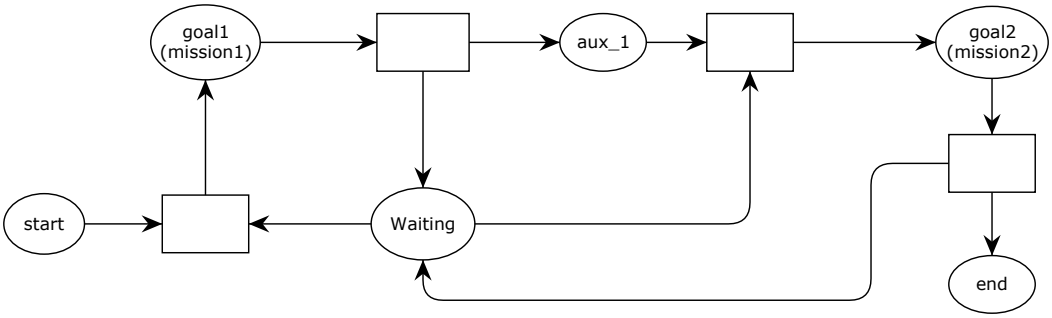


Figura 2. Structure of a Generic System mapped in CPN

marking is made with *NroleA'(roleA,groupA)* where *NroleA* is the number of *roleA* agents belonging to the group *groupA* , previously defined in the declarations.

The places that represent the goals and the *Waiting* place receive the color type *RG* that represents the set of colors of the roles combined with that of the groups. The initial, final and auxiliary places are of type *BOOL* (boolean) that receive a token named *true* as input, and as output the boolean variable *b*.

Functions calls are used on the entry and exit arcs of places with the type *RG*, for example between place *Waiting*, the transition *t1* and place *goal1* we have on the arcs the inscription *Fmission1(r1)* which is the call to a function that indicates the number of agents assigned to the mission *mission1*. In turn, *goal2* already belongs to *mission2*, and therefore the function *Fmission2(r2)* is used on the arcs.

Guard functions in the *t1* and *t3* transitions are used to ensure that the number of agents satisfies the minimum cardinality for each mission and also to define the type of role that will be used. For example, *t1* reads *NroleA* to be greater than or equal to *Cmin_mission1*, and that the variable *r1* contains *(roleA,groupA)*. It is only necessary to

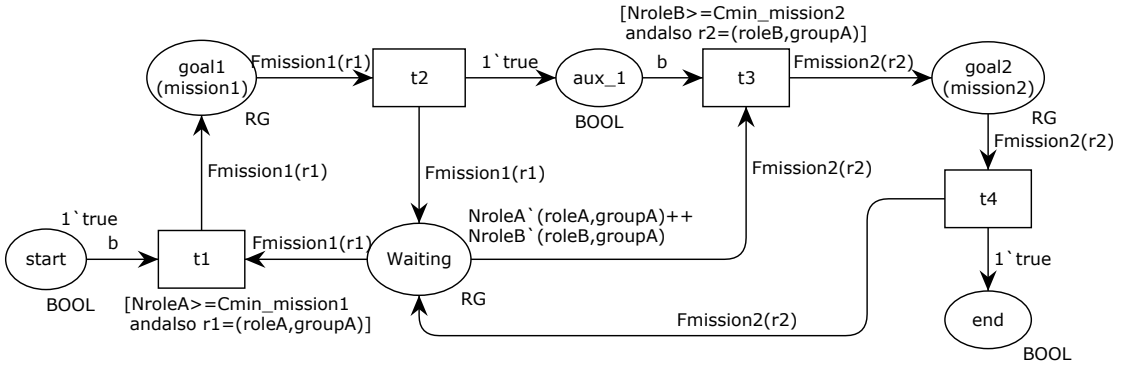


Figura 3. Inscriptions included in the CPN

use this type of guard function in transitions that receive as input an arc that came out of *Waiting*

After this stage, it is already possible to carry out simulations, using the generated model to identify the different test scenarios that the system is capable of executing. In the following section, a use case for this method will be presented, showing both a $\mathcal{M}$oise$^+$ model in detail and its respective mapping in an CPN.

5. Writing Paper Implementation Example

The scenario application used is the *Writing Paper* [Hübner et al. 2011]. It describes an agent group that has a goal to write a paper to be published. Figure 4 describes the structural specification composed by a *wpgroup* group that has two roles: *writer* and *editor*, and both roles are sub-roles from *author*.

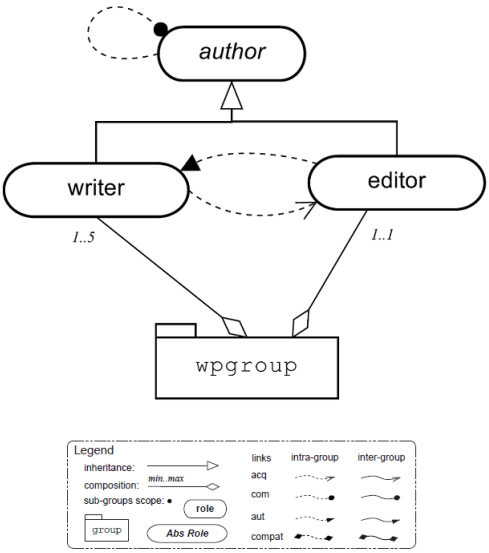


Figura 4. Structural Specification for a Writing Paper scenario, based on [Hübner et al. 2011].

The functional specification for this scenario is presented in Figure 5. According to this scheme, the first part is to conclude a paper draft (*fds*). Obeying a sequence, an

agent undertakes the *mMan* mission and must write a title (*wtitle*), an abstract (*wabs*), and write the section titles (*wsectitles*) ending this first version. The second branching, named (*sv*), the submission version is composed of the goals (*wsecs*), writing sections, and to finish the paper, it is necessary to reach two goals in parallel, (*wcon*) writing a conclusion and (*wrefs*) writing references.

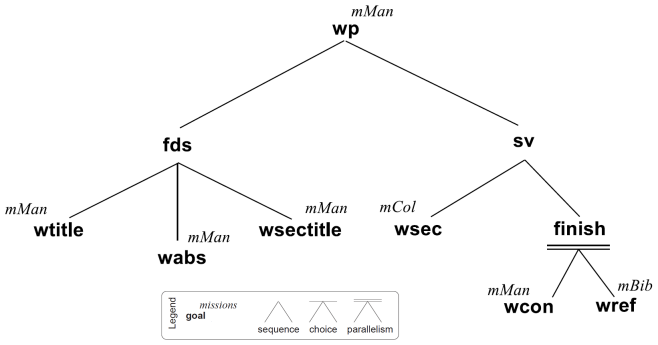


Figura 5. Functional Specification for a Writing Paper scenario, adapted from [Hübner et al. 2011].

Table 1 describes the Deontic Specification defining permissions and obligations for the roles that undertake the missions. The missions are *mMan*, project general managing; *mCol* collaborating for the writing process; and mission *mBib* for the agent which must gather and write the paper references. Here, the minimum and maximum cardinalities for each mission were also included.

Role	Deontic Relationship	Cmin	Cmax	Mission
editor	obligation	1	2	*mMan*
writer	permission	2	5	*mCol*
writer	obligation	3	3	*mBib*

Tabela 1. Deontic Specification for a Writing Paper scenario, based on [Hübner et al. 2011].

Now with the necessary information from each specification presented, will present the mapping using the sequence of steps defined in the previous section.

5.1. Generating Declarations

In the first stage and based on the $\mathcal{M}$oise$^+$ specifications presented above, the following elements are declared: The *editor* and *writer* roles, the *wpgroup* group, two variables *r1* and *r2*, and a third variable *b*, the constants for the number of agents in each role and for cardinality, and finally the functions, one for each mission:

```
Declarations:
colset R = with editor | writer;
colset G = with wpgroup;
colset RG = product R*G;
var r1,r2 : RG;
var b : BOOL;
val Neditor = 3;
val Nwriter = 4;
val Cmin_mMan = 1;
val Cmax_mMan = 2;
```

```
val Cmin_mCol = 2;
val Cmax_mCol = 5;
val Cmin_mBib = 3;
val Cmax_mBib = 3;
fun Fman(r1)=if Neditor <= Cmax_mMan then Neditor`r1 else Cmax_mMan`r1;
fun Fcol(r2)= if Nwriter <= Cmax_mCol then Nwriter`r2 else Cmax_mCol`r2;
fun Fbib(r2)= if Nwriter<=Cmax_mBib then Nwriter`r2 else Cmax_mBib`r2;
```

5.2. Modeling the Structure

With the statements done, now we present the second stage where the structure of the net is molded. Based on Figure 5 where we have the FE with a goal decomposition tree, we selected all the goals contained in a mission: *title*, *wabs*, *wsectitle*, *wcon* and *wp* belonging to the *mMan* mission; *wsec* to the *mCol* mission and finally *wref* to the *mBib* mission.

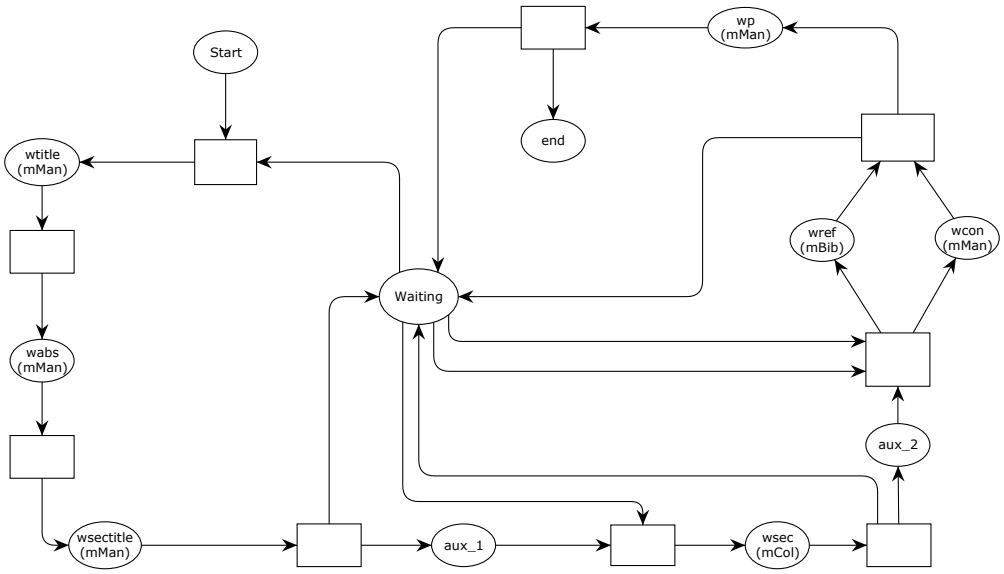


Figura 6. The Writing Paper Structure of a CPN

Figure 6 presents the structure of the CPN, where we first have a sequence of the first three goals belonging to *mMan* (*wtitle*, *wabs* and *wsectitle*). The next goal (*wsec*) also comes in sequence, but as it belongs to another mission an auxiliary place aux_1 is used. Following it we have a second auxiliary aux_2 because here the *mCol* mission ends and the net goes on to a parallelism between *wref* and *wcon* goals which belong to *mBib* and *mMan* missions respectively. As the *mBib* mission is done, a connection path to Waiting is necessary, but without the need for an auxiliary place since the net continues with the same *mMan* mission to the *wp* goal. Finally, a transition to the end place is made.

5.3. Including the Inscriptions

With the structure in place, the last step consists of including the inscriptions. Figure 7 shows the mapping in CPN ready with all inscriptions where guard functions were included in transitions *t1*, *t5* and *t7* that receive arcs from *Waiting*. The arc function calls and some expressions were also included as well as the color types of the places (*RG* and *BOOL*). In *Waiting* and *start*, initial markings are also placed.

With the CPN fully functional, it is now possible to carry out simulations by changing the number of agents available in the declarations *Neditor* and *Nwriter* and generating new test scenarios.

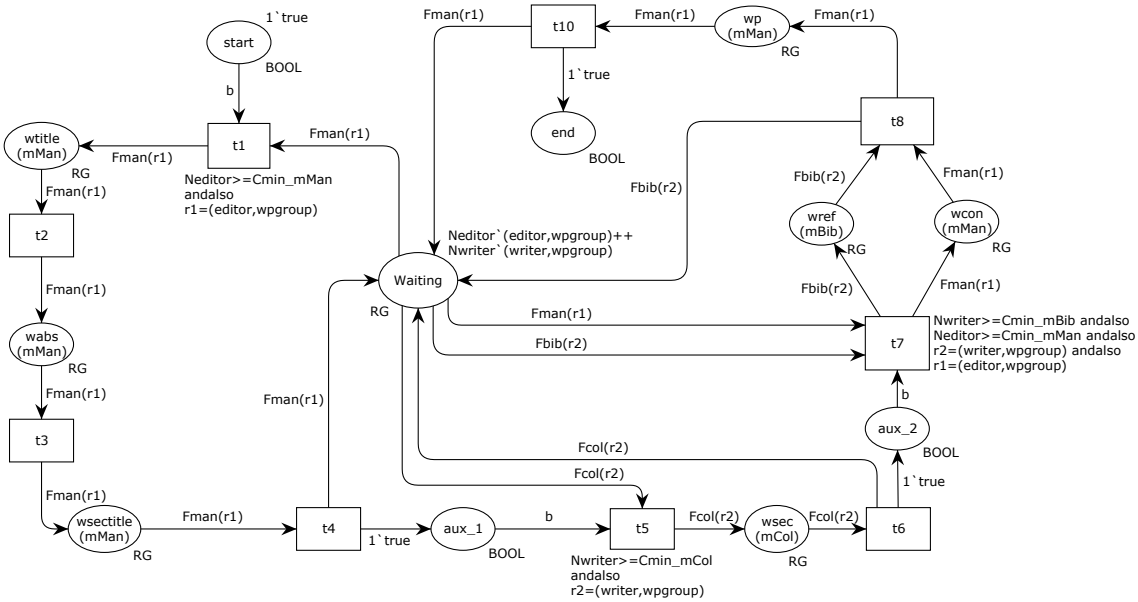


Figura 7. The Writing Paper Scenario Fully Mapped in a CPN

6. Conclusion and Future Work

In this work, a reformulation proposal was presented for the mapping method of a MAS that has the $\mathcal{M}$oise$^+$ organizational model for a CPN proposed in previous work. The purpose of this reformulation is to include elements such as cardinality, and the number of existing agents for a given role, thus generating a CPN model that is more faithful to the original system.

In future works, we have the possibility of using the method in a tool that automates the mapping, generating all the CPN elements from the $\mathcal{M}$oise$^+$ XML file. Another issue is to evaluate the need to include more elements of $\mathcal{M}$oise$^+$ so that the CPN can be used as a test tool that generates the possible test paths necessary for verifying the system. Finally, the formalization for this method will be reviewed, now taking into account the new elements included in this paper.

Referências

Abran, A., Moore, J. W., Bourque, P., Dupuis, R., and Tripp, L. (2004). Software engineering body of knowledge. *IEEE Computer Society, Angela Burgess*.

Boissier, O., Bordini, R. H., Hubner, J., and Ricci, A. (2020). *Multi-agent oriented programming: programming multi-agent systems using JaCaMo*. Mit Press.

Boucherit, A., Castro, L. M., Khababa, A., and Hasan, O. (2020). Petri net and rewriting logic based formal analysis of multi-agent based safety-critical systems. *Multiagent and Grid Systems*, 16(1):47–66.

Dehimi, N. E. H. and Mokhati, F. (2019). A novel test case generation approach based on auml sequence diagram. In *2019 International Conference on Networking and Advanced Systems (ICNAS)*, pages 1–4. IEEE.

Frasheri, M., Trinh, L. A., Cürüklü, B., and Ekström, M. (2017). Failure analysis for adaptive autonomous agents using petri nets. In *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 293–297. IEEE.

Gonçalves, E. M. N., Machado, R. A., Rodrigues, B. C., and Adamatti, D. (2022). Cpn4m: Testing multi-agent systems under organizational model m oise+ using colored petri nets. *Applied Sciences*, 12(12):5857.

Hübner, J. F., Boissier, O., and Bordini, R. H. (2011). A normative programming language for multi-agent organisations. *Annals of Mathematics and Artificial Intelligence*, 62(1-2):27–53.

Hübner, J. and Sichman, J. (2007). Developing organised multi-agent systems using the $Moise^+$ model: Programming issues at the system and agent levels. In *Int. J. Accounting, Auditing and Performance Evaluation*, pages 1–10.

Jensen, K. (1997). *Coloured Petri nets: basic concepts, analysis methods and practical use*, volume 1. Springer Science & Business Media.

Padgham, L. and Winikoff, M. (2005). *Developing intelligent agent systems: A practical guide*, volume 13. John Wiley & Sons.

Rehman, S. U., Nadeem, A., and Sindhu, M. (2019). Towards automated testing of multi-agent systems using prometheus design models. *Int. Arab J. Inf. Technol*, 16(1):54–65.

Zhu, H., Hall, P. A., and May, J. H. (1997). Software unit test coverage and adequacy. *Acm computing surveys (csur)*, 29(4):366–427.

Uma Ferramenta para Mapear o Modelo Organizacional $\mathcal{M}$oise$^+$ em Rede de Petri Colorida de Forma Automatizada

Arthur da Silva Z. Cardoso[1], Ricardo A. Machado[1], Eder M. Gonçalves[1], Diana F. Adamatti[1]

[1]Centro de Ciências Computacionais - Universidade Federal do Rio Grande (FURG)
Rio Grande - RS - Brasil

Resumo. *Este artigo apresenta como proposta o mapeamento de modelos organizacionais do $\mathcal{M}$oise$^+$ para Redes de Petri Coloridas, focando na automatização desse serviço. Desta forma, a ideia é o desenvolvimento de uma ferramenta capaz de criar uma rede de Petri colorida a partir do arquivo XML do modelo $\mathcal{M}$oise$^+$ existente. A ferramenta está sendo desenvolvida em Python e já apresenta resultados preliminares.*

1. Introdução

Devido a falta de ferramentas especializadas para o desenvolvimento automatizado de uma rede de Petri colorida (RPC) utilizando CPN Tools, percebeu-se a necessidade de desenvolver uma ferramenta capaz de automatizar, pelo menos o processo de inicialização de uma rede, facilitando sua criação e no melhor dos casos podendo gerar uma rede inteira, apenas passando como entrada o XML do modelo $\mathcal{M}$oise$^+$ existente. Portanto, tem-se o objetivo de facilitar a interação com os modelos e suas criações, utilizando um algoritmo que possa gerar uma rede de Petri colorida, mesmo que seja em sua forma inicial, usando a linguagem de programação Python, sendo um parser que gerará arquivos para os elementos de um arquivo da ferramenta CPN Tools.

Uma Rede de Petri é um modelo abstrato e formal usado para descrever o fluxo de informação em sistemas, onde componentes dinâmicos são capturados por *tokens* que podem se mover entre estados, permitindo que comportamentos assíncronos e concorrentes sejam capturados [Haddad and Poitrenaud 1999]. Dado que os agentes são frequentemente solicitados a realizar várias tarefas simultaneamente, as redes de Petri são uma ferramenta particularmente eficaz para especificar e testar o comportamento de sistemas baseados em agentes. Além disso, as redes de Petri são um dos formalismos mais confiáveis para simulação e análise de modelagem comportamental e análise de falha de SMA [Boucherit et al. 2020] e são capazes de integrar os diferentes níveis e tipos da especificação $\mathcal{M}$oise$^+$, como resultado de seus mecanismos de abstração e refinamento hierárquico.

Por ser um trabalho em andamento, serão apresentadas algumas partes da implementação já realizadas e também os próximos passos da realização do trabalho. Acredita-se que esse um trabalho promissor, que tem a capacidade de ajudar com a forma como as informações são trabalhadas entre estas duas abordagens: $\mathcal{M}$oise$^+$ e CPN Tools. Desta forma, será mais simples tanto para quem for começar a utilizá-las, quanto para quem já a utiliza, de forma que sua criação seja mais simplificada e tenha-se uma melhor fluidez quando nos referimos a criação de novas redes de Petri.

2. Fundamentação Teórica

Existem atualmente diferentes modelos de organização para SMA, um deles é o $\mathcal{M}$oise$^+$, que implementa um modelo de programação para a dimensão organizacional de um sistema multiagente. Essa abordagem inclui uma linguagem para a organização do sistema e especificação de sua infraestrutura, com suporte para mecanismos de raciocínio baseados na organização no nível do agente. Esse modelo contém uma Especificação Estrutural (EE) e uma Especificação Funcional (EF) independentes, que estão ligadas por uma Especificação Deôntica (ED) [Hübner and Sichman 2007].

As Redes de Petri Coloridas (RPC) combinam as capacidades de uma rede de Petri ordinária com as capacidades de uma linguagem de programação. Nas RPC, as fichas possuem um valor de dados anexado que são chamados de cores. Essas cores podem representar diferentes processos ou recursos de uma rede, reduzindo assim o tamanho dos modelos. Para um determinado lugar, todas as fichas devem ter cores que pertençam a um tipo específico que é chamado de conjunto de cores do lugar. Esse uso de conjunto de cores é equivalente aos tipos de dados nas linguagens de programação (inteiro, caractere, booleano) e a especificação dos conjuntos de cores e das variáveis da rede é feita por declarações [Jensen 1997].

Em [Gonçalves et al. 2022] uma formalização de mapeamento para o Modelo $\mathcal{M}$oise$^+$ em RPC, denominado CPN4M foi apresentado. Nesse trabalho, diferentes elementos do $\mathcal{M}$oise$^+$ se transformam em estruturas para a RPC. Por exemplo, metas são representados pelos lugares; os papéis e grupos são relacionados através das cores; também foram levados em consideração os diferentes operadores de planos que podem ser uma sequência, uma escolha ou um paralelismo.

3. Arquitetura Desenvolvida

A Figura 1 mostra como será realizada essa automatização. Primeiramente, é realizada a leitura do arquivo XML do $\mathcal{M}$oise$^+$, utilizando um *Parser*, que é um componente de software que recebe dados de entrada em forma de texto e constrói uma estrutura de dados a partir dele.

A ferramenta, que utiliza a linguagem de programação *Python*, então identifica os elementos do $\mathcal{M}$oise$^+$, como os papéis e grupos que pertencem a parte da EE e as metas e operadores de planos usados na EF. A partir destes elementos, é possível gerar um novo arquivo que possui sua estrutura toda, também escrito em XML, porém sua extensão tem a sigla CPN, pois ele será lido pela ferramenta CPN Tools.

Para que esse arquivo funcione adequadamente, ele precisa conter os elementos da RPC. Portanto, devem ser criadas funções para gerar as declarações, os lugares, as transições e os arcos. Terminada essa etapa, o arquivo CPN é gerado e estará pronto para ser lido pelo CPN Tools.

O código começa com a importação do parser ao qual se irá utilizar, no caso dessas partes iniciais, só se fez necessário a utilização do XML ElementTree, que será usado em cada elemento das funções, como mostrado na Figura 2.

