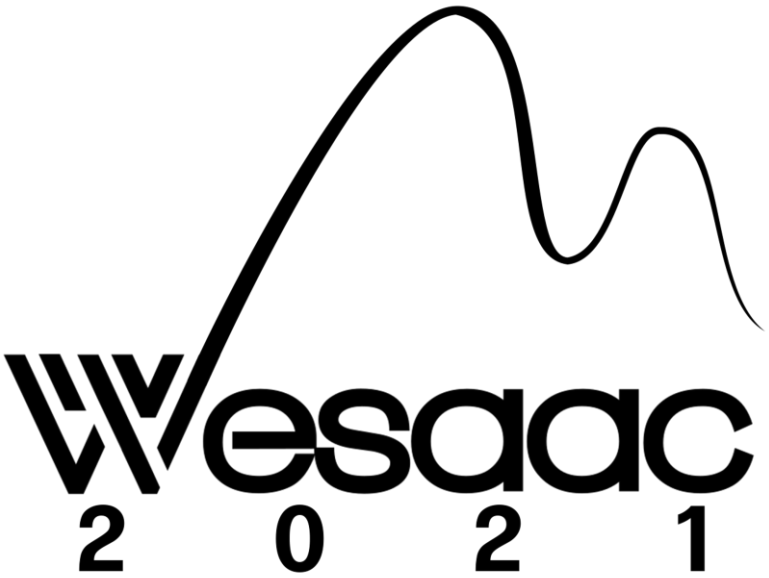


Anais do XV Workshop-Escola de Sistemas de Agentes, seus Ambientes e apliCações

— WESAAC 2021—

Organizado por

Carlos Eduardo Pantoja
Maiquel de Brito
Maicon Zatelli

Rio de Janeiro, 10 - 12 de Agosto de 2021.

(Realizado no formato online)

Workshop-Escola de Sistemas de Agentes, seus Ambientes e apliCações —XV WESAAC / Pantoja, C. E.; Brito, M. de; Zatelli, M.; (Org). ANAIS.— — Rio de Janeiro, 2021.

188p. : il.

ISSN 2177-2096 DOI 10.5281/zenodo.5774181

1. Agentes Inteligentes. 2. Sistemas de Agentes de Software. 3. Ambientes para Agentes. 4. Aplicações de Agentes. I. Pantoja, C. E. V. II. Brito, M. III. Zatelli, M.

PREFÁCIO

Este documento contém os trabalhos apresentados na Décima Quinta Edição do **WESAAC** (*Workshop Escola de Sistemas de Agentes, seus Ambientes e apliCações*), realizado no formato online com o apoio da Universidade Federal de Santa Catarina (UFSC) e do Centro Federal de Educação e Tecnológica Celso Suckow da Fonseca (CEFET/RJ), entre os dias 10 e 12 de Agosto de 2021.

Gostaríamos de agradecer aos pesquisadores convidados, Andrei Ciortea, Tobias Al Ahlbrecht, Jürgen Dix e Rafael H. Bordini, os quais aceitaram o convite para palestrarem do WESAAC 2021. Igualmente agradecemos à Jomi F. Hübner, Niklas Fiekas e Tabajara Krausburg pelas oficinas ministradas. Agradecemos ao trabalho de todo comitê de programa na revisão dos artigos e igualmente ao comitê consultivo que foi fundamental durante todos os estágios do evento. Estendemos os agradecimentos aos apresentadores e à todos participantes do Workshop. Por fim, agradecemos ao apoio da Comissão de Voluntários formada por alunos da Ciência da Computação e da Automação Industrial do CEFET/RJ.

Rio de Janeiro, 2021.