Em seguida parte-se para o desenvolvimento do lugar, que seria o equivalente a uma nova janela do CPN Tools, como mostrado na Figura 3 com o nome de wtitle e do

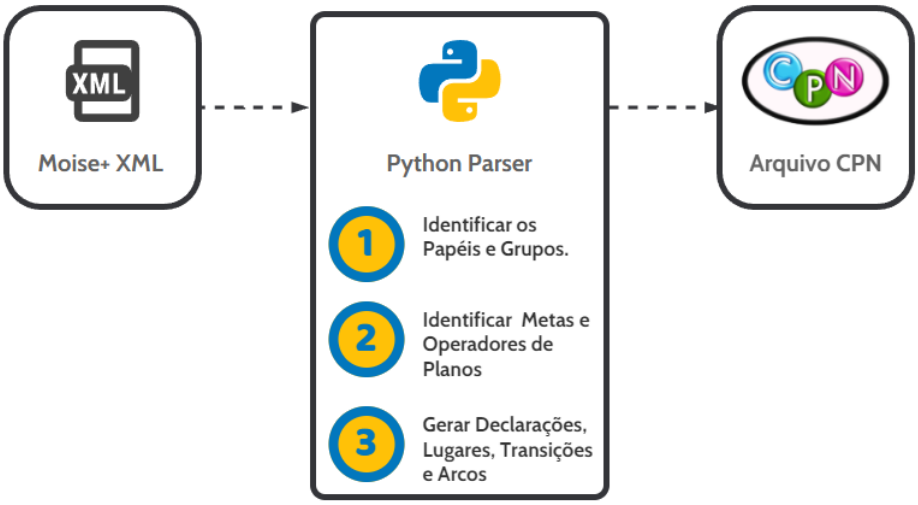


Figura 1. Visão Geral das Etapas de Automatização do Mapeamento

```
def gera_lugar(name, type):
  p1 = ET.SubElement(b37, "place")
  p1.set('id', 'ID1415407980')
  p2 = ET.SubElement(p1, "posattr")
  p2.set('x', '0.000000')
  p2.set('y', '-53.000000')
  p3 = ET.SubElement(p1, "fillattr")
  p3.set('colour', 'White')
  p3.set('pattern', '')
  p3.set('filled', 'false')
  p4 = ET.SubElement(p1, "lineattr")
  p4.set('colour', 'Black')
  p4.set('thick', '1')
  p4.set('type', 'solid')
  p5 = ET.SubElement(p1, "textattr")
  p5.set('colour', 'Black')
  p5.set('bold', 'false')
  p6 = ET.SubElement(p1, "text")
  p6.text = name
```

Figura 2. Exemplo da Criação de Lugar no Código de Arquitetura Desenvolvida

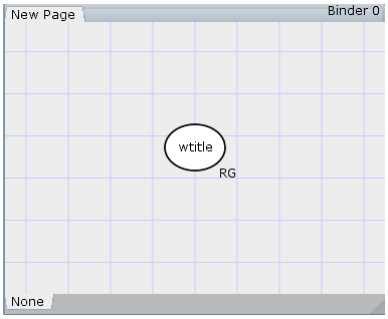


Figura 3. Exemplo da Criação de Lugar no CPN Tools

tipo RG. Por fim, tem-se o código que é gerado a partir desse algoritmo que gera o lugar, assim como está sendo mostrado na Figura 4.

4. Conclusões

Este trabalho apresentou uma abordagem para a criação de uma ferramenta que automatiza mapeamento organizacional de um SMA para uma RPC. Mais especificamente um modelo organizacional chamado $\mathcal{M}$oise$^+$ é convertido em elementos de RPC através de um método já descrito em trabalho anterior [Gonçalves et al. 2022]. Uma das possibilidades de utilização dessa ferramenta seria em ambientes de teste já que o modelo em RPC pode servir para para gerar os caminhos de teste necessários para a execução de testes.

Como próximos passos estão a conclusão desta ferramenta fazendo com que ela receba o arquivo XML do $\mathcal{M}$oise$^+$ como entrada, identifique os papéis, metas, a sequência com que as metas são realizadas, as relações deônticas entre as missões e os papéis, e as cardinalidades. Com essas informações já será possível gerar o XML da RPC completamente estruturada de acordo com a especificações $\mathcal{M}$oise$^+$ enviadas, utilizando as declarações, os lugares, transições e arcos da RPC.

Referências

Boucherit, A., Castro, L. M., Khababa, A., and Hasan, O. (2020). Petri net and rewriting logic based formal analysis of multi-agent based safety-critical systems. *Multiagent and Grid Systems*, 16(1):47–66.

Gonçalves, E. M. N., Machado, R. A., Rodrigues, B. C., and Adamatti, D. (2022). Cpn4m: Testing multi-agent systems under organizational model m oise+ using colored petri nets. *Applied Sciences*, 12(12):5857.

Haddad, S. and Poitrenaud, D. (1999). Theoretical aspects of recursive petri nets. In *International Conference on Application and Theory of Petri Nets*, pages 228–247. Springer.

Hübner, J. and Sichman, J. (2007). Developing organised multi-agent systems using the $Moise^+$ model: Programming issues at the system and agent levels. In *Int. J. Accounting, Auditing and Performance Evaluation*, pages 1–10.

Jensen, K. (1997). *Coloured Petri nets: basic concepts, analysis methods and practical use*, volume 1. Springer Science & Business Media.

```
<page id="ID6">
  <pageattr name="New Page" />
  <place id="ID1415407980">
    <posattr x="0.000000" y="-53.000000" />
    <fillattr colour="White" pattern="" filled="false" />
    <lineattr colour="Black" thick="1" type="solid" />
    <textattr colour="Black" bold="false" />
    <text>wtitle</text>
    <ellipse w="60.000000" h="46.000000" />
    <token x="-10.000000" y="0.000000" />
    <marking x="0.000000" y="0.000000" hidden="true">
      <snap snap_id="0" anchor.horizontal="0" anchor.vertical="0" />
    </marking>
    <type id="ID1415409755">
      <posattr x="32.500000" y="-79.000000" />
      <fillattr colour="White" pattern="Solid" filled="false" />
      <lineattr colour="Black" thick="0" type="Solid" />
      <textattr colour="Black" bold="false" />
      <text tool="CPN Tools" version="4.0.1">RG</text>
    </type>
    <initmark id="ID1415409615">
      <posattr x="56.000000" y="-26.000000" />
      <fillattr colour="White" pattern="Solid" filled="false" />
      <lineattr colour="Black" thick="0" type="Solid" />
      <textattr colour="Black" bold="false" />
      <text tool="CPN Tools" version="4.0.1" />
    </initmark>
  </place>
</page>
```

Figura 4. Exemplo da Criação de Lugar no Arquivo Gerado

MASPY: Towards the Creation of BDI Multi-Agent Systems

Alexandre L. L. Mellado[1], **Igor Guilherme Fidler**[1],
André Pinz Borges[1], **Gleifer Vaz Alves**[1]

[1]Departamento Acadêmico de Informática
Universidade Tecnológica Federal do Paraná (UTFPR)
Ponta Grossa – PR – Brasil

{mellado, fidler}@alunos.utfpr.edu.br

{apborges, gleifer}@utfpr.edu.br

Abstract. *Integrating intelligent agents is essential to the design and functionality of numerous modern computing solutions. Several industries and research domains, from health to finance, manufacturing to customer service, are influenced by advances in the area of Intelligent Agents. Therefore, this work presents a Python library for creating systems composed of intelligent agents following the Belief, Desire and Intention paradigm. To develop systems following these characteristics, four base classes were designed. These classes, agent, environment, communication and handler, create and manage the structural part, leaving the design and specific functions to the programmer. This paper shows that, to the best of our knowledge, there is no other library in Python with the same features and functionalities as the one described here.*

1. Introduction

Multi-Agent Systems (MAS) have gained prominence in various domains [Dorri et al. 2018], including Artificial Intelligence, Robotics and Social Sciences. With the increase in the complexity and scale of such systems, there is always a need for dedicated tools to streamline the development process. By providing a set of pre-built components, methods, and communication protocols, a library enables developers to focus on higher-level system design and behaviour aspects.

Incorporation of the concept of Belief-Desire-Intention (BDI) [Bratman 1987], further adds to the benefits of one such library. This widely recognized [Georgeff et al. 1999, Abar et al. 2017] paradigm is a classic theoretical framework used mainly to represent knowledge when developing agents and autonomous systems. The BDI architecture allows agents to represent and model their decision-making and behaviour using their beliefs, desires, and intentions [Bratman 1987].

This paper presents a Python library designed to simplify the development of MAS with BDI agents, our library is called **MASPY** (Multi-Agent System for PYthon). The motivation to create the MASPY is the need for a programmable agent system to add a reinforcement learning implementation for negotiations between agents. As Python is a mainstream language with several packages designed to quickly implement reinforcement learning techniques. Our idea was to find another library with the same characteristics.

Many tools, frameworks and libraries already offer the ability to program agents. This can be seen in [Kravari and Bassiliades 2015], [Pal et al. 2020] and

[Cardoso and Ferrando 2021], where, combined, over thirty platforms for agent development were compared. Even though their list was not exhaustive, it shows a significant number of platforms with *Java* as the target language, while only a few utilizing *Python*, and none using the latter implementing the BDI paradigm with multi-agents.

MAPSY includes new specific functionalities and heavy focus on the versatility of the design and capabilities of the BDI-MAS. The other main objective for creating this library, specifically in Python, is the use of learning methods in developing agents using old and new libraries for machine learning made for this language. It is inspired by functionalities defined in the JaCaMo Framework [Boissier et al. 2013], composed of three languages. Jason, to develop agents, Cartago, to implement environment artifacts, and Moise, to define the system organization. Some names of variables and structures are directly referenced to some in the Jason language. The main similarities are in managing beliefs, objectives and plans following the BDI paradigm and the directives for agent communication. This type of communication is referenced as KQML (Knowledge Query and Manipulation Language), which uses performatives and directives to support information sharing between agents. Another concept JaCaMo (Moise) inspired is using roles in the environment. While they are not close to the level of organization found in Moise, the role functions as a way to define what an agent can see and do in an environment. Section 3 provides more detail on these implementations.

2. Comparison Between Programmable Agents in Python

As described in [Kravari and Bassiliades 2015], [Pal et al. 2020] and more recently in [Cardoso and Ferrando 2021] there are few frameworks developed in *Python* for the development of agents. The existent tools are *PADE* [Melo et al. 2019] and *SPADE* [Palanca et al. 2020] for behaviour agents, *MESA* [Masad and Kazil 2015] for simulation, the ethical robot from [Bremner et al. 2019] and *PROFETA* from [Fichera et al. 2017] where they implement BDI agents. As follows, we describe each one of the agent tools designed with Python.

2.1. *PADE* framework

The PADE framework presented in [Melo et al. 2019] is used to develop behavioural agents distributed over nodes in a network. This framework contains base classes to create an agent and its behaviour. To manage the MAS, PADE uses an *Agent Management System* (AMS), the first agent initiated in the system. This agent manages a table with all identifiers of active agents. Agent communication is straightforward since each agent possesses knowledge of all the participants in the system using this table. Each message follows a FIPA protocol, and its contents can contain a serializable *Python* object.

The MASPY library has similarities with PADE framework: both support the creation of MAS, provide abstractions for managing and modelling agents. Their main difference is which paradigm is used: PADE uses behavioural agents while MASPY uses BDI agents. Also, PADE does not implement the environment in its code and only considers the sensors in a physical embedding as its environment.

2.2. SPADE

SPADE [Palanca et al. 2020] aims to be a general agent development middleware. Each agent has to register to SPADE using a unique identifier consisting of the agent name,

the server where it is running and a password. After that, the agent can create one or more behaviour to run. To allow communication between agents, SPADE provides a mechanism to dispatch messages to each registered agent which redirects an incoming message to the agent which can be waiting for it or relaying outgoing messages to the SPADE communication system.

Their communication system uses the eXtensible Messaging and Presence Protocol (XMPP) which is open and allows instant messaging and presence notification. It is always possible to know who is in the system and how to exchange messages with them. Using the XMPP as its communication protocol allows a SPADE agent to communicate with every XMPP server, making possible communication with other services or agents.

Although SPADE allows the execution of BDI agents, this is done using a plugin. While in MASPY, the agents are natively implemented as BDI. Besides, SPADE is ideal for systems focusing on the communication aspect or the ability to exchange information with external services. However, it does not contain an environment abstraction layer.

2.3. MESA

MESA [Masad and Kazil 2015] is a *Python* framework that allows simulation, visualization, and analysis of agent models inspired by NetLogo. It offers built-in core components such as spatial grids and agent schedulers. The different schedulers allow the control of the activation regime for each agent. The spatial components are used to model the environment where the agent is located. The agent model can act and change its current state based on its position, environment, and interaction with other agents. The communication between agents, while possible to be implemented, is not straightforward, as MESA does not offer communication utilities.

While it offers free control of created agents actions and ways to interact with a defined environment, this framework is strictly for simulation and data visualization purposes. Moreover, it does not contain an explicit communication layer necessary for a Multi-Agent System.

2.4. BDI Python

In [Bremner et al. 2019], the authors present how to embed ethical considerations into a BDI agent reasoning. This work is a proof of concept. Therefore, it does not provide a library or framework for managing MAS. Instead, it shows how a BDI agent could be implemented using *Python*. An agent's objective is to give a robot ethical reasoning in its actions. As it was only made to support a single agent, it can not be extended to a MAS and does not have a communication layer. MASPY has different goals and supports the creation of different agents while allowing them to communicate between themselves.

2.5. PROFETA

PROFETA (Python RObotic Framework for dEsigning sTrAtegies) [Fichera et al. 2017] is a programming framework designed for autonomous robots based on the BDI paradigm. It uses metaprogramming capabilities to incorporate the operational semantics of AgentSpeak into *Python*. This allows the implementation of object-oriented and declarative constructs and, therefore, the definition of an agent's behaviour more straightforwardly.

Comparing it to MASPY, it does support the integration of BDI reasoning, followed even by the introduction of new types of beliefs, but falls short of the need for a communication layer. PROFETA is a framework for the development of a Robot. Doing so involves only one agent. Also, the real environment perceived by this robot's sensors does not need to be, and was not, implemented in a separate programmable layer.

2.6. Comparative Agent Development in Python

For the MASPY library, the generic development of MAS with BDI Agents was the main characteristic when being designed. Table 3 shows the base components used for a MAS and compares all the presented tools. In MASPY, agents manage beliefs and objectives to execute plans, following the BDI paradigm. These agents can be situated in generically created environments and communicate using performatives closely following the KQML protocol used in JaCaMo.

As shown in this section, the goal of MASPY is to be able to build all of this in a single library. While other tools offer MAS creation capabilities, none contains all layers wanted. PADE and SPADE fall short on the environment and the BDI agent. MESA is focused too much on simulation. At the same time, BDIPython and PROFETA are made with only one agent in mind and not a multi-agent system.

Table 1. Tools for agent development in Python

Name	Description	Inspiration	Agent	Environment	Communication
MASPY	**BDI Multi-Agent System**	**JaCaMo Framework**	**BDI Agent**	**General Class Abstraction**	**Inspired by Speech-Based KQML**
PADE	**Distributed Multi-Agent Network**	**JADE Framework**	**Agent Behaviours**	**Physical Embedding**	**FIPA-ACL Messages**
SPADE	**Multi-Agent Plataform**	**XMPP Instant Messaging**	**Agent Behaviours**	**External Simulation Tool**	**XMPP Server**
MESA	**Agent-Based Simulation**	**NetLogo, MASON and Repast**	**Agent Model**	**Spatial Model Interaction**	**External Messaging Tool**
BDIPython	**Ethical BDI Agent**	**Autonomous Ethical Robot**	**BDI Agent**	**Physical Robot Sensors**	**Does Not Support**
PROFETA	**Autonomous BDI Agent**	**AgentSpeak Language**	**BDI Agent**	**Physical Robot Sensors**	**Does Not Support**

3. The MASPY Library

MASPY library aims to facilitate the development of a BDI-MAS. This section presents the library classes and how they work to allow this implementation. Agents represent entities with beliefs about their circumstances, desires or goals implemented as objectives they wish to achieve, and intentions in the form of plans that guides their action towards those goals. The environment class models its namesake, simulating an agent acting with its surroundings through actions and changing facts of a non-autonomous entity. And the communication class is used to open a route for information to travel between agents and form an interlocked system. While still necessary, the handler class that exists to help the programmer configure the order and connections for agents is entirely optional. This library was created by providing an abstraction of base classes and methods to enable programmers to design and implement a MAS quickly.

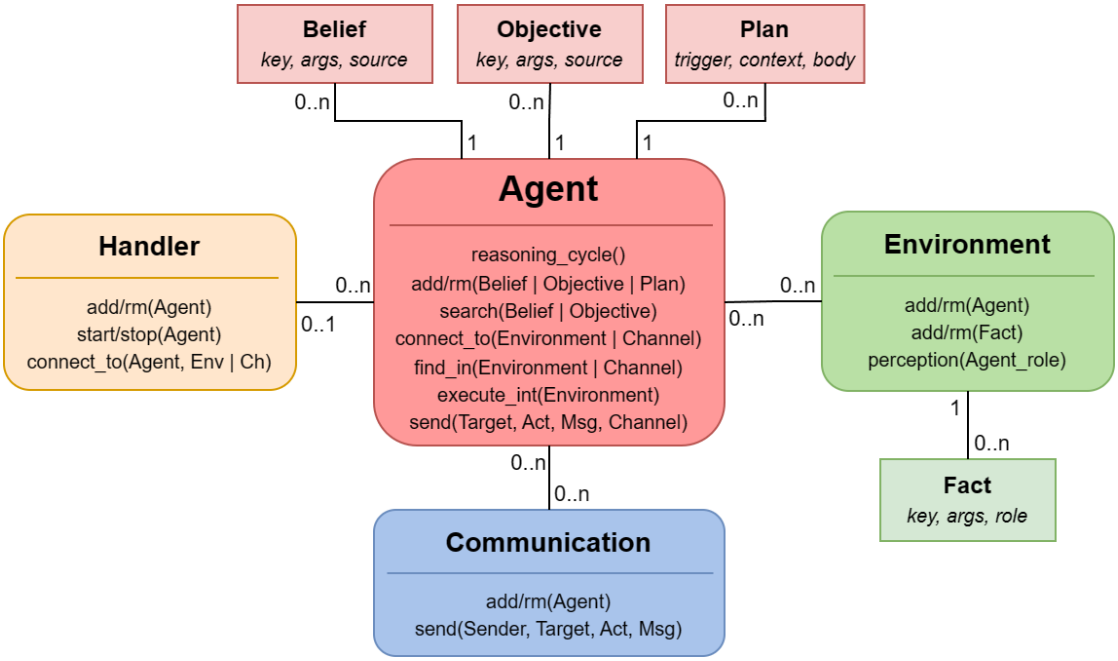


Figure 1. Diagram for the MASPY library

In practice, the system developer can use and extend the library components presented in Fig. 1. It allows the definition and initialization of agent objects, which can contain any number of beliefs, objectives and plans. These can interact with several environments and communicate through different instances of channels. Such agents and their connections to environments and channels can be configured by a single class that gives and saves distinct names for everyone, as described next.

3.1. Agent Class

The agent class is the fundamental building block of this library and the only strictly needed class to run a program. It contains the abstractions for managing an individual agent's beliefs, objectives and plans. It also has the methods to execute actions with any environment, as shown in subsection 3.2, and knows the protocol for sending and receiving messages (see subsection 3.3). The agent methods are used by being extended in classes created by the programmer. In code 1, it is presented the creation of an agent instance and the structure of methods to add a belief, objective and plan.

```
from maspy.agent import Agent
class Sample(Agent):
    def __init__(self, agent_name):
        super().__init__(agent_name)
# For removing, 'add' is just changed to 'rm'
        self.add(belief, key, arguments, source)
        self.add(objective, key, arguments, source)
        self.add_plan([(trigger, [context], Sample.body)])
# Every plan body must contain at least self and src arguments
    def body(self, src, *args, **kwargs):
```

Code Listing 1. Instance of an Agent

Beliefs, Objectives and Plans: Each agent may have any number of beliefs, objectives or plans. The structure of beliefs and objectives are similar: both consist of a key, a variable number of arguments to store any data and the source from which this belief or objective was created. A plan, however, is formed by its trigger, the context for its activation and a body, which can be seen as an ordinary function or method in *Python*. This context can consist of any number of beliefs or objectives that the agent must have to execute the plan. The agent class provides methods for the creation and removal of each.

Reasoning Cycle: Information from the environment and communication classes can be used to update the agent's beliefs and objective bases during the reasoning cycle. These objectives depict the agent's desires, and when combined with their beliefs, create intentions that activate plans. The environment affects this through the agent's perception and communication by exchanging messages between agents. This process is presented in Fig. 2 and in Code 2. In the current implementation, each agent only considers one intention per cycle. This intention, being achieved by a plan, is executed until their end without reconsideration to new beliefs or objectives gained during it.

```
def reasoning_cycle(self, stop_flag):
    while not stop_flag.is_set():
        self._perception()
        self._mail()
        chosen_plan, trigger = self._deliberation()
        if chosen_plan is not None:
            self._execution(chosen_plan, trigger)
```

Code Listing 2. Reasoning Cycle

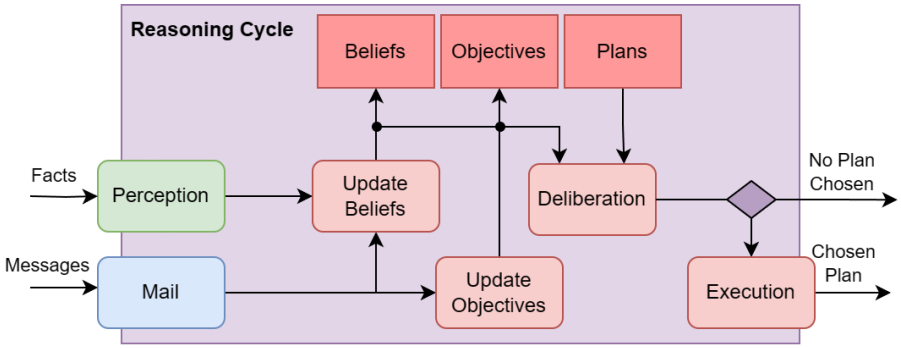


Figure 2. Reasoning Cycle

Each cycle starts by the agent perceiving all of its environments and polling all messages received from other agents, this will update its beliefs and objectives as necessary. Next, the agent deliberates which plan needs to be executed based on the available objectives and new beliefs acquired in this last cycle. In code, the reasoning cycle of each agent is started as a separate thread and is running until stopped or closing the system. This cycle can be stopped after every iteration if the *stop_flag* is set in another method.

3.2. Environment Class

The environment class represents a non-autonomous entity which agents can act upon. It can be used to model where the agent is situated, such as a street or a room. This class

provides the mechanisms for agents to perceive and act upon the environment, which uses facts to model the current state, updated by available actions in this environment. A fact consists of its key, a variable number of arguments, and the role field.

Facts and Roles: For the environment, facts work like beliefs do for agents. The main difference is how and which agents can change these facts. The role field, in fact, states that only connected agents with that designated role have access to that fact. The agent's role in the environment is defined when they are first connected, but can be changed. When perceiving the environment, the agent-assigned role is compared to the one in facts to determine which ones can be transmitted to the agent, who converts these facts to beliefs after being perceived, with its source being the environment it came from.

```
from maspy.environment import Environment
# Sample of an Environment with fact methods an action
class Sample_Env(Environment):
    def __init__(self, env_name="env"):
        super().__init__(env_name)
        self.create_fact(key, args, role)
        self.update_fact(key, new_args, new_role)
        self.extend_fact(key, added_args, added_role)
        self.shorten_fact(key, removed_args, removed_role)
# An environment action checking the agent role for execution
    def action(self, src):
        if not self.check_role(src, action_role):
            return None
```

Code Listing 3. Instance of an Environment

When trying to execute one of the environment actions, a role can be used to check if the agent can continue. Otherwise, agents without permission may still do other actions exposed by the environment that can indirectly affect those unavailable to perceive facts. The Code 4 show an example containing a sample agent connected to two different environments. In one, it enters as an "Observer", while in the other, it is connected as a "Manager". Without these roles, the presented facts, "Scoreboard" and "Quantity", would not be visible during the agent perception phase of its reasoning cycle.

```
ag = Sample("Ag")
env1 = Sample_Env()
env2 = Sample_Env("Warehouse")
env1.create_fact("Scoreboard",(10,50), "Observer")
env2.create_fact("Quantity",{"A": 3, "B": 7}, "Manager")
ag.connect_to(Sample_Env(),"Observer")
ag.connect_to(Sample_Env("Warehouse"), "Manager")
```

Code Listing 4. Examples of fact creation with roles

3.3. Communication Class

Our library enables the exchange of messages between agents by implementing a communication class. This class contains methods for connecting the message from the sender to the receiver. By default, an agent does not wait for a response after sending a message. An answer is only expected after asking for information from another agent.

Channels: In a classical agents' communication system, only one route for messages exists. Differently in this library, each instance of a communication class with a different name is a distinct channel in which connected agents can communicate with other connected agents. There is still an implicit way of using the classical method. Agents can connect to an unnamed channel that all other agents known by default.

The creation of channels came because of two reasons: first, the ease of implementation. In python, when defining a class for communication, different instances can be independent and still function, making a clear way to abstract different routes for messages; second, just as a simulation can contain multiple environments for agents to act upon, we considered it important for communication groups to be creatable.

```
1 from maspy.communication import Channel
2 ch1 = Channel() # Default Channel
3 ch2 = Channel("Crossroads") # Specific Channel
4 ch3 = Channel("Private") # Another Distinct Channel
```

Code Listing 5. Instances of Channels

Message Protocol: Each message has five parameters, of which four are required: the sender agent; the target agent; the type (or act) of the message; the content of the message; and an optional parameter, the channel used to send the message. Both sender and receiver have methods accounting for the different types of messages. These types always involve a transference of beliefs or objectives.

Directives: The type of message is defined as the directive of this message. It is mainly divided by way of exchanging beliefs or objectives. Agents can inform, request or ask for the contents of other agents. In code 6, three examples are presented. On line 1, a belief is sent to agent Ag1 through the "Private" channel. On line 2, a request is sent to Ag2 through the "Crossroads" channel. On line 3, an agent asks Ag3 for a belief using the default channel. On line 4, a broadcast is made, sending the belief "begin" to every agent in each channel contained in the channels list. And line 5 shows how a plan can be sent to multiple agents, by using a list, using the default channel.

```
1 self.send(Ag1,"tell",("price_interval",[10, 20]), Channel("Private"))
2 self.send(Ag2,"achieve",("cross","right"), Channel("Crossroads"))
3 self.send(Ag3, "ask", ("channel_name",))
4 self.send("Broadcast", "tell", ("begin",), Channel_List)
5 self.send(Ag_List, "tellHow", self.plan)
```

Code Listing 6. Examples of sent messages

3.4. Handler Class

The handler is responsible for assigning unique identifiers to each agent. It can be used to determine the timing and order of initialization for the agent's reasoning cycle. It can also more intuitively connect multiple agents to multiple communication channels and environments. Only one instance of this class will be active throughout the execution of the system; however, it is not a centralizing point. All of its methods exist only for the ease of configuring the created system.

Multi-Agent System Configuration: The agent class contains the necessary methods for beginning and stopping their reasoning cycle and ways for connecting with the environment and communication channels. The job of this handler class is for a more straightforward configuration step. When multiple agents must communicate, distinct names are necessary to avoid ambiguities. The programmer can, and sometimes should, create agents with different names for better distinction, but all instances of agents in this library are given an identifier used to differentiate between agents with the same name.

The codes 7 and 8 show the difference when using the handler. In this example, 50 Sample agents are created and started after being connected to the Communication Channel "Private". While listing 7 gives a number to the instance of Sample in line 3. The handler already gives a different ID to each agent in line 1 of Code 8.

```
1 agent_list = []
2 for i in range(50):
3     ag = Sample(f"Ag{i}")
4     ag.connect_to(Channel("Private"))
5     agent_list.append(ag)
6 for ag in agent_list:
7     ag.reasoning_cycle()
```

Code Listing 7. Explicit Configuration

```
1 agent_list = Handler().create_agents(50, Sample("Ag"))
2 Handler().connect_to(agent_list, [Channel("Private")])
3 Handler().start_all_agents()
```

Code Listing 8. Handler Configuration

4. MASPY in Practice

This section shows two practical implementations as examples of how this library works. The first one highlights a simple message exchange, using context to decide between sending and receiving the message. The second example shows an interaction by an agent executing an action in an environment.

4.1. Message Exchange Example

In code 9, two instances of the same Sample Agent are created to present a message being sent between instances. The first instance is given the name "*Sender*" and adds the belief "*Sender*" along with the objective "*send_info*". The second instance is the "*Receiver*" being given the belief "*Receiver*". After creating both instances of the Sample Agent, they are connected to the default Channel and their reasoning cycle is started by the Handler.

The execution goes as follows: The Agent "*Sender*" has the "*send_info*" objective and so triggers the available plan during deliberation. For this plan to execute, it checks if the agent contains the belief "*Sender*" that it has. In this plan, first, the "*Receiver*" is located in the channel's list of agents. Then all are sent the objective to "*receive_info*". The "*Receiver*" chooses to execute the plan "*recv_info*" triggered by this new objective. This plan also only is chosen if the agent has the belief "*Receiver*" defined in its context. In this plan, the message is displayed along with the sender.

```python
from maspy.agent import Agent
from maspy.communication import Channel
from maspy.handler import Handler

class Sample(Agent):
    def __init__(self, agent_name):
        super().__init__(agent_name)
        self.add_plan([
            ("send_info",[("belief","Sender")],Sample.send_info),
            ("receive_info",[("blf","Receiver")],Sample.recv_info)
        ])
    def send_info(self, src, msg):
        agents_list = self.find_in("Sample","Channel")["Receiver"]
        for agent in agents_list:
            self.send(agent,"achieve",("receive_info",msg))
    def recv_info(self, src, msg):
        self.print(f"Information [{msg}] - Received from {src}")

if __name__ == "__main__":
    sender = Sample("Sender")
    sender.add("blf","Sender")
    sender.add("obj","send_info",("Hello",))
    receiver = Sample("Receiver")
    receiver.add("belief","Receiver")
    Handler().connect_to([sender,receiver],[Channel()])
    Handler().start_all_agents()
```

Code Listing 9. Sending and Receiving a Message

4.2. Environment Interaction Example

Code 10 shows an example with most of the available functions in the MASPY library. It consists of a crossroads environment managed by an agent that chooses when another, the vehicle, can cross by communication in a private channel. Three classes were created. The environment Crossing has a fact for the traffic light indicating that it is "*Green*" and only agents with the role "*Manager*" can perceive it. This Crossing also has an action to cross it, which shows the agent that executed the action. This example has two agents, the Cross Manager who checks the traffic light and the Vehicle that crosses the junction.

During the configuration phase beginning in line 32, each of the wanted components is initiated before the agents' reasoning cycle starts. First, the channel "*Crossing*" and the environment "*Cross_Junction*" are instantiated. Second, both agents are given the exact name of their class. And finally, both are connected to the channel and environment, followed by starting the reasoning cycle using the handler.

After all connections and the handler starting the agents, the *Cross_Manager* is the first to act. It begins its plan *traffic_light* after perceiving the fact with the same name in the environment "Crossing" because it has the role of "*Manager*". In this plan, the *Cross_Manager* sends an objective for *crossing_over* to all connected vehicles in the "*Crossing*" channel. The Vehicle connected starts its plan to cross after receiving this objective from the *Cross_Manager* and executes an action in the environment to cross it. The *Cross_Junction* then shows the agent executing its action.

```python
from maspy.agent import Agent
from maspy.environment import Environment
from maspy.communication import Channel
from maspy.handler import Handler

class Crossing(Environment):
    def __init__(self, env_name):
        super().__init__(env_name)
        self.create_fact("traffic_light","Green","Manager")
    def cross(self, src):
        self.print(f"Agent {src.my_name} is now crossing")

class Cross_Manager(Agent):
    def __init__(self, mg_name):
        super().__init__(mg_name)
        self.add_plan([("traffic_light",[],Cross_Manager.trf_light)])
    def trf_light(self, src, color):
        vehicles = self.find_in("Vehicle","Env","Cross_Junction")
        for vehicle in vehicles["Vehicle"]:
            self.print(f"Detected traffic light: {color} in env {src}")
            self.print(f"Sending signal to {vehicle}")
            self.send(vehicle,"achieve",("crossing_over",),"Crossing")

class Vehicle(Agent):
    def __init__(self, vh_name):
        super().__init__(vh_name)
        self.add_plan([("crossing_over",[],Vehicle.crossing)])
    def crossing(self, src):
        self.print(f"Confirmation for crossing by {src}")
        self.execute_in("Cross_Junction").cross(self)

if __name__ == "__main__":
    cross_channel = Channel("Crossing")
    cross_env = Crossing("Cross_Junction")
    cross_manager = Cross_Manager("Cross_Manager")
    vehicle = Vehicle("Vehicle")
    Handler().connect_to([(cross_manager,"Manager"),vehicle],
                         [cross_channel,cross_env])
    Handler().start_all_agents()
```

Code Listing 10. Executing Action in Environment

5. Conclusion

This paper presented the MASPY library as a way to develop BDI MAS with Python. It describes its functionalities made possible by four base classes. The agent class manages their BDI components, Beliefs, Objectives and Plans, and the logic to perceive and communicate through environments and communication channels. The environment class models surrounding spaces for the agent to interact. The communication class contains functions to connect sent messages between agents. And the handler class can be used as a configuration tool to build and start your MAS more easily.

Other tools in Python were described and compared to MASPY. While they are more complete than our library, none offer all the wanted layers of abstraction as the

MASPY library. To present this, multiple implementation samples were shown to provide practical examples of this library's work. These varied between code snippets to the complete example with a real working system, albeit very simple. In these examples, the objective was to present a functional library with a very general disposition to multi-agent system development, with more room for such design.

In future work, we plan to define some expansions for the library. The implementation of more descriptive components to methods for context in plans and checking roles in actions. A warning system for better depuration of implemented code. Making the reasoning cycle more robust and adding the direct option for working with continuous and abstract execution. Finally, the introduction to machine learning, specifically reinforcement learning, in agent reasoning.

References

Abar, S., Theodoropoulos, G. K., Lemarinier, P., and O'Hare, G. M. (2017). Agent based modelling and simulation tools: A review of the state-of-art software. *Computer Science Review*, 24:13–33.

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.

Bratman, M. (1987). *Intention, Plans, and Practical Reason*. Cambridge: Cambridge, MA: Harvard University Press.

Bremner, P., Dennis, L. A., Fisher, M., and Winfield, A. F. (2019). On proactive, transparent, and verifiable ethical reasoning for robots. *Proceedings of the IEEE*, 107(3):541–561.

Cardoso, R. C. and Ferrando, A. (2021). A review of agent-based programming for multi-agent systems. *Computers*, 10(2):16.

Dorri, A., Kanhere, S. S., and Jurdak, R. (2018). Multi-agent systems: A survey. *IEEE Access*, 6:28573–28593.

Fichera, L., Messina, F., Pappalardo, G., and Santoro, C. (2017). A python framework for programming autonomous robots using a declarative approach. *Science of Computer Programming*, 139:36–55.

Georgeff, M., Pell, B., Pollack, M., Tambe, M., and Wooldridge, M. (1999). The belief-desire-intention model of agency. In *Intelligent Agents V: Agents Theories, Architectures, and Languages: 5th International Workshop, ATAL'98 Paris, France, July 4–7, 1998 Proceedings 5*, pages 1–10. Springer.

Kravari, K. and Bassiliades, N. (2015). A survey of agent platforms. *Journal of Artificial Societies and Social Simulation*, 18(1):11.

Masad, D. and Kazil, J. (2015). Mesa: an agent-based modeling framework. In *14th PYTHON in Science Conference*, volume 2015, pages 53–60. Citeseer.

Melo, L. S., Sampaio, R. F., Leão, R. P. S., Barroso, G. C., and Bezerra, J. R. (2019). Python-based multi-agent platform for application on power grids. *International transactions on electrical energy systems*, 29(6):e12012.

Pal, C.-V., Leon, F., Paprzycki, M., and Ganzha, M. (2020). A review of platforms for the development of agent systems. *arXiv preprint arXiv:2007.08961*.

Palanca, J., Terrasa, A., Julian, V., and Carrascosa, C. (2020). Spade 3: Supporting the new generation of multi-agent systems. *IEEE Access*, 8:182537–182549.

Benchmarking Scalability of Message Transport Systems in the JADE Platform: Experimental Evaluation and Performance Analysis

Luis Felipe Ferin Sgursky, Arthur Casals, Anarosa Alves Franco Brandao

[1]Escola Politécnica - Universidade de São Paulo (EPUSP)
Av. Prof. Luciano Gualberto, 158 - trav.3 - 05508-900 - São Paulo - SP - Brazil

Abstract. *Agents are distinct entities known for their independence and autonomy. In a multi-agent system, multiple agents can interact with each other. In this context, communication is a crucial component in enabling relationships between agents, and it is one of the fundamental features provided by a multi-agent system platform. As such, the performance of a multi-agent system can be directly affected by the implementation of its communication mechanism. In this paper, we analyze the scalability of the JADE platform from the perspective of its communication mechanism. We do this by defining benchmarks and evaluating the platform's response in different scale-up and scale-out scenarios.*

1. Introduction

Agents, as relatively independent and autonomous entities, are meant to solve problems of varying complexity [Ferber 1999]. There are different types of agents, ranging from reactive agents to those capable of intelligent reasoning, such as deliberative agents [Balke and Gilbert 2014]. A BDI agent is a particular type of deliberative agent that makes decisions similarly to humans [Thangarajah et al. 2002], based on beliefs, desires, and intentions. Other types of agents include hybrid agents, which combine the strengths of reactive and deliberative agents. With their overall interactive and social capabilities, agents serve as a paradigm for software engineering [Jennings 2000].

Multi-agent systems integrate software agents that collaborate to achieve goals, using different interaction mechanisms [Stone and Veloso 2000, Russell and Norvig 2016]. One of the purposes of a MAS is to autonomously solve complex problems that are challenging for monolithic systems. They reduce system complexity and coupling between components, thus being capable of adapting to unexpected conditions.

Software agents possess communication skills for data and message sharing [Russell and Norvig 2016]. Agents communicate among themselves through the use of agent communication languages (ACLs) [Jennings 2000, Bellifemine et al. 2001]. However, communication between MASs can face complexity and interoperability limitations [Poslad and Charlton 2001].

This paper presents an experimental study that aims to evaluate the scalability of the JADE platform, with a particular focus on its communication mechanism. The goal is to evaluate how the platform performance is affected when increasing communication components, such as the number of agents, the number of messages sent, and the size of the messages. By systematically varying these scenarios and measuring the related performance metrics, it is possible to analyze patterns that indicate characteristics of the platform's scalability.

This paper is structured as follows: Section 2 provides a brief overview of related work on scalability for MAS platforms. Section 3 presents the methodology chosen to perform the experiments in this paper. Section 4 details each experiment in depth. Section 5 presents the results of each experiment and its correlation with the scalability of the platform. In section 7 we discuss the results obtained from the experiments.

2. Related Work

In [Vitaglione et al. 2002], the authors evaluated the communication performance of the JADE platform by measuring the duration of a conversation between agents in two distinct situations: agents deployed into the same platform (intra-platform), and agents deployed into different platforms (inter-platform). Also, there were two distinct intra-platform scenarios: (i) agents deployed within the same container, and (ii) agents deployed into different containers. These tests, however, were focused on analyzing the communication implementation in JADE's communication middleware, attributing the results obtained to the higher level of abstraction required by ACL-compliant communication.

In [Such et al. 2007], the authors used a different methodology from [Vitaglione et al. 2002], using a fixed number of communication couples in each experiment (instead of gradually increasing their numbers). They proposed benchmark metrics to analyze the performance impact of scaling the communication between agents in different aspects, such as increasing the number of hosts, massive reception of messages from one host, and scaling the number of agents within a platform. Those experiments provide a higher level of abstraction because they evaluate communication aspects from MAS in general, instead of aiming for a specific platform. This allows different MAS platforms to be compared using the same methodology and metrics.

In another work, [Rodrigues 2019] provided a practical implementation and analysis between several MAS platforms such as JADE, SPADE, JIAC V, and ASTRA. More recently, [Alencar 2020] expanded this work using instances of the JADE platform deployed into the cloud, using on-demand Amazon Elastic Compute Cloud (Amazon EC2) services.

Our work extends the work done by [Rodrigues 2019] and [Alencar 2020] in terms of scalability. The communication metrics are the same as used for [Vitaglione et al. 2002], but the benchmarks chosen to evaluate the platform are the ones defined by [Such et al. 2007].

3. Context

3.1. Introduction

In distributed systems, scalability is an important aspect from the perspectives of performance, responsiveness, and reliability. Similarly, in a MAS, the system must be able to handle the communication between the agents efficiently, since increasing the number of agents in a system will also increase the interactions among them. As we mentioned before, our objective is to study the scalability of a MAS built using the JADE platform, focusing on its communication mechanism.

There are many different ways in which a system can be scaled. For the purposes of this work, we will focus on two different scaling approaches: *scaling up*, referring

to adding more resources (e.g. memory, processor) to a system already deployed, and *scaling out*, which is when we deploy more instances of a system (so it can handle a greater workload). For our study, we analyze how a MAS built using JADE responds to both approaches.

3.2. JADE

JADE (Java Agent Development Framework) [Bellifemine et al. 2007] is a decentralized FIPA-compliant, Java-based agent platform, used for the implementation of agents and MAS. The platform runs on the JVM (Java Virtual Machine) and hosts agents in containers, each being a Java process. The platform is capable of hosting containers within the same host or even hosts spread across the web. It also supports inter-platform communication, where agents deployed on different platforms can communicate with each other. To enable agent communication, JADE uses MTPs (Message Transfer Protocols) that allow communication between agents. The protocol and implementation used depend on the location of the agents [Bellifemine et al. 1999].

When dealing with intra-platform communication, if all agents are within the same container the communication is handled by IMTP (Internal Message Transfer Protocol). If the agents are located in different containers, the communication will be handled through RMI (Remote Method Invocation). In an inter-platform scenario, communication can be established via many different protocols. In our experiments, we used IMTP for intra-platform communication (since all agents were deployed in the same container) and HTTP MTP (MTP based on HTTP) for inter-platform communication.

3.3. Architecture and Environment

The MAS evaluated in this work consists of multiple JADE platforms running within a cluster. We use two different configurations for our experiments: when measuring benchmarks 1 to 3 (explained below), each JADE platform has only the main container, and all of its agents are registered under it. When measuring benchmark 4, each JADE platform has multiple containers. The details of each configuration are shown in the next section (Figures 2 and 3).

All infrastructure used in our experiments was cloud-based. We used AWS (Amazon Web Services) as our main cloud infrastructure provider. Each JADE platform was instantiated in a dedicated Docker [1] container, thus providing a high level of isolation between the deployed platforms. This setup can be seen in Figure 1

Using clusters to deploy the MAS was the approach we chose to overcome the limitations faced by [Alencar 2020]: the deployment environment was limited to 32 Amazon Elastic Compute Cloud (EC2) instances, and it was also necessary to manually deploy and setup each instance. By using a cluster with Amazon Elastic Container Service (ECS) as the orchestration mechanism, the limit of instances running goes from dozens to thousands, and the allocation of the JADE platforms within the docker containers is optimized across the cluster. This allows us not only to greatly increase the number of JADE hosts but also to better understand the communication constraints in the JADE platform within a highly scalable scenario.

[1] https://www.docker.com/

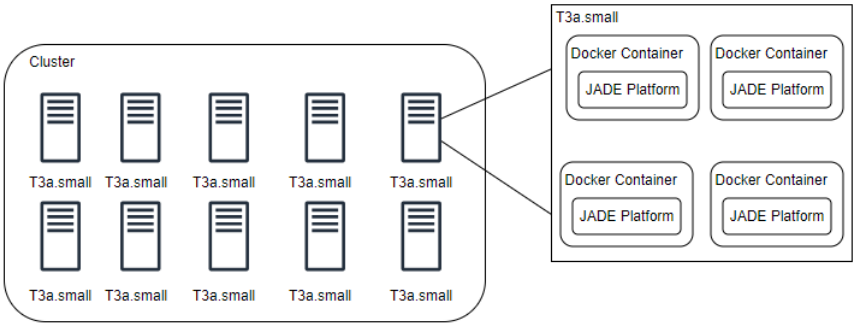


Figure 1. Cluster

The cluster is composed of multiple EC2 T3a.small instances with the default hardware configuration and networking settings [2]. The instances were created on demand by the orchestrator, and the resources used were kept during the experiments, thus maintaining the server performance stable across the experiments. By using the on-demand proposition on AWS, once the cluster is created, we guarantee that the created resources will be available on the cluster, removing the risk of impact on the test performance due to external factors (such as dynamic reallocation of resources by the cloud provider).

In this experiment, each cluster instance hosted multiple docker containers. Each of these containers was hard-limited to allocate a maximum 512MB of memory and 0,5 VCPU on the scale-out experiments (equivalent to 25% of the EC2 instance's hardware resources). For scale-up experiments, the allocation of Memory and VCPU per docker container varied.

Each agent can perform two different roles: sender and receiver. Sender agents are responsible for sending a message, waiting for its reply, and storing the round trip time. Receiver agents will wait for messages and reply to them. Communication can be established between agents in the same JADE platform (intra-platform) and agents in different platforms (inter-platform).

4. Experiments

The experiments were evaluated by measuring the communication performance between agents. The metric used to analyze the performance is the average round trip time (RTT) of the messages exchanged between two agents (Sender and Receiver). The Sender will start the communication and send a message to the Receiver, measuring the time it will take to receive a reply. The average RTT (avgRTT) is a well-known metric, commonly used on network communication-related benchmarking [Tanenbaum 2003]

4.1. Scale-Out

In the context of this paper, the scale-out approach refers to increasing the number of software instances running, in our case, JADE platforms. To evaluate the scalability of the platform using the scale-out approach, we adopted a set of four benchmarking scenarios (as defined by [Such et al. 2007]), which allowed the evaluation of different layers

[2]https://aws.amazon.com/ec2/instance-types/

of the platform communication mechanism. We will refer to these scenarios simply as ”benchmarks.” The first benchmark evaluates how the platform is affected as the multi-agent system grows and has more agents interacting with each other, providing a general response to the platform’s scalability.