Carlos Eduardo Pantoja
Maiquel de Brito
Maicon Zatelli

Organização e Realização

Organização

Organização Geral
Carlos E. Pantoja (CEFET/RJ)

Coordenação do Comitê do Programa
Maiquel de Brito (UFSC)
Maicon Zatelli (UFSC)

Comitê Consultivo
Anarosa Alves Franco Brandão (USP)
Diana Francisca Adamatti (FURG)
Gustavo Alberto Giménez Lugo (UTFPR)
Jerusa Marchi (UFSC)
Jaime Sichman (USP)
João Luis Tavares da Silva (UCS)
Jomi Fred Hubner (UFSC)
Mariela Inés Cortés (UECE)
Rafael Heitor Bordini (PUCRS)
Rejane Frozza (UNISC)
Ricardo Choren (IME)
Viviane Torres da Silva (UFF)

Comitê do Programa
Alessandro Ricci (Universidade de Bolonha, Itália)
Álvaro Moreira (UFRGS)
Ana Paula Lemke (IFRS)
Anarosa Alves Franco Brandão (USP)
André Pinz Borges (UTFPR)
Antonio Carlos Rocha Costa (PPGComp / FURG - PPGFIL / PUCRS)
Carlos Eduardo Pantoja (CEFET-RJ)
Cesar Tacla (UTFPR)
Clarissa Xavier (UFRGS)
Cleo Billa (FURG)
Diana Francisca Adamatti (FURG)
Eder Mateus Nunes Gonçalves (FURG)
Fernanda Alencar (UFPE)
Fernando de La Prieta (Universidade de Salamanca)
Fernando dos Santos (UDESC)
Francisca Raquel de Vasconcelos Silveira (IFCE)
Gabriel Ramos (UNISINOS)
Gleifer Vaz Alves (UTFPR)
Graçaliz Dimuro (FURG)
Guillermo Simari (Universidade do Sul - Bahia Blanca, Argentina)
Gustavo Augusto Lima de Campos (UECE)
Gustavo Giménez Lugo (UTFPR)
Jaime Sichman (USP)
Jerusa Marchi (UFSC)

João Luis Tavares da Silva (UniFTEC)
Jomi Fred Hubner (UFSC)
Luis Gustavo Nardin (Faculdade Nacional da Irlanda)
Maicon Zatelli (UFSC)
Maiquel de Brito (UFSC)
Mariela Cortés (UECE)
Marilton Sanchotene de Aguiar (UFPel)
Olivier Boissier (Minas Saint-Etienne, Instituto Henri Fayol, Laboratório Hubert Curien UMR CNRS 5516, França)
Rafael Cardoso (Universidade de Liverpool, Inglaterra)
Patricia Tedesco (UFPE)
Paulo Trigo (ISEL, Portugal)
Paulo Leitão (IPB, Portugal)
Rafael Bordini (PUCRS)
Raquel Barbosa (IFRS)
Rejane Frozza (UNISC)
Ricardo Choren (IME)
Ricardo Grunitzki (Instituto de Ciência e Tecnologia SIDIA Samsung)
Ricardo Azambuja Silveira (UFSC)
Sara Casare (USP)
Túlio Lima Basílio (IFRS)
Vera Maria Benjamim Werneck (UERJ)
Viviane Torres da Silva (IBM)

SUMÁRIO

ARTIGOS

ARTIGOS COMPLETOS

Does a Q-Learning NetLogo Extension Simplify the Development of Agent-based Simulations?

Eloísa Bazzanella,[1] **Fernando Santos**[1]

[1]Departamento de Engenharia de Software
Universidade do Estado de Santa Catarina (UDESC)
Ibirama – SC – Brazil

elobazzanella@gmail.com, fernando.santos@udesc.br

Abstract. *Agent-based modeling and simulation is a simulation paradigm that allows focusing on individuals, their interactions, and the complex behavior that emerges from them. Agent-based simulations are typically developed in simulation platforms that provide features related to agents. One such platform is NetLogo, to which a reinforcement learning extension was made available recently. The extension provides commands for using the Q-Learning algorithm, but no evaluation on whether it simplifies the development of simulations is available. This paper presents a quantitative evaluation on using the extension in two simulations: the classic cliff walking problem; and a real-world, adaptive traffic signal control (ATSC) simulation. Results show that the size of simulations source code developed using the extension is smaller than those developed without using it, giving evidence that the extension simplifies the development of simulations*

1. Introduction

Computer simulations have been used to study phenomena and support decision making. For example, traffic simulations assist in the design of transport infrastructure, and evacuation simulations assist the design of stadiums that can be quickly evacuated in emergency situations. It is not trivial to develop analytical models that simulate the behavior of these complex systems. Producing them is a task that has been extensively investigated in the context of agent-based modeling and simulation (ABMS), a simulation paradigm that makes use of agents to reproduce and study a phenomenon under investigation [6].