The second benchmark involves concentrating all the messages being sent to a single receiver agent. In this context, we can evaluate how the platform handles the massive receive of messages from a single agent’s perspective. The third benchmark expands the previous benchmark by adding multiple receiver agents in the same platform. The total number of received messages is still the same, but as the messages are received by different agents, using both benchmarks allows us to evaluate the limitations of the platform when receiving multiple messages and how its performance is affected when multiple agents are responsible for handling the messages (in contrast to a single agent).

The fourth benchmark evaluates the platform response in dealing with inter-platform and intra-platform communication, and how the messages exchanged between agents in different platforms affect the average RTT.

In our experiments, all benchmarks are evaluated against the same parameter variations, the number of senders(NS), the number of messages(NM), and the message size(MS).

4.1.1. Scale-Out Benchmark 1: Number of Hosts

In this benchmark, each host platform contains one Sender and one Receiver. The Senders send messages to all Receivers allocated in different JADE platforms. In this scenario, the total number of exchange messages is NS*(NS-1)*NM. An example of this benchmark is shown in Figure 2.

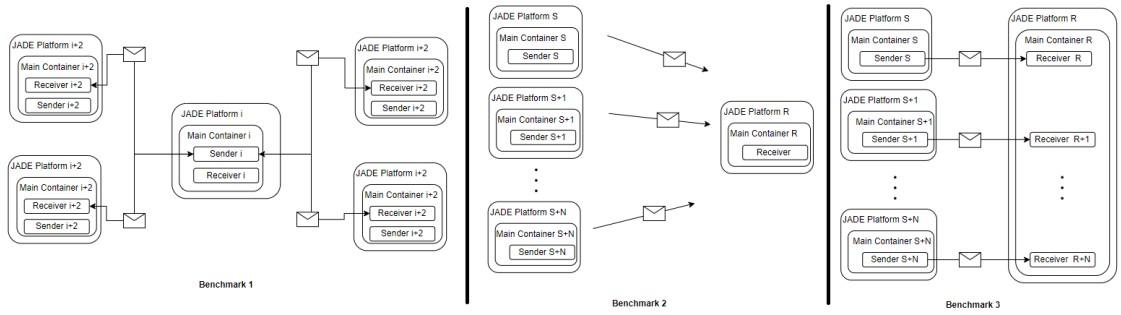


Figure 2. Benchmarks 1, 2 and 3

4.1.2. Scale-Out Benchmark 2: Massive Reception on Single Receiver

This benchmark analyzes how the platform behaves if a single Receiver is in charge of receiving and replying to all the messages originating from the Senders. For this benchmark, the total number of received messages by each receiver agent is NS*NM, as shown in Figure 2.

4.1.3. Scale-Out Benchmark 3: Massive Reception on Multiple Receivers

This benchmark complements the previous one by adding multiple Receivers to handle the messages. All messages are still sent to the same host, but now the workload to receive the messages is divided by several agents across the platform. The host still receives NS*NM messages, but every agent receives only NM messages since every Sender only sends messages to one Receiver agent during the experiment. A diagram of this flow is presented in Figure 2.

4.1.4. Scale-Out Benchmark 4: Number of Agents per Host

The last scale-out benchmark analyzes the platform performance when dealing with multiple agents exchanging messages on it. This benchmark analyzes the effect on the average RTT in two different communication scenarios: intra-platform and inter-platform. For intra-platform communication, all agents are placed on the same platform, and message exchanges follow the 1:1 relationship between sender and recipient, as shown in Figure 3.

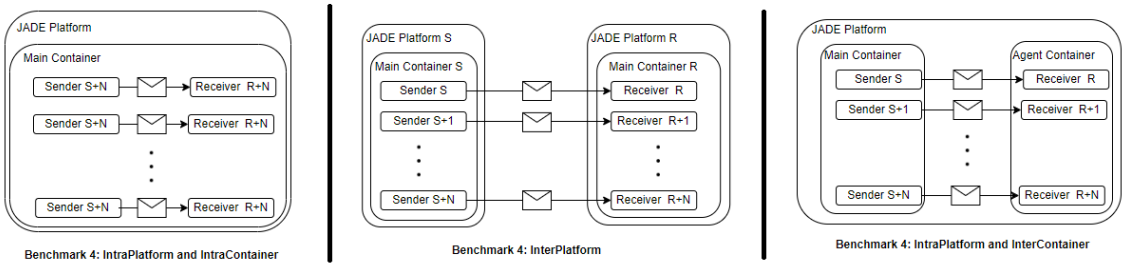


Figure 3. Benchmark 4

For inter-platform communication, the Senders are placed on one platform and the Receivers on another platform. The communication pattern is the same as the one used for intra-platform communication: the Senders only send messages to a unique Recipient, as shown in Figure 3. Also, as explored in [Vitaglione et al. 2002], JADE has two levels of abstraction where agents can be placed: the platform itself and containers, which also function as agent groups. This is a specificity of the JADE platform, which makes it necessary to evaluate the communication performance of agents (i) within the same container and (ii) in distinct containers.

For intra-container communication, the design of the experiment follows the same one as illustrated by ”Benchmark 4: IntraPlatform and IntraContainer”, in Figure 3. For communication between agents in different containers, the design of the experiment is illustrated by ”Benchmark 4: InterContainer”, also in Figure 3.

4.2. Scale-Up

To assess the platform’s scalability by employing the scale-up approach, we analyze how the variance in physical components such as CPU and Memory affect the average RTT performance. The chosen method in this experiment follows the design proposed in 4.1.1. This benchmark simulates a common behavior where each host has agents that behave actively and passively when it comes to initiating communication.

4.2.1. Scale-Up 1: CPU limit

For this assessment, we vary the number of CPU units that each JADE platform will have available, starting from 128 CPU units up to 2048 CPU units, while keeping the available memory at 2048MB.

4.2.2. Scale-Up 2: Memory limit

For this assessment, we vary the memory that each JADE platform will have available, starting from 128MB and going up to 2048MB. Similarly to the previous assessment, we maintain the CPU units available at 2048 units.

5. Results

5.1. Scale-Out Boxplot Results

5.1.1. Benchmark 1: Number of Hosts

For the first benchmark, it was possible to see a positive correlation when the number of hosts and the number of messages sent were increased. These results can be seen in Figure 4, but it shows no correlation to situations when the message size was increased.

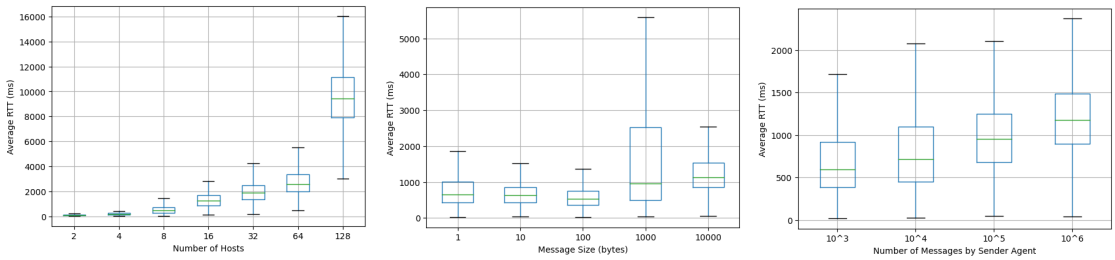


Figure 4. Scale-Out Benchmark 1 Results

5.1.2. Benchmark 2: Massive Reception on Single Receiver

For the second benchmark, it was not possible to observe any correlation between the increase of hosts and the increase of messages sent (Figure 5).

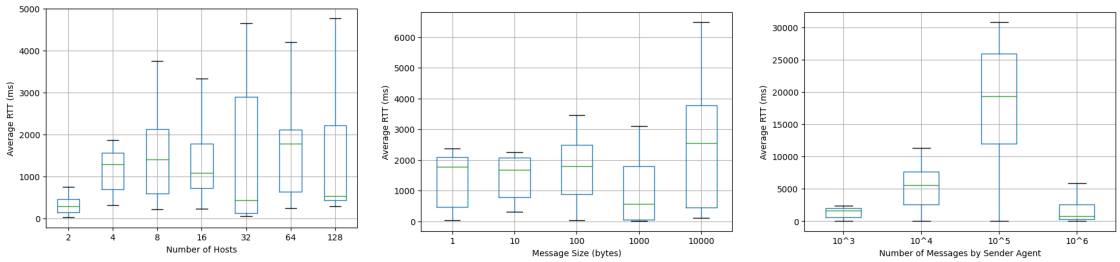


Figure 5. Scale-Out Benchmark 2 Results

5.1.3. Benchmark 3: Massive Reception on Multiple Receivers

This benchmark has shown the same pattern as seen in the 4.1.1. Increasing both the number of hosts in the system and the number of messages have a correlated impact on the average RTT, as shown in Figure 6. However, it was not possible to observe this pattern when varying the size of the messages.

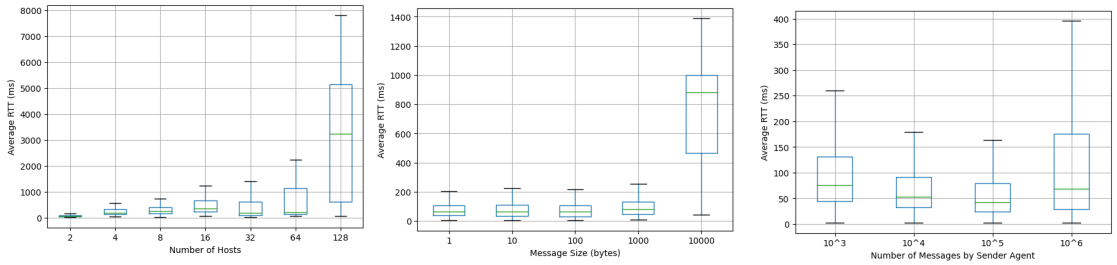


Figure 6. Scale-Out Benchmark 3 Results

5.1.4. Benchmark 4: Number of Agents per Host

For inter-platform communication, increasing the number of hosts and the number of messages sent also increased the average RTT, as can be seen in Figure 7. However, increasing the message size didn't provide any pattern that could be used to identify a relationship in the average RTT.

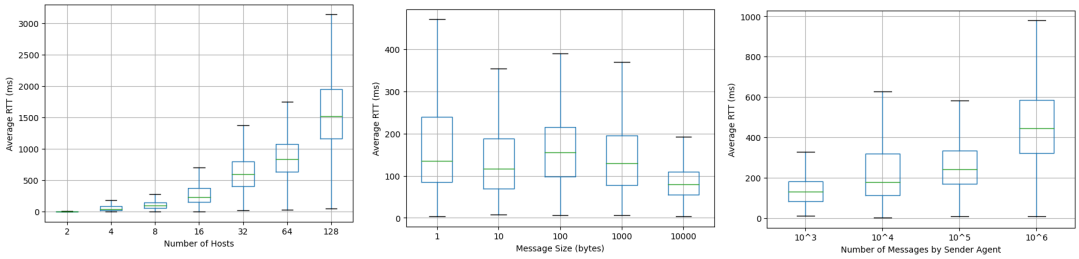


Figure 7. Scale-Out Benchmark 4 Results

When looking into intra-platform communication, the results differ. Changing the message size and the number of messages sent didn't result in a proportional variation in the average RTT (Figure 8), but increasing the number of hosts affected the metrics.

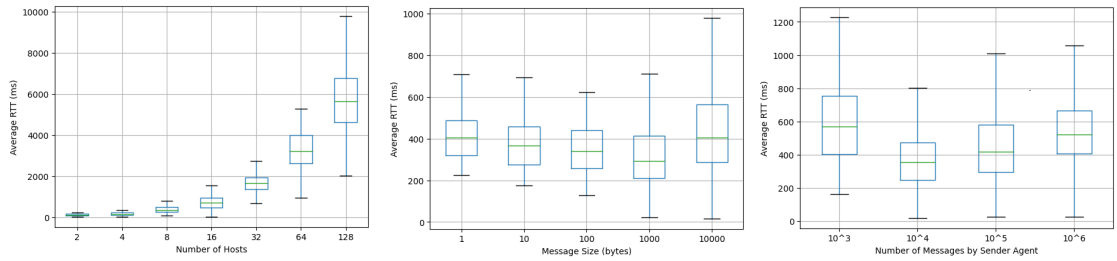


Figure 8. Scale-Out Benchmark 5 Results

The results for the experiments that stressed communication inter-container and intra-container presented a similar result. The correlation and possible limitation of the platform was presented when increasing (i) the number of agents in the experiment and (ii) the number of messages, as can be seen in Figures 9 (intra-container communication) and 10 (inter-container communication).

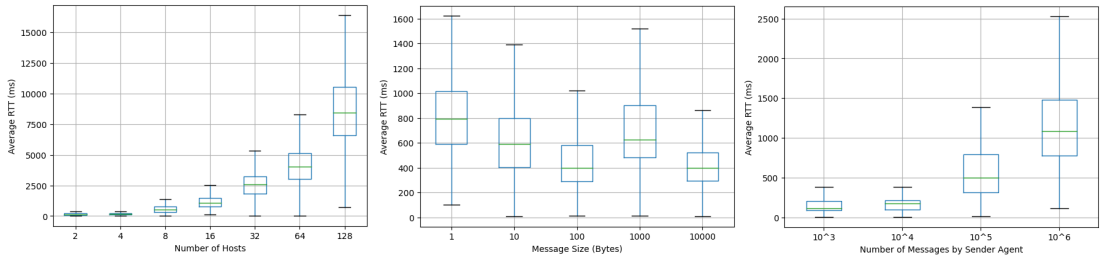


Figure 9. Scale-Out Benchmark 6 Results

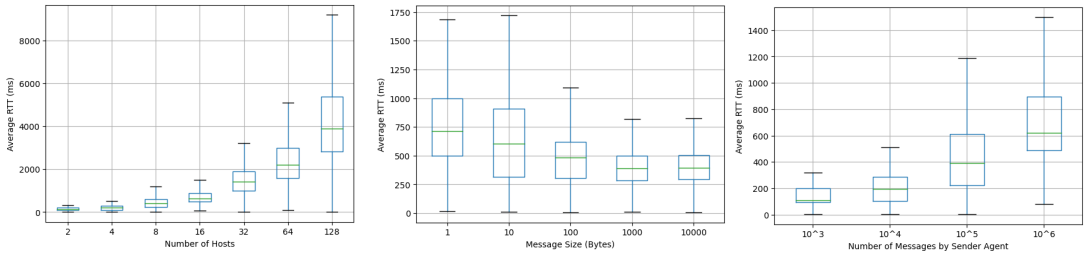


Figure 10. Scale-Out Benchmark 7 Results

5.2. T-test Variation Across Scale-Out Benchmarks

5.2.1. Number of Host Variation

The t-test results for the number of hosts variation indicate that all benchmarks have significant performance differences according to the increase in the number of hosts. The differences in the performance are substantial due to the large absolutes values observed in the t-statistics values, as can be seen in Table 1.

Table 1. Test T RTT Variation for Number of Hosts

Benchmark	Test T per Number of Hosts											
	2 - 4		4 - 8		8 - 16		16 - 32		32 - 64		64 - 128	
	Statistic	pValue	Statistic	pValue	Statistic	pValue	Statistic	pValue	Statistic	pValue	Statistic	pValue
Benchmark 1	-8.08	<0,05	-22.47	<0,05	-44.17	<0,05	-32.10	<0,05	-39.88	<0,05	-513.09	<0,05
Benchmark 2	-20.33	<0,05	-2.64	<0,05	-0.13	<0,05	- 0.003	<0,05	-1.34	<0,05	0.45	<0,05
Benchmark 3	-15.79	<0,05	-4.15	<0,05	-4.67	<0,05	0.70	<0,05	-1.64	<0,05	-10.11	<0,05
Benchmark 4	-69.35	<0,05	-44.66	<0,05	-96.90	<0,05	-151.75	<0,05	-112.40	<0,05	-259.08	<0,05
Benchmark 5	-4.23	<0,05	-9.17	<0,05	-6.61	<0,05	-18.18	<0,05	-25.50	<0,05	-20.84	<0,05
Benchmark 6	-9.09	<0,05	-93.56	<0,05	-92.22	<0,05	-164.05	<0,05	-149.99	<0,05	-394.66	<0,05
Benchmark 7	-17.88	<0,05	-52.85	<0,05	-54.35	<0,05	-129.50	<0,05	-123.44	<0,05	-634.65	<0,05

5.2.2. Message Size Variation

For the message size variation, it was not possible to identify an explicit correlation between the increase in the message size and the increase in the average RTT. When analyzing the results for the T-Test, it's possible to identify a variation for most of the cases, but the variation is not linear as can be seen in Table 2.

Table 2. Test T RTT Variation for Message Size

Test T per Message Size (bytes)								
Benchmark	1 - 10		10 - 100		100 - 1000		1000 - 10000	
	Statistic	pValue	Statistic	pValue	Statistic	pValue	Statistic	pValue
Benchmark 1	21.86	<0.05	34.70	<0.05	-86.47	<0.05	53.87	<0.05
Benchmark 2	-0.31	0.75	-1.94	0.05	6.52	<0.05	-12.37	<0.05
Benchmark 3	-2.16	<0.05	3.33	<0.05	-8.02	<0.05	-28.88	<0.05
Benchmark 4	23.76	<0.05	-24.47	<0.05	17.84	<0.05	59.46	<0.05
Benchmark 5	1.55	0.12	1.13	0.25	1.78	0.07	-10.60	<0.05
Benchmark 6	31.48	<0.05	43.89	<0.05	-58.98	<0.05	67.63	<0.05
Benchmark 7	18.23	<0.05	33.87	<0.05	33.01	<0.05	-4.43	<0.05

5.2.3. Number of Message Variation

When increasing the number of message exchanges between agents, the t-test results for the number of messages data indicate that all benchmarks have significant performance differences across all number of message ranges, as can be seen in Table 3. For most of the benchmarks, when varying the number of messages, the t-statistics values are negative, indicating that the first group (with lower messages being sent) has a lower mean than the second group (with a higher number of messages sent). This is concluded by the pValue obtained being lower than 0.05.

Table 3. Test T RTT Variation for Number Of Messages

Test T per Number of Message						
Benchmark	10^3 - 10^4		10^4 - 10^5		10^5 - 10^6	
	Statistic	pValue	Statistic	pValue	Statistic	pValue
Benchmark 1	-41.16	<0.05	-96.47	<0.05	-175.47	<0.05
Benchmark 2	-33.52	<0.05	-74.99	<0.05	86.06	<0.05
Benchmark 3	6.69	<0.05	9.19	<0.05	-129.65	<0.05
Benchmark 4	-75.86	<0.05	-50.37	<0.05	-1271.23	<0.05
Benchmark 5	8.45	<0.05	-17.00	<0.05	-34.75	<0.05
Benchmark 6	-43.81	<0.05	-370.19	<0.05	-1280.49	<0.05
Benchmark 7	-36.32	<0.05	-146.96	<0.05	-1405.03	<0.05

6. Scale-Out Boxplot Results

When limiting each host to use CPU units from 128 to 2058, it was evidenced that the performance of the platform was affected (Figure 11).

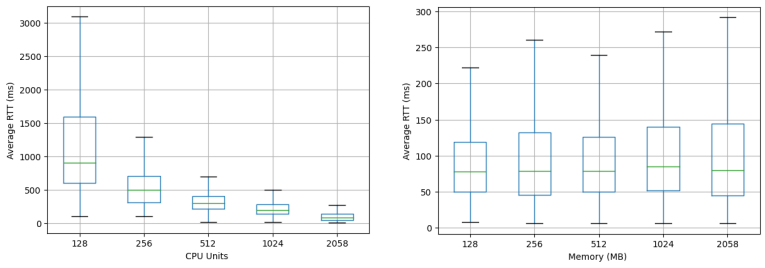


Figure 11. Scale-Up Benchmark Results

Differently from the variation of the CPU Units, the memory limit didn't show

any correlation between the average RTT and the available memory when the host had at least 128MB to operate, as shown in Figure 11.

6.1. T-Test Variation Across Scale-Up Benchmarks

In summary, the t-test results for the resource allocation showed that the increase of CPU leverage led to a reduction of the average RTT mean in all scenarios, while the variation of memory provided no direction correlation between a higher resource allocation and an impact in the average RTT.

Table 4. Test T RTT Variation for Message Size

Test T per Resource								
Benchmark	128 - 256		256 - 512		512 - 1024		1024 - 2048	
	Statistic	pValue	Statistic	pValue	Statistic	pValue	Statistic	pValue
Benchmark 1	65.70	<0.05	40.57	<0.05	28.35	<0.05	58.63	<0.05
Benchmark 2	-13.55	0.75	2.65	0.05	-17.27	<0.05	-7.63	<0.05

7. Discussion and future work

This experiment analyzed the response of the JADE platform in specific contexts for scale-up and scale-out scalability aspects. The performance evaluation approach presented in the current work was first adopted by [Rodrigues 2019], and [Alencar 2020] continued these experiments by deploying the agents into a cloud-based infrastructure. Our work analyzes the response of the JADE platform within a larger system with hundreds of platforms, and it also includes the analysis of the platform from a scale-up approach.

When analyzing the results from the scale-out experiments, it's possible to see that the number of hosts was a common bottleneck factor for most of the experiments, followed by the number of messages sent by each agent. This is an indicator that for a large distributed system, the HTTP MTP protocol might be a limiting factor for the overall system's scalability. In the scale-up approach, varying the memory available for the hosts didn't directly interfere with the average RTT, but the average RTT decreased with the increase of the number of CPU Units available per host, thus indicating that for scaling large distributed MAS that require fast response, making more CPU units available should be the priority.

In the future, the present work can be replicated for other MAS platforms in order to understand how they behave according to the benchmarks used, which can provide an important indicator of whether or not a platform might be used when considered for large-scale MAS systems that need to adhere to specific performance conditions or that must be deployed within devices with hardware limitations. All tables and figures, as well as the data used for analyzing the experiments, are available online [3].

Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES) - Finance Code 001

[3]https://github.com/lti-usp/2023-wesaac-sgursky-casals-brandao/

References

Alencar, R. F. (2020). Escalabilidade e comunicação. Undergraduate thesis, Escola Politécnica da Universidade de São Paulo, São Paulo.

Balke, T. and Gilbert, N. (2014). How do agents make decisions? a survey. *JASSS*, 17:13.

Bellifemine, F., Poggi, A., and Rimassa, G. (1999). *JADE - A FIPA-compliant agent framework*, pages 97–108. The Practical Application Company Ltd.

Bellifemine, F., Poggi, A., and Rimassa, G. (2001). Developing multi-agent systems with a fipa-compliant agent framework. *Softw., Pract. Exper.*, 31:103–128.

Bellifemine, F. L., Caire, G., and Greenwood, D. (2007). *Developing Multi-Agent Systems with JADE*. John Wiley & Sons.

Ferber, J. (1999). *Multi-agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley.

Jennings, N. R. (2000). On agent-based software engineering. *Artificial Intelligence*, 117(2):277–296.

Poslad, S. and Charlton, P. (2001). *Standardizing Agent Interoperability: The FIPA Approach*, volume 2086, pages 98–117. Springer.

Rodrigues, H. (2019). Avaliação de escalabilidade e desempenho da camada de transporte de mensagens em plataformas multiagente. Master's thesis, Escola Politécnica da Universidade de São Paulo.

Russell, S. and Norvig, P. (2016). *Artificial Intelligence: A Modern Approach*. Pearson, New York, NY, 2nd edition.

Stone, P. and Veloso, M. (2000). Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8:345–383.

Such, J., Alberola, J., Mulet, L., Espinosa, A., García-Fornes, A., and Botti, V. (2007). Large-scale multiagent platform benchmarks. *Proceedings of the Multi-Agent Logics, Languages, and Organisations-Federated Workshops (LADS07)*.

Tanenbaum, A. S. (2003). *Computer networks*. Pearson Education India.

Thangarajah, J., Padgham, L., and Harland, J. (2002). Representation and reasoning for goals in bdi agents. *Australian Computer Science Communications*, 24(1):259–265.

Vitaglione, G., Quarta, F., and Cortese, E. (2002). Scalability and performance of jade message transport system. *Autonomous Agents and Multi-Agent Systems*.

Uma Proposta de Mapeamento do Ambiente Exógeno de Sistemas Multi-Agentes Usando Visão Computacional

Douglas Bernardino[1], Leandro Botelho[2], Carlos Eduardo Pantoja[1]

[1]Centro Federal de Educação Tecnológica (CEFET/RJ)
– Rio de Janeiro – RJ – Brasil

[2]Blueshift Brasil
– São Paulo — SP – Brasil

{douglas.bernardiiino, leandrobotelhoalves}@gmail.com

pantoja@cefet-rj.br

Abstract. *The dumping of garbage in the seas, which millions of tons of garbage reach the oceans every year, harms the environment and can even lead to the death of species of aquatic living beings. One proposal to mitigate this problem is autonomous boats that scan a certain part of the sea and collect solid waste that floats on the surface. Existing proposals allow the use of guided boats or completely autonomously. However, current approaches do not consider the performance of cognitive agents and a mixed approach to autonomy, which can reduce search costs in certain regions. The objective of this work is to propose a Multi-Agent System focused on not only mapping but also monitoring an exogenous environment through a camera, using Machine Learning and Deep Learning algorithms for object identification and collection to a Multi-agent System.*

Resumo. *O despejo de lixo nos mares, o qual milhões de toneladas de lixo chegam aos oceanos todos os anos, prejudica o meio ambiente e até pode levar a morte de espécies de seres vivos aquáticos. Uma proposta para mitigar esse problema são barcos autônomos que façam a varredura de certa parte do mar e a coleta de resíduos sólidos que flutuem na superfície. As propostas existentes permitem a utilização de barcos guiados ou de forma totalmente autônoma. Contudo, as atuais abordagens não consideram a atuação de agentes cognitivos e uma abordagem mista de autonomia, as quais podem reduzir custos de buscas em determinadas regiões. O objetivo deste trabalho é propor um modelo de visão computacional capaz de mapear um ambiente exógeno através de uma câmera, utilizando algoritmos de Machine Learning e Deep Learning, para identificação de objetos e protótipos informando seus posicionamentos para um Sistema Multi-Agentes Embarcado.*

1. Introdução

Em um relatório publicado em 2021, a Organização das Nações Unidas enfatiza que a poluição aumentou significativamente ao longo dos anos e deve dobrar até 2030, afetando o estilo de vida humana, a biodiversidade e o clima. O monitoramento e recolhimento de resíduos sólidos são tarefas as quais dependem de intervenção humana

para condução de embarcações especializadas em coleta e na identificação de resíduos. Um Sistema Multi-agente (SMA) é composto por diversos agentes cognitivos que podem atuar de forma autônoma e pró-ativa em ambiente físico para atingir um objetivo em comum [Wooldridge 2000]. Este tipo de sistemas é um candidato a reduzir a dependência humana e na identificação de resíduos, dado suas características.

Existem trabalhos que utilizam SMA para monitorar ambientes aquáticos, como o controle de acesso de barcos autorizados em uma marina na região norte da Itália [Armano and Vargiu 2009], e pela iniciativa de uma *start-up* de Hong Kong que desenvolveu uma embarcação equipada com IA para reconhecer e registrar os tipos de lixo coletados e sua localização [Riccio 2022]. Em contraste com os trabalhos anteriores, esta proposta envolve um barco autônomo com um SMA Embarcado [Brandão et al. 2021] que realiza a coleta autônoma dos resíduos presentes na superfície da água. Para alcançar esse objetivo, é essencial contar com um modelo que possa identificar tanto a posição do barco quanto dos resíduos e transmitir essas informações ao SMA. Dessa forma, o barco autônomo se tornaria uma solução viável para enfrentar o desafio da coleta de resíduos em uma baía, permitindo uma atuação eficiente e consciente do meio ambiente. Com o uso do modelo cognitivo *Belief-Desire-Intention* (BDI) [Bratman 1987], os agentes seriam capazes de raciocinar baseados em crenças, desejos e intenções, podendo melhorar a tomada de decisões e a coordenação das atividades de coleta.

Nesse artigo, será proposto um modelo de visão computacional que seja capaz de identificar protótipos de um veículo terrestre, um objeto específico (uma bola) e montar automaticamente as crenças para os agentes. Uma câmera específica será posicionada no topo de uma sala para identificar os objetos e protótipos que estiverem no quadro da câmera. Uma vez identificados, o modelo encaminhará para os agentes as percepções do protótipo e dos objetos para que este possa deliberar. Para isso serão utilizados a linguagem Python para treinar uma máquina usando *Machine Learning*, junto com um dispositivo Kinect para mapeamento e captura de informações. O SMA embarcado será programado utilizando o framework JaCaMo [Boissier et al. 2013] com agentes especializados em interface de hardware [Pantoja et al. 2016] e agentes que se comunicam com a IoT [Pantoja et al. 2018]. Para identificação de objetos será utilizada a biblioteca Open Source Computer Vision Library (OpenCV) que possui módulos de Processamento de Imagens, estrutura de dados, álgebra linear e algoritmos de Visão computacional como filtros de imagem, calibração de câmera, reconhecimento de objetos, análise estrutural e outros, tendo seu processamento em tempo real de imagens, apoiada pelo framework TensorFlow que é uma biblioteca criada para aprendizado de máquina, computação numérica entre outros, tornando-se uma das principais ferramentas para Machine e Deep Learning.

As informações da localização dos resíduos serão encaminhadas a embarcação a partir de uma rede da Internet das Coisas (*Internet of Things* — IoT) usando o ContextNet [Endler et al. 2011] ou através de uma Interface de Programação de Aplicação — *Application Programming Interface* (API). Espera-se que com uma arquitetura híbrida apoiada pela IoT e a API com monitoramento externo ao SMA embarcado, o modelo informe o posicionamento dos objetos e protótipos adequadamente. Para validar a proposta deste trabalho, optou-se por escolher um modelo terrestre para validar a proposta por facilidade na montagem do *setup* de testes. Esta proposta divide-se em: na Seção 2 são apresentados os conceitos básicos; na Seção 3 a arquitetura do projeto e direcionamentos

são apresentados. Por fim, a Seção 4 apresentam algumas considerações finais.

2. Conceitos Básicos

A tecnologia de agentes surge como uma alternativa para apoiar uma resolução cooperativa de sistemas distribuídos, apresentando vantagens em relação as arquiteturas monolíticas tradicionais. Os SMA são sistemas baseados em agentes os quais grupos de agentes trabalham juntos como um único sistema para cooperarem ou competir por seus objetivos [Russel and Norvig 2004]. A visão computacional é a área da ciência que estuda métodos de aquisição, processamento, análise e entendimento de imagens e, no geral, informações do mundo real, com o intuito de produzir dados numéricos ou simbólicos para realizar alguma função ou tomar decisões [Klette 2014]. Neste trabalho, será necessário treinar um modelo de visão computacional utilizando a linguagem Python e bibliotecas como o OpenCV [Harvey 2010] e TensorFlow. Esse modelo será capaz de identificar o protótipo de um veículo terrestre e uma bola nas imagens capturadas pelo dispositivo Kinect, fornecendo assim as informações necessárias para os agentes do SMA embarcado.

3. Sistema Cognitivo de Monitoramento Aquático

Nesta seção serão apresentadas a metodologia e a arquitetura do sistema cognitivo utilizando o framework JaCaMo para monitoramento de resíduos sólidos superficiais em água. Por exemplo, na Baía de Guanabara, no Rio de Janeiro, frequentemente encontram-se resíduos provenientes do descarte inapropriado de diversos tipos de lixos (Figura 1). De acordo com uma matéria publicada pelo jornal O Globo, a concentração de resíduos na baía de Guanabara é de 100 toneladas por dia [Carvalho 2022]. Sistemas cognitivos podem auxiliar em prover pró-atividade e autonomia de para identificar e coletar resíduos.

Figura 1. Foto da Baía de Guanabara

Neste cenário, um barco com um SMA embarcado será capaz de se locomover pelas águas e recolher de forma autônoma os resíduos sólidos que estiverem flutuando. Este barco receberá através de uma rede da IoT as localizações de onde existem resíduos, que são identificados automaticamente por uma câmera monitorando 24h a baía. O Barco utilizará um SMA utilizando o JaCaMo com arquiteturas customizadas de agentes Argo,

o qual é responsável pelo interfaceamento com o Hardware e agentes Comunicadores, que se comunicam com um servidor IoT usando o ContextNet. O JaCaMo foi escolhido por se tratar de um *framework* com ampla adoção na academia e contar com as extensões para hardware e comunicação. A Figura 2 exibe a arquitetura do sistema de monitoramento.

Para a identificação dos resíduos sólidos em água, uma câmera seria responsável por monitorar constantemente certa região da Baía de Guanabara. A partir desse monitoramento, serão armazenadas imagens que auxiliarão a criar uma base de dados supervisionada para identificar quando há ou não resíduos. A partir daí, algoritmos de Aprendizado de Máquina (*Machine Learning* — ML) poderão ser utilizados para mapear em quadrantes as regiões da Baía. Uma vez identificadas regiões com potenciais resíduos sólidos, o sistema de monitoramento enviará uma notificação ao barco utilizando a rede IoT ou uma API. O agente Comunicador do SMA embarcado no barco receberá a notificação e o agente Argo de forma autônoma irá obter seu posicionamento atual através do Sistema Global de Localização (*Global Positioning System* — GPS). Essas informações serão utilizadas para que o SMA identifique em qual quadrante ele está localizado e para qual quadrante ele precisa se deslocar.

Levando em consideração a dificuldade inicial em reproduzir o cenário descrito, foi substituído o barco autônomo por veículo terrestre autônomo e o lixo por uma bola de plástico. De formar que um kinect ficará disposto a aproximadamente 2 metros do chão, o carro irá ficar no chão na posição inicial (1,1) e a bola será colocada em vários pontos diferentes no chão, levando em consideração os limites dos quadrantes baseado na percepção da câmera. Desta forma, um veículo poderá realizar a coleta de uma bola, em um ambiente exógeno utilizando as informações recebidas de uma rede IOT ou de uma API em tempo real através de uma solução atrelada a câmera de monitoramento (Kinect). O veículo será capaz de se locomover por quadrantes associados a informação recebida pelo agente, sendo o carro acionado somente se existe identificação do objeto bola.

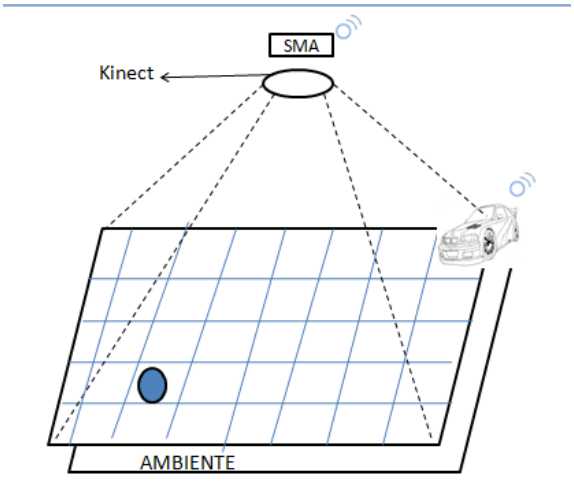


Figura 2. Modelo Conceitual Inicial

3.1. Primeiras Impressões

Inicialmente, a nível de implementação, foram definidas algumas construções a nível de SMA como as crenças, intenções e planos do agente que controlará o barco e procurará

o lixo, chamados respectivamente de **boat** e **garbagefinder**. O **boat** possui uma posição inicial com coordenadas, por exemplo, números inteiros positivos como uma crença. A única intenção inicial é a *hasGarbage* a qual realizará a movimentação nos eixos abscissas e ordenadas no caso de existir uma crença da posição do lixo e do barco. Os planos *hasGarbage* e *move* consiste inicialmente em verificar se o **boat** está em cima do resíduo, se a resposta for positiva, o mesmo irá coleta-lo. O plano *move* é responsável por mover o **boat** até que ele chegue na posição onde foi identificado o lixo.

Primeiramente, o barco é iniciado em uma posição (1,1). Após o recebimento das posições do barco e lixo, acionará plano *hasGarbage* o qual checará se a posição do barco é diferente da posição do lixo. Caso as posições sejam diferentes mover-se-á inicialmente de forma unitária, ora positiva ora negativa, no eixo das abscissas até que sejam iguais. Igualando-se, o agente **boat** movimentar-se-á, agora, no eixo das ordenadas, novamente, de forma unitária ou positivamente ou negativamente, como fosse traçar uma plano cartesiana até que alcance o lixo. Ademais a coleta, o agente **garbagefinder** mandará mensagem avisando ao outro dispositivo que foi realizada a coleta com sucesso. A Figura 3 mostra a implementação do agente **boat** e o *output* da execução do SMA.

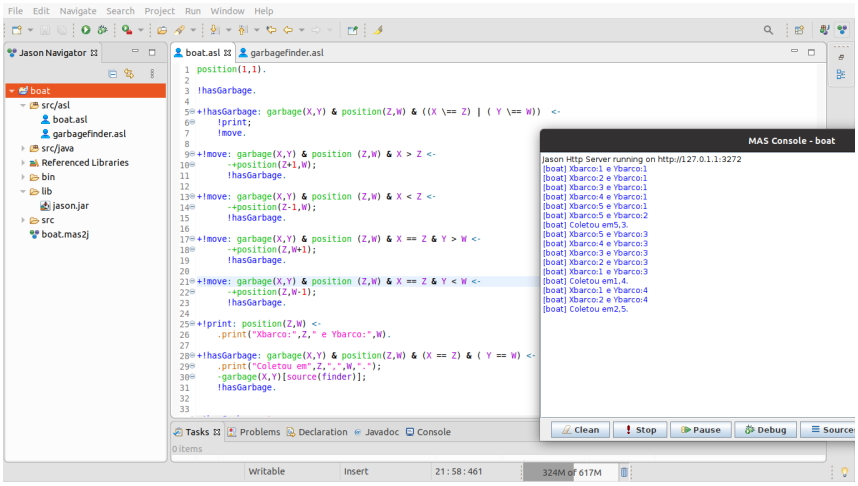


Figura 3. A programação do agente boat.

Na versão atual, algumas limitações ainda existem, tais como: o agente **boat** só consegue lidar com uma informação de lixo por vez e caso receba uma outra informação de outro lixo, poderá ocorrer um desvio de percurso. Além disso, existe ausência na verificação se há um outro dispositivo próximo, podendo haver colisões.

4. Discussão

Por ser um trabalho em andamento, o projeto apresenta as tecnologias iniciais e o funcionamento do SMA de forma que possa no futuro ser integrado ao projeto finalizado. A proposta é utilizar um SMA embarcado e a IoT ou uma API para o monitoramento e detecção de objetos para a coleta destes por um carro autônomo ou por um barco autônomo.

Como trabalhos futuros, é planejado calibrar a câmera Kinect, pesquisar uma forma de comunicação entre o código Python e o framework de programação JaCaMo, realizar o treinamento de máquina, realizar testes em ambientes reais e, posteriormente, escrever sobre os resultados obtidos com o protótipo terrestre e marinho em ambiente

controlado (piscina). Inicialmente, não será levado em consideração a atuação do vento, das correntes marítimas, outros agentes coletando lixo, o cálculo da menor distância entre os pontos do barco e do lixo mais próximo, a interação com outros barcos e outros fatores naturais. Posteriormente, com as fases mencionadas concluídas, as questões acima serão abordadas. Dado a dificuldade em resolver esses problemas em ambiente marinho foi optado por iniciar o projeto considerando um ambiente terrestre.

Referências

Armano, G. and Vargiu, E. (2009). A multiagent system for monitoring boats in marine reserves. In *International Workshop on Programming Multi-Agent Systems*, pages 254–265, Berlin. Springer.

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.

Brandão, F. C., Lima, M. A. T., Pantoja, C. E., Zahn, J., and Viterbo, J. (2021). Engineering approaches for programming agent-based iot objects using the resource management architecture. *Sensors*, 21(23). https://doi.org/10.3390/s21238110.

Bratman, M. E. (1987). *Intention, Plans and Practical Reasoning*. Cambridge Press.

Carvalho, Janaina e Lima, M. (2022). Baía de guanabara agoniza com despejo de quase 100 toneladas de lixo por dia, 30 anos após a eco-92. *O Globo*.

Endler, M., Baptista, G., Silva, L., Vasconcelos, R., Malcher, M., Pantoja, V., Pinheiro, V., and Viterbo, J. (2011). Contextnet: context reasoning and sharing middleware for large-scale pervasive collaboration and social networking. In *Proceedings of the Workshop on Posters and Demos Track*, page 2. ACM.

Harvey, A. (2010). Opencv face detection: Visualized. *Vimeo [En lınia] Disponible: https://vimeo. com/12774628*.

Klette, R. (2014). *Concise computer vision*, volume 233. Springer.

Pantoja, C. E., Soares, H. D., Viterbo, J., and El Fallah-Seghrouchni, A. (2018). An architecture for the development of ambient intelligence systems managed by embedded agents. In *SEKE*, pages 215–220. https://doi.org/10.18293/SEKE2018-110.

Pantoja, C. E., Stabile, M. F., Lazarin, N. M., and Sichman, J. S. (2016). ARGO: An Extended Jason Architecture that Facilitates Embedded Robotic Agents Programming. In Baldoni, M., Müller, J. P., Nunes, I., and Zalila-Wenkstern, R., editors, *Engineering Multi-Agent Systems*, pages 136–155, Cham. Springer International Publishing. https://doi.org/10.1007/978-3-319-50983-9_8.

Riccio, G. (2022). Clearbot neo, barco robótico que limpa portos e rios com a ajuda da ia. *Futuro próximo*.

Russel, S. and Norvig, P. (2004). *Inteligência Artificial*.

Wooldridge, M. J. (2000). *Reasoning about rational agents*. MIT press.

Modelagem da Estrutura Organizacional do Sistema Multiagente associado a um Jogo RPG no contexto de Recursos Hídricos

Míriam Blank Born[1], Marilton Sanchotene de Aguiar[1] e Diana F. Adamatti[2]

[1]Programa de Pós-Graduação em Computação
Universidade Federal de Pelotas (UFPEL)
Pelotas – RS – Brasil

[2]Centro de Ciências Computacionais (C3)
Universidade Federal do Rio Grande (FURG)
Rio Grande – RS – Brasil

{marilton,mbborn}@inf.ufpel.edu.br, dianaada@gmail.com

Abstract. *Managing water resources presents numerous and complex challenges since it must guarantee water availability for various activities such as public supply and preservation of the environment. In this context, techniques such as role-playing games and multiagent systems show promise for problem-solving. This article presents as a case study an RPG game applied to the participatory management of water resources within the watershed of Lagoa Mirim and Canal São Gonçalo. Furthermore, we modeled the agents belonging to this study as a multiagent organization composed of three groups of agents: regulators, producers, and inspectors. Also, we modeled the organization of the multiagent system in the* $\mathcal{M}$OISE$^+$ *tool, which integrates the JaCaMo platform for multiagent programming.*

Resumo. *A gestão de recursos hídricos apresenta inúmeros e complexos desafios visto que, deve garantir a disponibilidade da água para as diversas atividades como o abastecimento público e a preservação do meio ambiente. Neste contexto, técnicas como jogos de RPG e também a utilização de sistemas multiagente mostram-se promissoras para a resolução de problemas. Este artigo apresenta como estudo de caso um jogo de RPG aplicado a gestão participativa dos recursos hídricos no âmbito da bacia hidrográfica da Lagoa Mirim e Canal São Gonçalo. Além disso, os agentes pertencentes a este estudo são modelados como uma organização multiagente composta por três grupos de agentes: reguladores, produtores e fiscalizadores. A organização do sistema multiagente foi modelada na ferramenta* $\mathcal{M}$OISE$^+$ *a qual integra a plataforma JaCaMo para a programação multiagente.*

1. Introdução

Os recursos naturais são elementos indispensáveis na vida do ser humano e, por isso, é de extrema importância em nossa sociedade o gerenciamento destes recursos visando melhorar a maneira de gerenciar terras, águas, minerais, petróleo, florestas e animais [Darby 2010]. Com relação ao recurso hídrico, especificamente, o território

brasileiro dispõem de grandes bacias hidrográficas, no entanto, apresentando inúmeros conflitos relacionados à distribuição e compartilhamento da água [Brito et al. 2020, Born et al. 2019b].

Segundo [Fuller et al. 2007], os principais desafios computacionais no contexto da gestão de recursos naturais são: (a) gerenciamento e comunicação de dados, (b) análise de dados e (c) controle e otimização. Na busca de soluções que auxiliem neste cenário existem ferramentas computacionais que utilizam técnicas da Inteligência Artificial como Sistemas Multiagente (SMA), Redes Neurais, Algoritmos Genéticos, Autômatos Celulares, Inteligência de Enxames, dentre outros. Os modelos de simulação baseados em Agentes e em Sistemas Multiagente, por se apresentarem como técnicas bastantes flexíveis, destacam-se neste domínio onde se apresentam cenários complexos, como os sistemas sociais, ecológicos e políticas públicas, apresentando soluções adequadas e satisfatórias [Filatova et al. 2013].

Os jogos de papéis (RPG, acrônimo do termo em inglês, *Role-Playing Game*) compreendem outra técnica aplicada a diversos estudos [Adamatti 2007, Filatova et al. 2013, Page et al. 2000]. Neste tipo de jogo, os jogadores interpretam um personagem desempenhando um papel e tomam decisões de acordo com este para alcançarem um determinado objetivo, seja este individual ou coletivo [Born 2022].

Desta forma, a utilização dos SMA produz bons resultados em um cenário onde é utilizado o RPG, na gestão de recursos hídricos. Quando utilizam-se os jogos de papéis (RPG) torna-se possível compor discussões, aprendizado e estratégias sobre um contexto específico e isso possibilita a tomada de decisões [Born 2022].

Este artigo tem o objetivo de apresentar parte da modelagem organizacional do Sistema Multiagente subjacente ao Jogo RPG Gorim, estudo de caso deste trabalho inicialmente apresentado em [Born et al. 2019a], mais especificamente a especificação estrutural da organização.