The ABMS paradigm allows focusing on the individuals (agents) and the implications resulting from their behavior and from the interaction among them. Artificial intelligence techniques can be incorporated into agents to enable them to adapt to changes in themselves or in the environment [6]. One of such techniques is reinforcement learning, that enable agents to learn through experience. A well-known reinforcement learning method is Q-Learning [16].

Although agent-based simulations can be developed with general-purpose languages such as Java, there are languages and platforms specifically tailored for developing agent-based simulations. These platforms provide agent and simulation features that simplify the development of agent-based simulations. A popular agent-based simulation platform is NetLogo [17]. Statistics show that, among all the simulations available in the CoMSES [5] repository in 2020, 33.6% were developed in NetLogo. It provides its own programming language, with high-level commands for operations frequently required in

agent-based simulations. There are modules (called *extensions*) that provide additional commands and features for developing simulations. Recently, an extension that provides commands for using the Q-Learning method in simulations was made available.[1] However, no evaluation regarding whether this extension simplifies the development of agent-based simulations is available.

In this paper we present a quantitative evaluation conducted to investigate whether the existing NetLogo Q-Learning extension simplifies the development of simulations. The evaluation considered two simulations: the classic cliff walking problem; and an adaptive traffic signal control (ATSC) simulation. Both simulations were implemented using the Q-Learning extension, and their source code were compared to existing Q-Learning implementations without the extension. For the comparison we used the *lines of code* software size metric. Results show that the size of simulations source code developed using the extension are 13.72% (cliff walking) and 47.06% (ATSC) smaller than source codes developed without using it, giving evidence that the existing NetLogo Q-Learning extension simplifies the development of simulations.

The remaining of this paper is organized as follows. Section 2 presents the required background on ABMS, reinforcement learning, and the existing NetLogo Q-Learning extension. In Section 3 we describe the conducted quantitative evaluation, which includes: the adaptive traffic signal control simulation and the reasons why it was selected for our study; the procedure and metric adopted in the evaluation; and the obtained results. Finally, Section 4 presents concluding remarks and future work.

2. Background

This section presents concepts related to our quantitative evaluation. We first introduce the ABMS paradigm. Then, we describe reinforcement learning, in particular the Q-Learning algorithm. Finally, we present the existing Q-Learning NetLogo extension and detail how it is used to incorporate the Q-Learning method into agent-based simulations.

2.1. Agent-based Modeling and Simulation

Agent-based modeling and simulation (ABMS) is a simulation paradigm that makes use of agents to reproduce and study a phenomenon under investigation [6]. In the ABMS paradigm, a simulation is composed of agents that can interact with each other and are situated on an environment they can perceive and modify through their actions. Agents have two fundamental properties: they are autonomous in their decision making towards their objectives; and they are able to interact with each other. Agents are able to decide based on logical deductions, just like human reasoning, or via deduction combined with a decision-making mechanism. Additionally, agents can have different skills and architectures, and they can perform distinct roles [18].

ABMS has been used to model and run simulations in many application domains, such as traffic, ecology, economics and epidemiology [9]. In these cases, ABMS has been chosen as the simulation paradigm because it is able to incorporate the inherent complexity of individual behavior and interactions in real-world scenarios [9]. Artificial intelligence techniques can be incorporated into agents to improve their behavior. Bazzan and

[1] `https://github.com/agentbasedsimulations/qlearning-netlogo-extension`

Klügl [3], for example, enumerate studies on applying agents endowed with intelligent abilities to improve traffic and transportation systems.

Agent-based simulations can be developed with general-purpose programming languages such as Java and Python. However, there are agent-based simulation platforms which provide agent and simulation related features that ease the development of simulations. One of such platforms is NetLogo [17]. NetLogo is a programmable modeling environment that allows simulating natural and social phenomena. It provides a programming language, with high-level commands for features frequently required in agent-based simulations, such as for setting up the environment and moving agents through it. Additional commands and features are provided via modules (called *extensions*). One of such extensions, which is detailed next in Section 2.3, provides commands for using the Q-Learning reinforcement learning method.