O artigo encontra-se organizado da seguinte forma, na Seção 2 é apresentado o referencial teórico de Sistemas Multiagente, Jogos de Papéis e do *framework* JaCaMo. Na Seção 3 é introduzido um panorama geral sobre a ferramenta $\mathcal{M}$OISE$^+$ com suas características, é apresentado o estudo de caso, bem como a modelagem de uma organização multiagente no $\mathcal{M}$OISE$^+$. E por fim, a Seção 4 apresenta as conclusões e trabalhos futuros deste estudo.

2. Referencial Teórico

As aplicações que utilizam Sistemas Multiagente para modelagem e simulação no gerenciamento de recursos naturais são vistas como uma possibilidade efetiva na busca de soluções para estes problemas complexos pois apresentam benefícios como: (a) rapidez na resolução de problemas visto a inerência do processamento concorrente; (b) aumento da flexibilidade e escalabilidade através da conexão de vários sistemas; (c) aumento da capacidade de resposta a um determinado problema pelo fato de todos os recursos estarem localizados no mesmo ambiente; e, (d) a modularidade obtida mediante esta técnica [Alvares and Sichman 1997, Bordini et al. 2001].

Os modelos biológicos, assim como as interações sociais, servem de inspiração para o desenvolvimento de sistemas onde agentes inteligentes podem ser concebidos

por meio de dispositivos de *hardware* e/ou *software* [Artero 2009]. Os agentes representados por estes equipamentos ou programas devem ter a capacidade de perceber seu ambiente por meio de sensores e de agir sobre este por meio de atuadores [Russell and Norvig 2013].

Sendo assim, é relevante que os agentes de um sistema possuam três características: *cooperar*, *aprender* e *agir de maneira autônoma* de acordo com [Nwana 1996]. [Bordini et al. 2001] acrescentam ainda coordenação, competição e negociação como aspectos relevantes na concepção de agentes. A partir destas características, existem diferentes maneiras de classificar os agentes [Nwana 1996, Artero 2009, Coppin 2010], compreendendo-se basicamente em agentes reativos, agentes colaborativos, agentes de comunicação e agentes de aprendizado.

A arquitetura amplamente utilizada no desenvolvimento de SMA é a BDI (acrônimo para os termos em inglês *Beliefs*, *Desires*, *Intentions*), baseada em um modelo cognitivo que representam crenças, desejos e intenções [Hübner et al. 2004]. Para [Wooldridge 2002], *crenças* representam o que o agente sabe sobre si mesmo, dos demais agentes e o ambiente ao qual está inserido; *desejos* representam os estados que o agente almeja atingir, geralmente são objetivos; e, *intenções* são representadas pela sequência de ações que determinado agente executa para alcançar um objetivo.

Contudo, as propriedades, a arquitetura e a estrutura dos agentes, bem como o ambiente em que estes estão inseridos, são implementados de acordo com o problema a ser resolvido, a complexidade e o domínio específico de cada aplicação. Em [Russell and Norvig 2013, Rezende 2005, Luger 2013], diversas aplicações e algoritmos são apresentados de forma a exemplificar a busca de soluções para esta demanda crescente de problemas.

Os Jogos de Papéis ou RPG (*Role-Playing Game*) compreendem um tipo de jogo onde os jogadores "interpretam" um personagem, criado dentro de um determinado cenário ou ambiente. Esses personagens respeitam um sistema de regras, que serve para organizar suas ações, determinando os limites do que pode ou não ser feito [Born et al. 2019a].

Os jogos de papéis estão situados entre os jogos e o teatro e consistem numa técnica onde se determinam regras e comportamentos de jogadores, bem como um contexto imaginário (ambiente) [Adamatti 2007]. Assim, o RPG também é um meio de revelar alguns aspectos das relações sociais, permitindo a observação direta das interações entre os jogadores. Desse modo, em um RPG não há vencedores e perdedores, dado que possui aspecto de colaboração em vez de competição. Ao final, deve-se completar uma história construída a partir das regras do jogo, na busca dos objetivos individuais e/ou coletivos [Adamatti 2007].

Assim, os RPG são muito utilizados em treinamento, pois podem colocar os jogadores em situações de tomada de decisão similares às reais. Além disso, devido ao fator lúdico envolvido nos jogos, fazem com que o treinamento e/ou aprendizagem de determinado assunto seja facilitado [Born et al. 2019a].

O *framework* de programação multiagente JaCaMo [Boissier et al. 2016] tem sido desenvolvido ao longo dos anos e integra as plataformas Jason [Bordini et al. 2007], utilizado no desenvolvimento de agentes autônomos; CArtAgO [Ricci et al. 2011], aplicado

no desenvolvimento de ambientes compartilhados; e, o $\mathcal{M}$OISE$^+$ [Hübner et al. 2007] desenvolvido para a modelagem de organizações multiagente.

Em JaCaMo consideram-se três dimensões: agente, ambiente e organização, com intuito de facilitar ao desenvolvedor a modelagem e implementação de sistemas multiagente complexos [Thomasi 2014]. A partir da integração de ferramentas, composta pelos agentes, ambiente, interação e organização, oferece um recurso para a escalabilidade de aplicativos complexos permitindo assim sua distribuição em diversos nós. Em Jason, interpretador para a linguagem *AgentSpeak-L*, é possível programar o comportamento do agente de acordo com uma abordagem declarativa, baseada em lógica e arquitetura BDI.

O CArtAgO possui artefatos especiais possibilitando o gerenciamento da estrutura do ambiente e permite que os agentes gerenciem o ciclo de vida do ambiente de trabalho e interajam com o ambiente externo através de serviços disponíveis como: criar, destruir, ingressar e sair de áreas de trabalho locais ou remotas. O $\mathcal{M}$OISE$^+$, onde estrutura organizacional é concebida, possui três dimensões: a estrutura (papéis), o funcionamento (planos globais) e as normas (obrigações) da organização multiagente. Considerando que a visão da organização centraliza-se no sistema, as normas pertencentes à esta são externas aos agentes e podem ser consultadas por estes.

Na literatura, inúmeras pesquisas buscam e apresentam soluções viáveis e interessantes no contexto de gerenciamento de recursos naturais com técnicas de SMA e RPG. Nestes casos, o RPG auxilia na busca por soluções compartilhadas entre partes de determinado sistema e também na tomada de decisão sobre aspectos importantes[Born 2022].

No trabalho [Farias et al. 2019] foi realizada uma revisão sistemática sobre as áreas de gestão de recursos naturais, sistemas multiagente e jogos de papéis verificando-se quatro formas de integração entre RPG e SMA. Em [Adamatti et al. 2009], a partir da integração entre RPG e SMA foi desenvolvida a metodologia GMABS para a implementação de dois protótipos, o jogo JogoMan de RPG jogado em sessões presenciais e o ViP-JogoMan com inserção de jogadores virtuais.

Em [Farolfi et al. 2010] foi proposto uma metodologia detalhada para formalizar e sistematizar as fases de modelagem do SMA desenvolvido chamado KatAWARE no estudo de caso da bacia hidrográfica de Kat River. No estudo de [Le Page et al. 2014] a modelagem SMA desenvolvida buscou investigar a interação entre os agricultores/produtores de arroz, a disponibilidade de água e terra para esta produção do nordeste da Tailândia. De acordo com a pesquisa de [Le Page et al. 2016] foi criado o de jogo de RPG ReHab com o objetivo de introduzir as principais ideias de gestão natural para os alunos nos primeiros anos de um curso de pós-graduação na área. Com o jogo, os desenvolvedores concluíram que os jogadores, neste caso os estudantes, conseguiram modelar, aprender e refletir sobre as respostas do sistema socioecológico de forma individual ou coletiva, considerando vários regimes de gerenciamento.

3. Modelagem da Organização

A modelagem deste estudo de caso baseia-se na representação das interações básicas entre os papéis do sistema e a atuação destes no ambiente, de acordo com a especificação proposta em [Born et al. 2019a]. No estudo, os agentes são classificados de acordo com os papéis que assumem e são divididos em três grupos principais (reguladores, fiscalizadores ou produtores), conforme Figura 1.

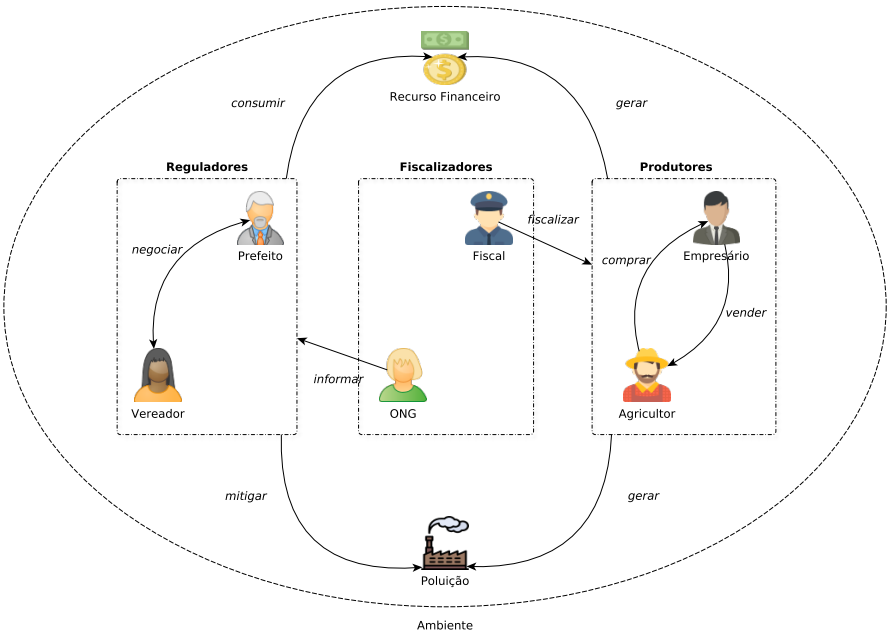


Figura 1. Diagrama de Integração do modelo proposto [Born et al. 2019a].

De acordo com [Born 2022], os agentes reguladores podem assumir os papéis de *prefeito* ou *vereador*, sendo responsáveis pela administração de recursos financeiros, com o objetivo de controlar/mitigar a poluição do ambiente. Os agentes fiscalizadores possuem como atribuição fiscalizar ou informar irregularidades que impactam no ambiente a partir da produção e assumem os papéis de *fiscal ambiental* ou *ONG* (Organização Não-Governamental). Os agentes produtores, nos papéis de *empresário* ou *agricultor*, exploram o ambiente para suas produções e seu principal objetivo é obter recursos financeiros. É relevante salientar que todos os papéis podem interagir neste sistema.

A ferramenta $\mathcal{M}$OISE$^+$ (do inglês, *Model of Organization for multI-agent SystEms*), que integra a plataforma JaCaMo, caracteriza-se por ser um modelo organizacional para sistemas multiagente composto de três dimensões: (a) *estrutural*, constituída pelos grupos, papéis e ligações; (b) *funcional*, englobando planos globais, metas e missões; e, (c) *normativa* ou *deôntica*, a qual define as obrigações e permissões dos agentes. Em [Hübner et al. 2010] encontra-se um amplo tutorial com a conceitualização e exemplos de utilização do $\mathcal{M}$OISE$^+$.

No desenvolvimento deste trabalho, elaborou-se a especificação estrutural da organização para este estudo de caso, conforme Figura 2, a partir do diagrama de interação da Figura 1. Esta especificação estrutural possui quatro grupos: *Jogo, Regulador, Fiscalizador* e *Produtor*, considerando que a interação pode ser interna e externa a estes. Todos os papéis são herdados do papel *Jogador* e possuem interação bilateral. Ainda, cada grupo e papel possuem as cardinalidades correspondentes, sendo que esta elaboração baseou-se no trabalho de [Hübner et al. 2002] com as devidas adaptações ao contexto deste estudo.

No grupo *Regulador*, o agente pode assumir o papel de *prefeito* ou *vereador*, permitindo-se um prefeito e dois vereadores. No grupo *Fiscalizador*, o agente pode assumir o papel de *ONG* ou de *fiscal ambiental*, A ONG é responsável por informar aos

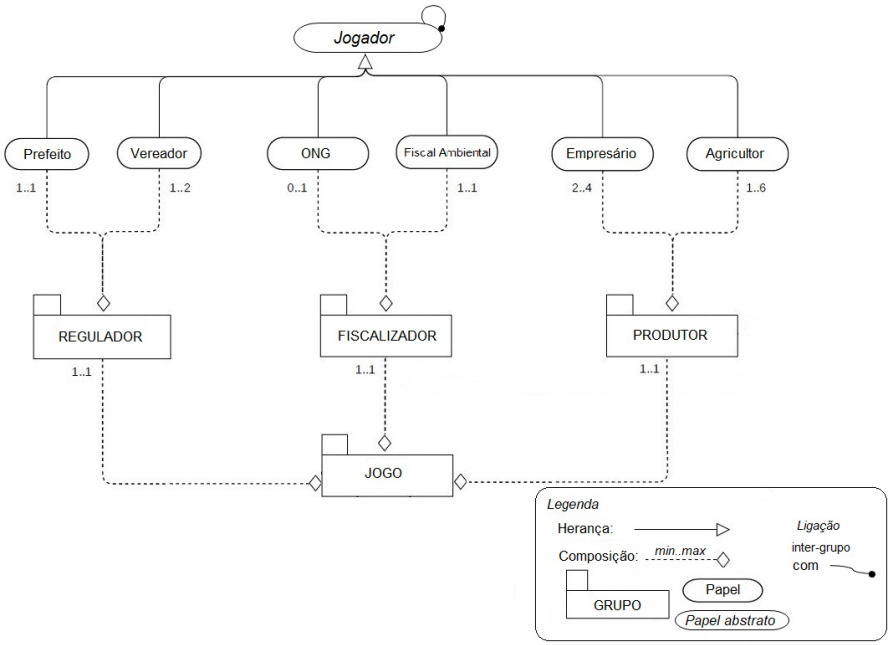


Figura 2. Especificação Estrutural da Organização para o modelo proposto.

agentes reguladores o estado atual dos níveis de poluição do ambiente, com o objetivo de conscientizar/pressionar os outros agentes a realizarem ações que diminuam os níveis de poluição. Entretanto, é importante salientar que definiu-se que a ONG é um NPC (acrônimo do termo em inglês, *Non Player Character*), ou seja, representa uma interface da situação atual do jogo que fornece informações sobre o ambiente. Na modelagem do estudo de caso e é obrigatório pelo menos um fiscal ambiental. No grupo *Produtor* o agente pode assumir o papel de *empresário* (no mínimo 1 e no máximo 4) ou *agricultor* (no mínimo 1 e no máximo 6). Por fim, o grupo *Jogo* deve conter pelo menos um grupo de cada um dos demais.

Na modelagem da Especificação Funcional foram construídos os esquemas sociais e a relação de preferência entre as missões que os papéis envolvidos devem alcançar. Na organização proposta tais esquemas estão representados nas Figuras 3, 4 e 5, bem como a proposta de propina na Figura 6, que pode ser negociada, ou seja, transferida e/ou recebida entre quaisquer papéis do sistema.

A Figura 3 apresenta a especificação funcional do grupo dos *Reguladores*. Esta árvore de decomposição representa todas as ações que podem ser assumidas pelos dois papéis de *prefeito* e *vereador*. Neste caso, para concluir o objetivo raiz da árvore *g0 - Concluir medidas regulatórias*, *g1 - Realizar tratamentos* e *g2 - Regulamentar taxas* as ações podem ser realizada em paralelo.

Na Figura 4 para concluir o objetivo *g18 - Contatar produtor* as ações *g22 - Localizar produtor*, *g23 - Ir a propriedade* e *g24 - Falar com o produtor* são realizadas em sequência.

Na Figura 5 para concluir o objetivo *g16 - Solicitar selo verde* é realizada uma seleção entre as ações *g63 - Requisitar selo verde* ou *g64 - Não requisitar selo verde*.

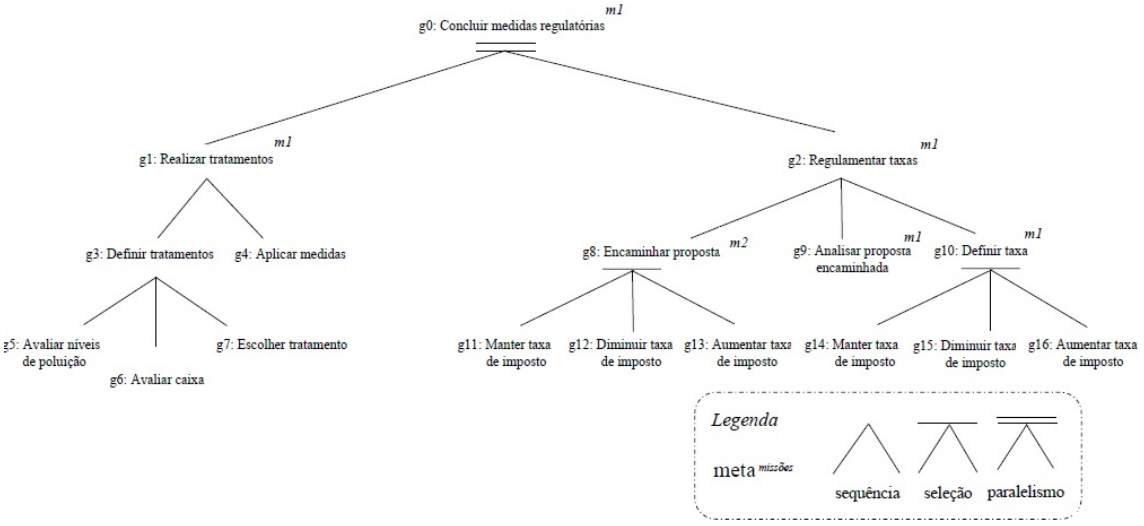


Figura 3. Especificação funcional do grupo dos *Reguladores*.

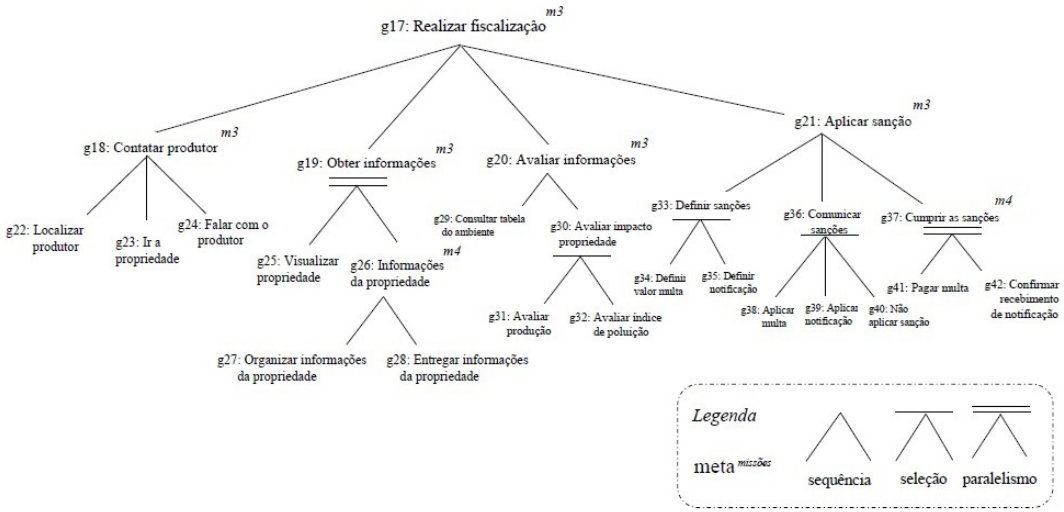


Figura 4. Especificação funcional do grupo dos *Fiscalizadores*.

A Figura 6 apresenta a dinâmica da obtenção de propina que pode ser negociada e transferida entre quaisquer agentes do sistema.

Na Especificação Deôntica (ED) ou Normativa têm-se a relação entre a EE e a EF especificando qual(is) missão(ões) cada papel possui permissão ou obrigação de se comprometer no sistema [Born 2022]. Por uma questão de espaço serão apresentadas algumas ED do sistema desenvolvido. Desta forma, as ED da `Propina` (todos os grupos, na Tabela 1) e dos grupos `Regulador` (`Prefeito` na Tabela 2 e `Vereador` na Tabela 3).

4. Conclusões e Trabalhos Futuros

O gerenciamento de recursos naturais mostrou-se, ao longo dos anos, de grande importância. Desta forma, a aplicação de SMA e de um *framework* que permita a

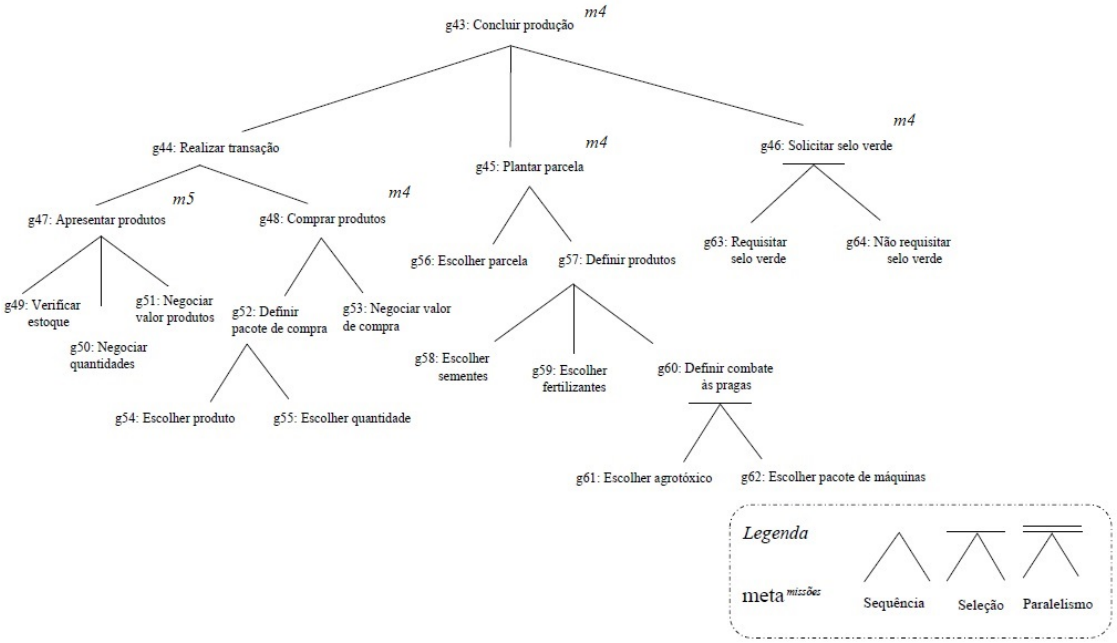


Figura 5. Especificação funcional do grupo dos *Produtores*.

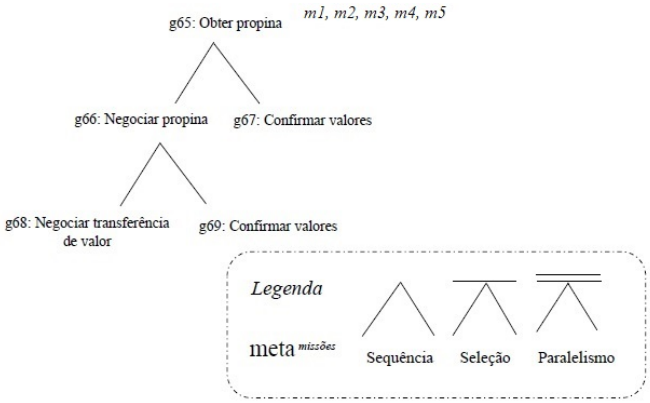


Figura 6. Especificação funcional da propina geral a todos os grupos.

Tabela 1. Missões e Planos dos grupos `Regulador`, `Fiscalizador`, `Produtor` para Propina.

Missão	m_1, m_2, m_3, m_4, m_5 = $\langle O, \{g_{65}, g_{66}, g_{67}, g_{68}, g_{69}\}$, Todos os Grupos$\rangle$
Objetivos	g_{65}: Obter propina g_{66}: Negociar propina g_{67}: Confirmar valores g_{68}: Negociar transferência de valor g_{69}: Confirmar valores
Planos	$[g_{65}]$, $[g_{66}]$, $[g_{67}]$, $[g_{68}]$, $[g_{69}]$ g_{65} = g_{66}, g_{67} g_{66} = g_{68}, g_{69}

especificação da organização do sistema e sua implementação de forma integrada, pode fornecer resultados promissores.

Tabela 2. Missões e Planos do grupo `Regulador -- Prefeito`.

Missão	$m_1 = \langle O, \{g_0, g_1, g_2, g_3, g_4, g_5, g_6, g_7, g_9, g_{10}, g_{14}, g_{15}, g_{16}\}, \text{Prefeito}\rangle$
Objetivos	g_0: Concluir medidas regulatórias
	g_1: Realizar tratamentos
	g_2: Regulamentar taxas
	g_3: Definir tratamentos
	g_4: Aplicar medidas
	g_5: Avaliar níveis de poluição
	g_6: Avaliar caixa
	g_7: Escolher tratamento
	g_9: Analisar proposta encaminhada
	g_{10}: Definir taxa
	g_{14}: Manter taxa de imposto
	g_{15}: Diminuir taxa de imposto
	g_{16}: Aumentar taxa de imposto
Planos	$[g_0]$, $[g_1]$, $[g_2]$, $[g_3]$, $[g_4]$, $[g_5]$, $[g_6]$, $[g_7]$, $[g_9]$, $[g_{10}]$, $[g_{14}]$, $[g_{15}]$, $[g_{16}]$
	$g_0 = g_1 \mid\mid g_2$
	$g_1 = g_3, g_4$
	$g_3 = g_5, g_6, g_7$
	$g_2 = g_9, g_{10}$
	$g_{10} = g_{14} \mid g_{15} \mid g_{16}$

Tabela 3. Missões e Planos do grupo `Regulador -- Vereador`.

Missão	$m_2 = \langle O, \{g_8\}, \text{Vereador}\rangle$
Objetivos	g_8: Encaminhar proposta
Planos	$[g_8]$
	$g_8 = g_{11} \mid g_{12} \mid g_{13}$

A modelagem da organização deste estudo é complexa, devido aos inúmeros aspectos envolvidos, tais como: as especificações devem ser bem definidas de todos os papéis que os agentes podem assumir, bem como suas metas e relacionamentos, troca de mensagem e ambiente.

Neste artigo apresentou-se a modelagem de um sistema multiagente de uma organização no contexto de estudo dos recursos hídricos. O estudo de caso no qual baseou-se este trabalho, apresenta de forma definida, através do diagrama de interação (Figura 1), a modelagem do sistema para a região de uma bacia hidrográfica. É importante salientar também que, o foco deste trabalho foi mostrar o estudo de caso e a modelagem da organização nas especificações estrutural e funcional.

Desta forma, o *framework* JaCaMo fornece os subsídios necessários para uma completa especificação, modelagem, implementação e simulação de cenários que permitam um maior entendimento do problema em questão e de análise de estratégias tomadas pelos agentes do sistema quando assumem determinado papel.

Como trabalhos futuros pretende-se, a partir desta modelagem do sistema multiagente da organização na ferramenta $\mathcal{M}$OISE$^+$, implementar e integrar todo o sistema no

Tabela 4. Missões e Planos do grupo `Fiscalizador`.

Missão	$m_3 = \langle O, \{g_{17}, g_{18}, g_{19}, g_{20}, g_{21}, g_{22}, g_{23}, g_{24}, g_{25}, g_{29}, g_{30}, g_{31}, g_{32}, g_{33}, g_{34}, g_{35}, g_{36}, g_{38}, g_{39}, g_{40}\}$, Fiscal Ambiental$\rangle$
Objetivos	g_{17}: Realizar fiscalização g_{18}: Contatar produtor g_{19}: Obter informações g_{20}: Avaliar informações g_{21}: Aplicar sanção g_{22}: Localizar produtor g_{23}: Ir a propriedade g_{24}: Falar com o produtor g_{25}: Visualizar propriedade g_{29}: Consultar tabela do ambiente g_{30}: Avaliar impacto propriedade g_{31}: Avaliar produção g_{32}: Avaliar índice de poluição g_{33}: Definir sanções g_{34}: Definir valor multa g_{35}: Definir notificação g_{36}: Comunicar sanções g_{38}: Aplicar multa g_{39}: Aplicar notificação g_{40}: Não aplicar sanção
Planos	$[g_{17}]$, $[g_{18}]$, $[g_{19}]$, $[g_{20}]$, $[g_{21}]$, $[g_{22}]$, $[g_{23}]$, $[g_{24}]$, $[g_{25}]$, $[g_{29}]$, $[g_{30}]$, $[g_{31}]$, $[g_{32}]$, $[g_{33}]$, $[g_{34}]$, $[g_{35}]$, $[g_{36}]$, $[g_{38}]$, $[g_{39}]$, $[g_{40}]$ $g_{17} = g_{18}, g_{19}, g_{20}, g_{21}$ $g_{18} = g_{22}, g_{23}, g_{24}$ $g_{19} = g_{25} \mid\mid g_{26}$ $g_{20} = g_{29}, g_{30}$ $g_{30} = g_{31} \mid g_{32}$ $g_{21} = g_{33}, g_{36}, g_{37}$ $g_{33} = g_{34} \mid g_{35}$ $g_{36} = g_{38} \mid g_{39} \mid g_{40}$

framework JaCaMo com a versão *web* do jogo de RPG Gorim desenvolvido na linguagem de programação Java no trabalho [Martins 2021].

Referências

Adamatti, D. F. (2007). *Inserção de jogadores virtuais em jogos de papéis para uso em sistemas de apoio à decisão em grupo: um experimento no domínio da gestão de recursos naturais*. PhD thesis, Escola Politécnica – Universidade de São Paulo, São Paulo, Brasil. doi:10.11606/T.3.2007.tde-07012008-154915.

Adamatti, D. F., Sichman, J. S., and Coelho, H. (2009). An analysis of the insertion of virtual players in GMABS methodology using the ViP-JogoMan prototype. *Journal of Artificial Societies and Social Simulation*, 12(3).

Alvares, L. O. and Sichman, J. S. (1997). Introduçao aos sistemas multiagentes. In *XVII Congresso da SBC-Anais JAI'97*.

Artero, A. O. (2009). *Inteligência Artificial: Teoria e Prática*. Editora Livraria da Física, São Paulo/SP, 1a edition.

Boissier, O., Hübner, J. F., and Ricci, A. (2016). The jacamo framework. In *Social coordination frameworks for social technical systems*, pages 125–151. Springer.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming multi-agent systems in AgentSpeak using Jason*, volume 8. John Wiley & Sons.

Bordini, R. H., Vieira, R., and Moreira, A. F. (2001). Fundamentos de sistemas multiagentes. In *Anais do XXI Congresso da Sociedade Brasileira de Computação (SBC2001)*, volume 2, pages 3–41.

Born, M., Leitzke, B., Farias, G., Melo, M., Gonçalves, M., Rodrigues, P., Martins, V., Barbosa, R., Aguiar, M., and Adamatti, D. (2019a). Sistema multiagente para gestão de recursos hídricos: Modelagem da bacia do são gonçalo e da lagoa mirim. In *Anais do X Workshop de Computação Aplicada a Gestão do Meio Ambiente e Recursos Naturais*, pages 87–96, Belém/PA. SBC, sol.sbc.org.br.

Born, M., Leitzke, B. S., Farias, G., Aguiar, M., and Adamatti, D. F. (2019b). Modelagem baseada em agentes para análise de recursos hídricos. In *Anais do XIII Workshop-Escola de Sistemas de Agentes, seus Ambientes e apliCacoes (WESAAC 2019)*, pages 107–118, Florianópolis/SC. wesaac.c3.furg.br.

Born, M. B. (2022). *Modelagem a nível organizacional de agentes em um jogo do tipo RPG: estudo da complexidade de seus personagens e de suas funcionalidades.* Tese (doutorado em computação), Centro de Desenvolvimento Tecnológico – Universidade Federal de Pelotas, Pelotas/RS/Brasil.

Brito, A. D., Lopes, J. C., and dos Anjos Neta, M. M. S. (2020). Tripé da governança: Poder público, setor privado e a sociedade civil em busca de uma gestão integrada dos recursos hídricos. *Revista Gestão & Sustentabilidade Ambiental*, 8(4):506–522.

Coppin, B. (2010). *Inteligência Artificial*. Rio de Janeiro: LTC, 3a edition.

Darby, S. (2010). Natural resource governance: New frontiers in transparency and accountability. *Transparency Accountability Initiative*.

Farias, G., Leitzke, B., Born, M., Aguiar, M., and Adamatti, D. F. (2019). Systematic review of natural resource management using multiagent systems and role-playing games. *Research in Computing Science*, 148(11):91–102.

Farolfi, S., Müller, J.-P., and Bonté, B. (2010). An iterative construction of multi-agent models to represent water supply and demand dynamics at the catchment level. *Environmental Modelling & Software*, 25(10):1130–1148.

Filatova, T., Verburg, P. H., Parker, D. C., and Stannard, C. A. (2013). Spatial agent-based models for socio-ecological systems: Challenges and prospects. *Environmental modelling & software*, 45:1–7.

Fuller, M. M., Wang, D., Gross, L. J., and Berry, M. W. (2007). Computational science for natural resource management. *Computing in Science & Engineering*, 9(4):40.

Hübner, J. F., Bordini, R. H., and Vieira, R. (2004). Introdução ao desenvolvimento de sistemas multiagentes com jason. *XII Escola de Informática da SBC*, 2:51–89.

Hübner, J. F., Sichman, J. S., and Boissier, O. (2002). Moise+ towards a structural, functional, and deontic model for mas organization. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pages 501–502.

Hübner, 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. *International Journal of Agent-Oriented Software Engineering*, 1(3/4):370–395.

Hübner, J. F., Sichman, J. S., and Boissier, O. (2010). Moise+ tutorial.

Le Page, C., Dray, A., Perez, P., and Garcia, C. (2016). Exploring how knowledge and communication influence natural resources management with ReHab. *Simulation & Gaming*, 47(2):257–284.

Le Page, C., Naivinit, W., Trébuil, G., and Gajaseni, N. (2014). Companion modelling with rice farmers to characterise and parameterise an agent-based model on the land/water use and labour migration in northeast Thailand. In *Empirical Agent-Based Modelling - Challenges and Solutions*, pages 207–221. Springer, New York/USA.

Luger, G. F. (2013). *Artificial intelligence : structures and strategies for complex problem solving*. Addison-Wesley, Boston, Massachusetts, EUA, 6a edition.

Martins, V. B. (2021). Gorimweb: um rpg para gestão de recursos hídricos na plataforma web. Dissertação (mestrado em engenharia da computação), Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande, Rio Grande, RS.

Nwana, H. S. (1996). Software agents: An overview. *The knowledge engineering review*, 11(3):205–244.

Page, C. L., Bousquet, F., Bakam, I., Bah, A., and Baron, C. (2000). CORMAS : A multiagent simulation toolkit to model natural and social dynamics at multiple scales. In *Wageningen : Resource Modeling Association*.

Rezende, S. O. (2005). *Sistemas inteligentes: fundamentos e aplicações*. Editora Manole Ltda, Barueri/SP, 1a edition.

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.

Russell, S. and Norvig, P. (2013). *Inteligência Artificial*. Elsevier Ltda, Rio de Janeiro/RJ, 3a edition.

Thomasi, C. D. (2014). Orias: uma infraestrutura de nível micro-organizacional baseada em artefatos para sistemas multiagentes. Master's thesis, Programa de Pós-Graduação em Computação, UFRGS, Porto Alegre/RS.

Wooldridge, M. (2002). An introduction to multi agent systems, department of computer science, university of liverpool, uk.

Autocuidado de indivíduos com diabetes mellitus com apoio de Agente Conversacional

Mateus Elias Gundel[1], Jordana Kich[2,4], Rejane Frozza[1,3], Andreia Rosane de Moura Valim[2,4], Janine Koepp[2,4], Lia Gonçalves Possuelo[2,4]

[1] Departamento de Computação, Universidade de Santa Cruz do Sul - UNISC, Santa Cruz do Sul, Rio Grande do Sul, Brasil

[2] Departamento de Ciências da Saúde, Universidade de Santa Cruz do Sul - UNISC, Santa Cruz do Sul, Rio Grande do Sul, Brasil

[3] Programa de Pós-Graduação em Sistemas e Processos Industriais – UNISC, Santa Cruz do Sul, Rio Grande do Sul, Brasil

[4] Programa de Pós-Graduação em Promoção da Saúde – UNISC, Santa Cruz do Sul, Rio Grande do Sul, Brasil

gundel5@mx2.unisc.br, jordanakich@gmail.com, frozza@unisc.br, avalim@unisc.br, janinek@unisc.br, liapossuelo@unisc.br

Abstract. *The objective of this work was to develop a system that would help people with diabetes in their self-care, with information being made available through a conversational agent (Chatbot Dóris). The knowledge base of Chatbot Dóris was developed based on the need of the public with diabetes: Food, Food Labels, Physical Activity and Diabetic Foot. The application developed is called "Cuidado Diabetes", accessible both on the web and on a mobile device. Validation was carried out with specialists in the health area who followed the development of the system and with the target audience of the research, people with diabetes.*

Resumo. *O objetivo deste trabalho foi desenvolver um sistema que auxilie as pessoas com diabetes no seu autocuidado, com informações sendo disponibilizadas por meio de um agente conversacional (Chatbot Dóris). A base de conhecimento do Chatbot Dóris foi desenvolvida baseando-se na necessidade do público com diabetes, abordando: Alimentação, Rótulos de alimentos, Atividade Física e Pé Diabético. O aplicativo desenvolvido denomina-se "Cuidado Diabetes", acessível tanto pela web como por dispositivo móvel. A validação foi realizada com especialistas da área da saúde que acompanharam o desenvolvimento do sistema e com o público-alvo da pesquisa, pessoas com diabetes.*

1. Introdução

A Diabetes *Mellitus* (DM) é uma doença crônica não transmissível, caracterizada pela elevação da glicose no sangue (hiperglicemia), resultante da deficiência na ação e/ou produção da insulina. Por ser uma doença crônica, causa complicações a longo prazo se não houver um tratamento adequado, que inclui uma série de cuidados e recomendações, e tem como base central o autocuidado. O autocuidado refere-se à capacidade dos indivíduos de preservar a sua vida, sendo ele o agente ativo na gestão de seus próprios cuidados em saúde (BRASIL, 2013; SBD, 2022; WHO, 2020a).

Segundo a Federação Internacional de Diabetes (IDF - *International Diabetes Federation*), 537 milhões de pessoas estavam vivendo com DM em 2021 no mundo, representando um crescimento de 16% nos últimos dois anos e com previsão de chegar a 643 milhões até 2030, o que faz desta patologia a emergência sanitária que mais cresce mundialmente. O Brasil ocupava em 2021 o sexto lugar na relação dos dez países com maior número de pessoas com diabetes, com 15,7 milhões de acometidos, representando um aumento de 26,61% nos últimos dez anos (IDF, 2021).

No início do ano de 2020, a infecção viral denominada covid-19 assolou todo o mundo, com características clínicas variáveis, incluindo infecções assintomáticas, infecções leves, infecções agudas e óbito (WHO, 2020b). Neste cenário, as pessoas com diabetes, em função da sua condição de saúde, estavam mais propensas a vivenciar sintomas de maior gravidade, podendo necessitar de internação hospitalar para receber cuidados e tratamento intensivo (DOWN, 2020). Ainda, os protocolos de segurança, como o isolamento social, adotados pelas entidades de saúde para contenção da propagação do vírus, também trouxeram outros problemas para esta população, como descompensação clínica, depressão e ansiedade, que foram agravados, especialmente, nos grupos já fragilizados, incluindo os idosos e pessoas com comorbidades (associação de duas ou mais doenças), como os diabéticos (BANERJEE; CHAKRABORTY; PAL, 2020).

Desta forma, neste período de pandemia, a inserção digital ocorreu em muitos setores, e na área da saúde, as tecnologias se mostraram eficientes modelando os conhecimentos técnico-científicos em ferramentas por meio das quais a atenção e os cuidados em saúde pudessem ser prestados à população (GOLINELLI *et al.*, 2020; PETRACCA *et al.*, 2020). Neste sentido, o presente estudo levantou a hipótese de que o uso de sistemas inteligentes voltados para a atenção à saúde nas doenças crônicas, como é o caso da diabetes, poderia auxiliar no autocuidado destes indivíduos, principalmente, em períodos em que o acesso aos serviços de saúde é reduzido. Utilizando técnicas inteligentes, como, por exemplo, *chatbots* para coleta de dados, e Processamento de Linguagem Natural (PLN) para estabelecer um processo de interação mais natural, seria possível auxiliar no monitoramento de indivíduos com diabetes e indicar ações de autocuidado, a fim de melhorar a sua qualidade de vida. Os *chatbots* são agentes conversacionais que realizam interações homem computador na forma de diálogos (ADAMOPOULOU; MOUSSIADES, 2020). Já a PLN estuda as interações entre a máquina e a linguagem humana (MANNING; SCHÜTZE, 1999).

Assim, o objetivo principal deste trabalho foi desenvolver um sistema inteligente baseado em *chatbot* para auxílio ao autocuidado de indivíduos com diabetes *mellitus*.

O artigo está organizado nas seguintes seções: a seção 2 apresenta alguns trabalhos relacionados ao tema da pesquisa; a seção 3 descreve o método definido para a pesquisa realizada; a seção 4 apresenta os resultados atingidos; a seção 5 aborda a conclusão.

2. Trabalhos relacionados

O presente estudo integrou dois trabalhos acadêmicos desenvolvidos pelos autores: projeto de pesquisa intitulado "Avaliação da interferência da covid-19 na saúde de indivíduos diabéticos e proposta de aplicativo para o autocuidado", em nível de mestrado da autora Jordana Kich, no Programa de Pós-graduação em Promoção da Saúde (PPGPS) da Universidade de Santa Cruz do Sul (UNISC), cuja defesa foi realizada no ano de 2023; e projeto de pesquisa intitulado "Sistema baseado em *chatbot* para auxiliar no autocuidado de indivíduos com diabetes *mellitus*", em nível de graduação em Engenharia da Computação do autor Mateus Elias Gundel, da Universidade de Santa Cruz do Sul (UNISC), cuja apresentação foi realizada no ano de 2022.

Para a busca de trabalhos relacionados ao tema, foram realizadas revisão em base de lojas de aplicativos, no instituto de propriedade industrial, e a bibliometria.

2.1 Aplicativos em bases de lojas

A partir de um levantamento realizado nas lojas do Google – Play Store, e da Apple – App Store, observou-se que os aplicativos existentes, em sua maioria, são para registro e acompanhamento dos dados como glicemia, aplicação de insulina, alimentos ingeridos e registo de atividades realizadas, como consultas, exames e exercícios físicos. Alguns permitem a geração de gráficos e relatórios e opção de compartilhamento das informações. Os aplicativos utilizam menus com navegação em textos e imagens para mostrar as informações, possuindo o seu *download* gratuito, mas alguns com funcionalidades pagas. A busca foi realizada utilizando quatro termos, em português e inglês: 1) autocuidado/self-care; 2) autogerenciamento/self-management; 3) diabético(s)/diabetic(s) e 4) diabete(s). Foram considerados apenas aplicativos com linguagem ou tradução para o português e com média de avaliações superior ou igual a 4 estrelas, tendo como resultado 29 aplicativos.

2.2 Instituto Nacional de Propriedade Industrial (INPI)

A busca foi realizada no site do INPI no campo "Programas de Computador", para verificar a existência de registros relacionados aos termos: diabetes (14 resultados); diabético (5 resultados); autocuidado e autogerenciamento (nenhum resultado). Foram encontrados disponíveis para *download* apenas dois destes aplicativos, ambos relacionados ao pé diabético.

2.3 Bibliometria

A busca foi realizada nas bases de dados Scopus, PubMed e Science Direct com os termos *diabetes, machine learning* e *chatbot* no período de 2018 a 2022. Os artigos foram escolhidos utilizando o método PRISMA (LIBERATI *et al*., 2009). Na fase de triagem foram selecionados os artigos que continham os termos de busca definidos. Na fase de elegibilidade foi realizada a análise do título, palavras-chaves e resumo dos

artigos, e por fim, na etapa de inclusão, foram escolhidos os 3 artigos que mais se relacionavam ao tema abordado. A partir da comparação e estudo dos artigos, foi possível observar que já existem pesquisas propondo auxiliar o diagnóstico, controle e cuidado da Diabetes *Mellitus*, que, na maioria, faz uso de algoritmos de Aprendizado de Máquina ou Inteligência Artificial, bem como utiliza *chatbots* para realizar a interação com o usuário, mostrando que é um assunto crescente, e uma preocupação válida atualmente. Destes trabalhos, dois são de 2020 e um deles de 2019, mostrando que é um assunto atual, mas sem tantos trabalhos depois do início da pandemia, que intensificou o autocuidado para pessoas com diabetes. Assim, o autocuidado ainda pode ser melhorado com o uso de sistemas computacionais, buscando melhor assertividade nas indicações clínicas (BALI *et al*., 2019; CHAKI *et al*., 2020; SOWAH *et al*., 2020).

A Tabela 1 apresenta um comparativo dos trabalhos relacionados.

Tabela 1. Trabalhos Relacionados

Artigo	Objetivo	Área de aplicação	Parâmetros verificados	Técnicas utilizadas
(CHAKI *et al*., 2020)	Detectar o diagnóstico, realizar controle e tratamento da *Diabetes Mellitus* com aprendizado de máquina	Saúde, Aprendizado de máquina, Inteligência artificial, chatbots	Bases de dados, classificações em Aprendizado de Máquina e métodos de diagnóstico e métricas de desempenho	Redes neurais convolucionais
(SOWAH *et al*., 2020)	Desenvolver um sistema para melhorar o gerenciamento da Diabetes utilizando Aprendizado de Máquina	Saúde, Aprendizado de máquina	Precisão dos modelos preditivos e validação através de métodos de entropia cruzada	Algoritmo KNN, redes neurais utilizando Tensorflow, chatbot usando algoritmos cognitivos
(BALI *et al*., 2019)	Desenvolver uma aplicação para o diagnóstico daDiabetes com o uso de um chatbot	Saúde, chatbots	Precisão entre os classificadores e os modelos de classificação	RASA NLU, React UI, Python, modelos de previsão, modelos de Naive Bayes e Árvore de Decisão
Este trabalho	Desenvolver um sistema inteligente para auxílio ao autocuidado de indivíduos com *Diabetes Mellitus*	Saúde, chatbots	Avaliação e validação com profissionais da área da saúde e usuários com *diabetes*	Sistema web com integrações com serviços de linguagem natural e Chatbot.

3. Métodos

A abordagem do *Design Science Research* (DSR) foi escolhida para guiar o desenvolvimento do sistema inteligente, pois é considerado um método que fundamenta e operacionaliza a condução da pesquisa para desenvolvimento de inovações. O método é estruturado por etapas, as quais constituem estratégias e ações metodológicas que podem ser adaptadas de acordo com as necessidades de cada pesquisa (DRESCH; LACERDA; ANTUNES, 2015). Indica-se, a seguir, as fases de desenvolvimento do aplicativo:

• Identificação do problema: "De que forma é possível disponibilizar informações úteis aos indivíduos com Diabetes *Mellitus* por meio de um sistema baseado em *chatbot*?".

• Conscientização do problema: Formalização de todos os componentes do problema a ser solucionado através de dados transversais obtidos mediante entrevista com diabéticos tipo 2.

• Revisão em bases de dados: Busca na base de dados do Instituto Nacional de Propriedade Industrial, para identificar o que se tem produzido na temática, considerando que muitos artefatos ainda não foram divulgados em periódicos, pois aguardam a liberação do pedido de patente ou registro de software, busca nas bases das lojas de aplicativos Play Store e Apple Store, e bibliometria.

• Identificação dos aplicativos e configuração das classes de problemas: Análise dos aplicativos já disponíveis no mercado e determinação das classes de problema que o aplicativo irá se propor a auxiliar.

• Proposição, projeto, desenvolvimento e avaliação do artefato: Envolve todas as etapas de desenvolvimento do sistema baseado em *chatbot*, para interação com o público-alvo do sistema, com modelagem, implementação, testes e avaliação.

• Explicitação das aprendizagens: Descrição dos limites da utilização do aplicativo, bem como das condições de uso no ambiente real.

• Conclusões: Descrição dos objetivos e metas alcançados e os não alcançados.

• Generalização para uma classe de problemas: Indicação de uso do aplicativo para solucionar problemas elencados anteriormente.

• Comunicação dos Resultados: Relato dos resultados da pesquisa.

4. Resultados

Como resultado, foi desenvolvido um PWA (*Progressive Web App*), no qual o indivíduo com Diabetes Mellitus acessa uma interface por meio do seu celular ou computador, navega em algumas informações e inicia uma conversa com o *chatbot* Dóris. A Dóris fornece informações sobre a diabetes e orientações sobre os cuidados necessários, auxiliando na tomada de decisões do indivíduo. As informações e orientações são em relação à alimentação, às atividades físicas possíveis de serem realizadas, os rótulos do alimento e como entendê-los e informações sobre o pé diabético.

O sistema foi definido e desenvolvido em cinco módulos que se integram para gerar toda a experiência de uso para os usuários, sendo eles:

• Interface do sistema: A parte que realiza a interface e interação com o usuário, que possui instruções de uso, o *chat* para o contato com a Dóris, e realiza a captação de métricas de uso dos usuários. O Vue.js foi a base para o desenvolvimento do *front-end* da aplicação.

• Aplicação: É a parte que recebe as requisições e ações dos usuários e realiza o seu processamento. As ações possíveis referem-se à interação com o *chatbot*, com o sistema de texto para fala e com o salvamento de informações no banco de dados. O sistema é um PWA (*Progressive Web App*), no qual o indivíduo com *Diabetes Mellitus* acessa uma interface através do seu celular ou computador, navega em algumas informações e inicia uma conversa com o *chatbot* Dóris.

• Banco de dados: Responsável por armazenar todos os dados e informações inseridas na aplicação pelos usuários. Foi utilizado o PostgreSQL.

• *Chatbot*: Responsável por armazenar, processar e responder às interações dos usuários, bem como entender as intenções dos usuários e relacionar com os fluxos possíveis de informações previamente configurados.

• Texto para Fala: Realiza a leitura dos textos recebidos, convertendo-os do *chatbot* para voz em português, utilizando os algoritmos já treinados de PLN (Processamento de Linguagem Natural) dos serviços do Watson e AWS.

Toda a interface do sistema foi desenvolvida com base na estruturação das telas realizada por uma especialista em experiência do usuário, toda a estruturação e desenho das telas foi realizado na plataforma Figma antes do desenvolvimento.

A Figura 1 demonstra as telas desenhadas.

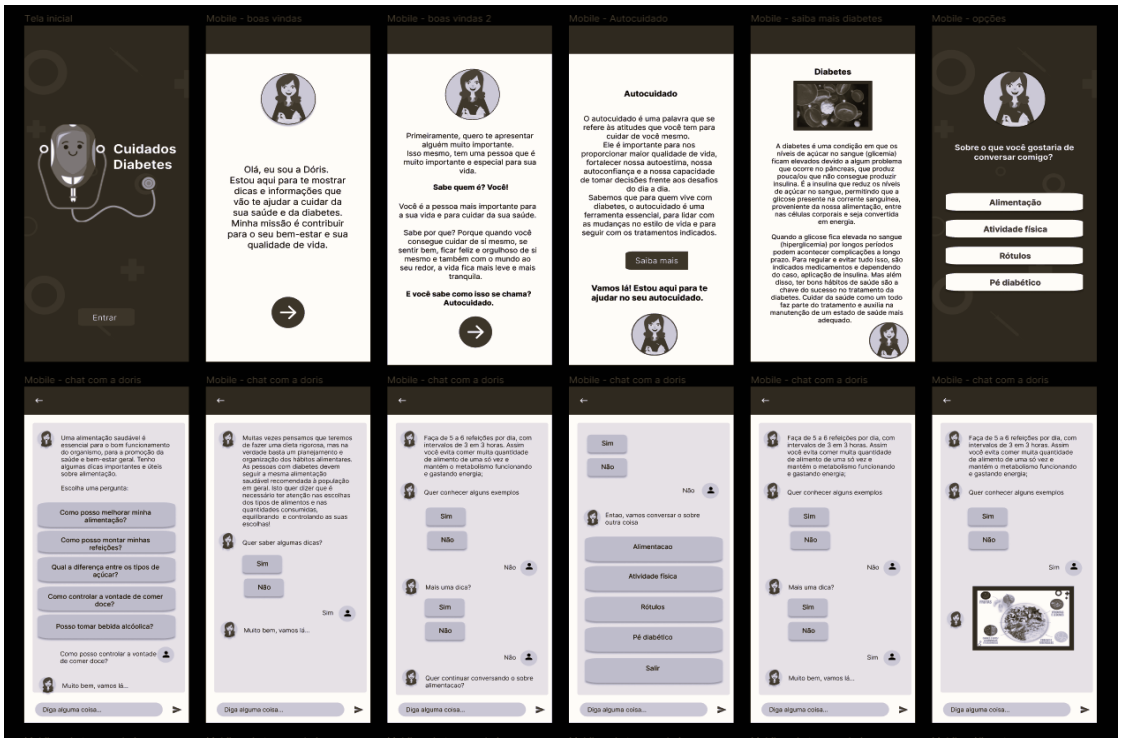


Figura 1. Telas desenvolvidas no sistema

Após o desenho e discussão da interface, o *front-end* foi desenvolvido utilizando Vue.js, que se comunica com o *back-end* python através de chamadas API Rest para a busca dos dados a serem exibidos e interações que são realizadas com o *chatbot*.

Para o desenvolvimento do *back-end*, foi utilizado o Python com o seu *framework* para desenvolvimento web FastAPI, o qual monta os endpoints da API Rest e realiza a comunicação com o bando de dados Postgresql. O Python também é responsável por guardar as informações das conversas realizadas pelos usuários, o envio e o processamento da conversação para o Watson, e o processamento de texto para fala, que é realizado após receber uma resposta do *chatbot*.

A Figura 2 ilustra o fluxo da informação quando ocorre a comunicação com o *chatbot*: i) o dado é coletado pela interface, chegando na aplicação; ii) a aplicação envia para o *chatbot*; iii) o *chatbot* devolve a próxima mensagem, que é enviada para o serviço de conversão de texto para fala; iv) a mensagem retorna como áudio para a aplicação; v) a aplicação une estes dados e devolve para a interface exibir para o usuário.

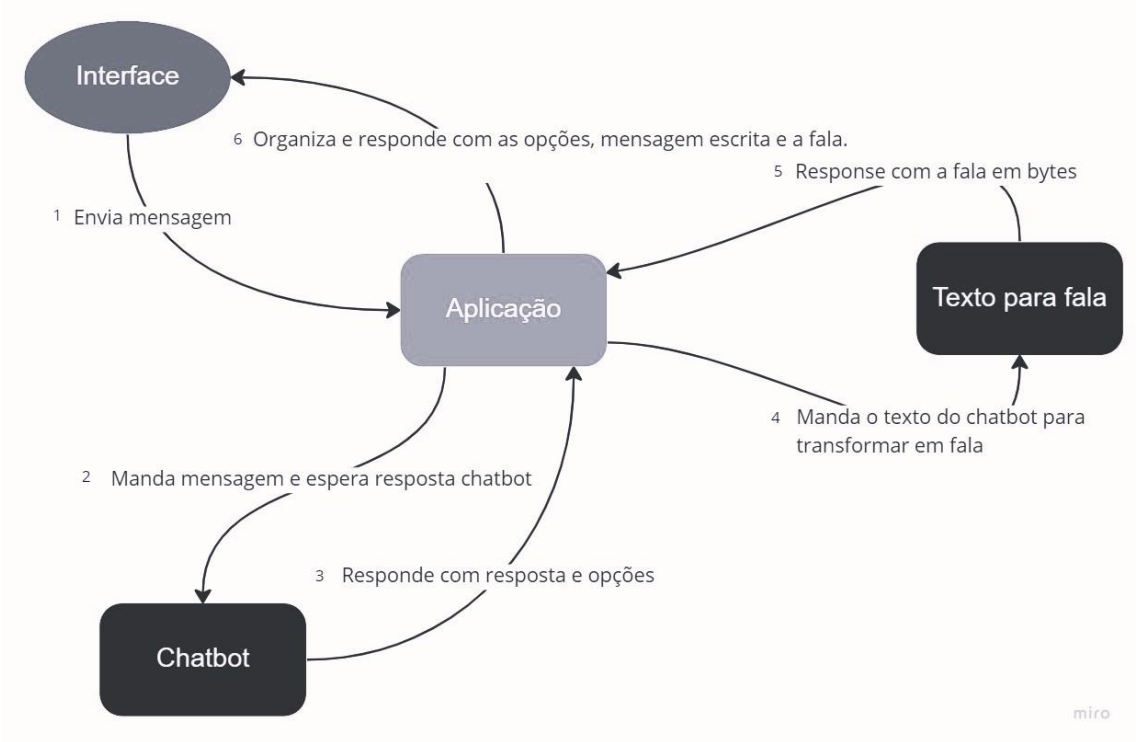


Figura 2. Fluxo da informação no sistema.

A Dóris é um *chatbot* que interage com usuários por meio de texto escrito e áudio, com expressões faciais, e utilizando os recursos de PLN do IBM Watson. Sua arquitetura possui três módulos: i) o módulo Perceptivo, responsável por receber como entrada a interação realizada pelo usuário, que poderá ser por meio de áudio ou texto e,a pós realizar o processo de entrada de informação, ocorre o encaminhamento de dados para o módulo cognitivo; ii) o módulo Cognitivo realiza os processos de tomadas de decisão, ou seja, o agente define as ações que serão tomadas futuramente; iii) o módulo Reativo é responsável por responder, por meio de áudio ou texto, a decisão tomada pelo módulo cognitivo (COSSUL *et al.*, 2018).

Para o desenvolvimento do *chatbot* Dóris, foi utilizada a plataforma IBM Watson Assistant em seu plano grátis, que permite a criação dos fluxos de conversas e possui integração através de sua API. Como conteúdo para a elaboração dos fluxos na plataforma, foram definidos quatro assuntos principais, os quais mais foram relatados como dificuldades por pacientes diabéticos que se dispuseram a responder uma pesquisa, que teve 174 entrevistas realizadas com o público-alvo do aplicativo.

Para a interação com os usuários, foram gerados quatro fluxos de conversas distintos para cada um dos assuntos de maior dificuldade relatados. Cada um deles foi discutido e validado com profissionais das áreas. Por exemplo, sobre exercícios físicos, houve auxílio de um profissional de Educação Física para validar as informações disponibilizadas no aplicativo.

Todo o fluxo de conversação com o *chatbot* ocorre através de escolhas e direcionamentos que o usuário informa ao *chatbot*. Há opções em caixas de diálogos a serem selecionadas em cada tema, como sugestões para seguir os fluxos definidos de conversação, ou então, quem estiver utilizando o sistema pode inserir uma dúvida diretamente na caixa de texto. A Figura 3 apresenta todos os fluxos definidos, sendo Alimentação, Atividade Física, Rótulos e Pé diabético. A Figura 4 ilustra um dos fluxos específicos, o de Atividades Físicas.

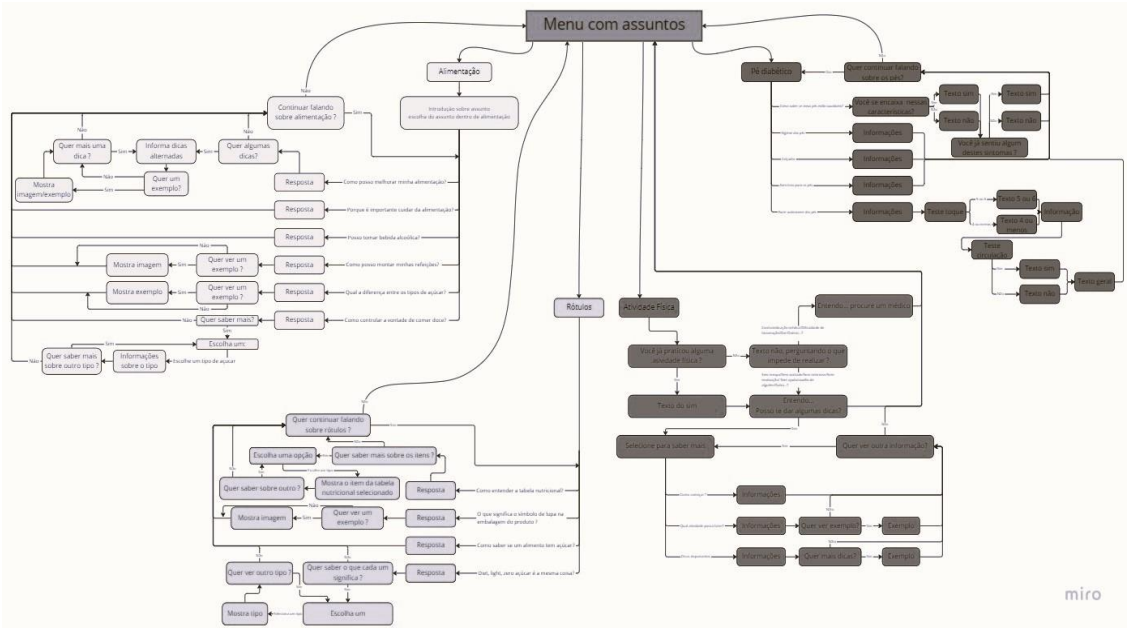


Figura 3. Fluxos de interação com a Dóris

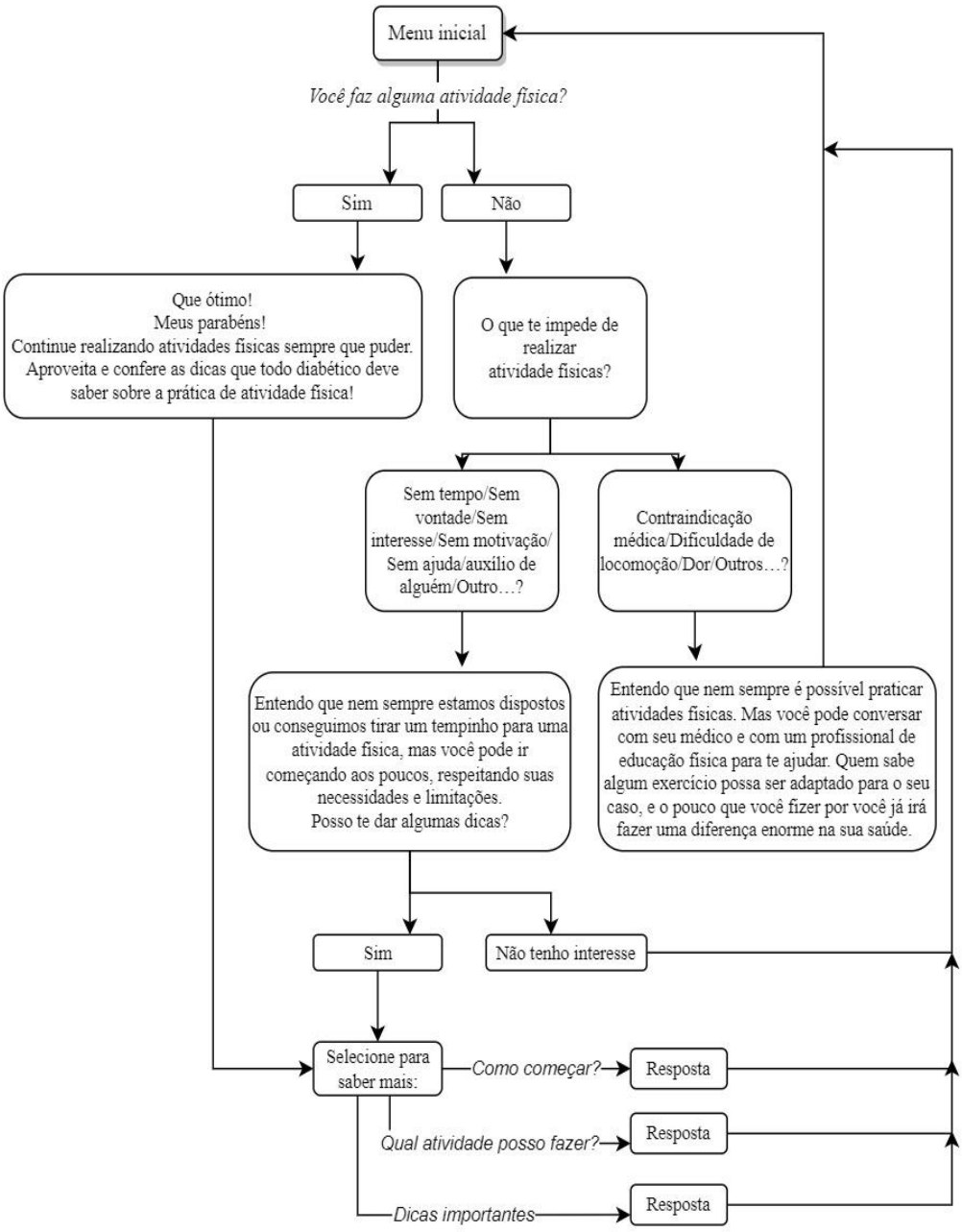


Figura 4. Fluxo de atividade física

Na interação com a Dóris, as respostas fornecidas, seguindo o fluxo, são convertidas para áudio, e podem ser tanto lidas na interface do sistema, quanto ouvidas. Esta conversão é realizada pelos serviços IBM Watson Text to Speech ou Amazon Poly.

5. Conclusão

O aplicativo “Cuidados Diabetes” foi desenvolvido e validado, e está disponível na web e para dispositivos móveis, com o objetivo de auxiliar as pessoas com diabetes no seu autocuidado, fornecendo informações úteis através de uma interface simples e fácil de ser utilizada. A integração com a *chatbot* Dóris e conversores de texto para fala, torna a interação do usuário com o sistema facilitada, garantindo melhor experiência para os usuários.

Uma pesquisa com um grupo de 174 pessoas com diabetes foi realizada, a fim de definir os requisitos a serem contemplados pelo aplicativo, além de questões para um *design* agradável e útil para os usuários. No entanto, ressalta-se a importância do contato regular dos indivíduos com diabetes com seu médico, sendo o uso do aplicativo um auxílio para o autocuidado.

O aplicativo desenvolvido é uma inovação tecnológica na área da saúde por ser o primeiro aplicativo para este público, agregando um agente virtual (*chatbot*) com interação a partir de texto e áudio. A escolha de utilizar *chatbot,* além de garantir o diferencial deste aplicativo, sustenta a ideia de uma conversação com o usuário, proporcionando um dos elementos fundamentais para a construção de habilidades para o autocuidado - a informação.

Referências

Adamopoulou, E; Moussiades, L. (2020). Chatbots: History, technology, and applications. Machine Learning with Applications.

Bali, M. et al. (2019). An automated, yet interactive and portable DB designer. In: International Journal of Recent Technology and Engineering (IJRTE).; 8:6334–6340.

Banerjee M, Chakraborty S, Pal R. (2020). Diabetes self-management amid covid-19 pandemic. Diabetes Metab Syndr.;14(4):351-354.

Brasil (2013). Ministério da Saúde. Cadernos de Atenção Básica: estratégias para o cuidado da pessoa com doença crônica – Diabetes Mellitus, n. 36. Brasília, 2013.

Chaki, J. et al. (2020). Machine learning and artificial intelligence-based diabetes mellitus detection and self-management: A systematic review. In: Journal of King Saud University – Computer and Information Sciences.

Cossul, D.; Frozza, R.; Fagundes, B. J.; Ferreira, G.; Kipper, L. M.; 54 Witczak, M. V. C. Evolução do agente pedagógico emocional Dóris em um ambiente virtual de aprendizagem. In: Editora Poisson. (Org.). Gestão da Produção em Foco.1ªed. Belo Horizonte: Poisson, Vol. 24, p. 28-38. 2018.

Down, S. (2020). Covid-19 and diabetes. In: Journal of Diabetes Nursing 24.

Dresch, A, Lacerda DP, Antunes JAVJ (2015).Desing Science Research: método de pesquisa para avanço da ciência e tecnologia. Porto Alegre: Boolmann.

IDF. International Diabetes Federation (2021). IDF Diabetes Atlas 10th Edition, Belgium, 2021.

Golinelli, D. et al. (2020). Adoption of digital technologies in health care during the covid-19 pandemic: systematic review of early scientific literature. *Journal of Medical Internet Research*.;22(11): 1-23.

Liberati, A. et al. The prisma statement for reporting systematic reviews and meta-analyses of studies that evaluate health care interventions: Explanation and elaboration. 2009.

Manning, C.D., Schutze, H. (1999). Foundations of statistical natural language processing. MIT Press.

Petracca, F. et al. (2020). Harnessing digital health technologies during and after the covid-19 pandemic: context matters. Journal of Medical Internet Research.;22(12): 1-7.

SBD. Sociedade Brasileira de Diabetes (2022). *Diretrizes da Sociedade Brasileira de Diabetes.*

Sowah, R.A. et al. (2020). Design and development of diabetes management system using machine learning. In: International Journal of Telemedicine and Applications.

WHO. World Health Organization (2020a). *Self-care health interventions*.

WHO. World Health Organization. (2020b). *Naming the coronavirus disease (COVID-19) and the virus that causes it*.