2.2. Reinforcement Learning

Humans usually learn through interactions with the environment. Throughout life, these interactions are an important source of knowledge about the environment and about themselves. Learning through interaction is a fundamental idea of almost all theories of learning and intelligence [15].

There are learning methods that can be incorporated into agents to enable them to learn over time. Reinforcement learning (RL) is one of such methods, in which an agent learn through experience. According to Monteiro and Ribeiro [11], RL is "a computational learning paradigm in which a learning agent seeks to maximize a performance measure based on the reinforcement (reward or punishment) it receives when acting on an unknown environment". RL methods often adopt a state-based representation of the environment. Periodically, the agent must select which action to execute. When an action is executed, the agent receives a reward signal based on the outcomes of previous states and actions. By observing the reward received when executing different actions on different states for a period of time, an agent running a RL method is able to learn an optimal control policy (one that maximizes the expected reward).

Q-Learning [16] is one of the available RL methods. Q-Learning works by estimating optimal state-action values, named *Q-values*. Each *Q-value* is a numerical estimator of quality for a given pair of state and action. Therefore, a *Q-value* $Q(s, a)$ represents the maximum discounted sum of future rewards the agent can expect to receive if it starts in state s, choose action a, and then continues to follow an optimal policy.

Q-Learning maintains a data structure called *Q-table*, which stores a *Q-value* for every pair (s, a). As the agent acts on the environment, $Q(s, a)$ values are updated to consider the reward signal r received when the action a is executed in state s. In addition to a set S of states and A of actions, Q-Learning has two parameters: a learning rate α and a discount factor γ. The first specifies how much of the agent expertise is replaced by the outcomes of recently experienced actions. The last describes the agent preference for immediate over future rewards.

Figure 1 shows the Q-Learning algorithm. Q-Learning lets the agent run for a number of episodes. In each episode the agent starts from an initial state and goes through countless states until it reaches a terminal state. At each step, the agent selects an action a for the current state s using a selection policy. Greedy action selection policies exploit

Input: S, A, $\alpha \in (0,1)$ $\gamma \in (0,1)$
foreach $episode$ **do**
  $s \leftarrow initial\_state$
  repeat
    choose an action a for state s using a selection policy (e.g., ϵ-greedy)
    perform the action a
    observe the new state s' and the reward r received
    update *Q-table*:
      $Q(a,s) \leftarrow Q(a,s) + \alpha(r + \gamma\, max_a\, Q(a',s') - Q(a,s))$
    $s \leftarrow s'$
  until $s \neq terminal\_state$;
end

Figure 1. Q-Learning algorithm. Adapted from Russel and Norvig [13]

the current agent expertise and increase immediate reward. However, with such greedy policies the agent may not explore some actions and end up learning sub-optimal control policies. An alternative selection policy commonly used with Q-Learning to avoid sub-optimal policies is the ϵ-greedy [15]. With ϵ-greedy the agent behaves greedily most of the time, but with probability ϵ it selects an action at random. The ϵ value is decreased by an ϵ-decay rate to let the agent exploit the optimal policy after a learning period.

Following the Q-Learning algorithm, the chosen action is performed by the agent, which observes the new state s' and the reward r received. The $Q(a,s)$ value is then updated in the *Q-table* according the Q-Learning update rule, which considers the learning rate α, the reward r, the discount factor γ, and the expected future reward provided by following the optimal policy for the new state s' onward (given by the action a' that maximizes $Q(a',s')$). Finally, the current state s is updated and the learning process is repeated until the agent reaches a terminal state.

2.3. Q-Learning NetLogo Extension

An *extension* is a NetLogo module that extends its programming language with additional commands and features. Recently, a NetLogo extension with a ready-to-use implementation of the Q-Learning algorithm was made available [7]. The extension provides additional NetLogo commands to specify the following Q-Learning elements: states, actions, the reward, the action selection policy, the end of episode clause, the episode reset procedure, learning rate, and discount factor. In addition, the extension also provides commands to execute the Q-Learning algorithm when the agent is expected to act and learn. The extension, called *Q-Learning Extension*, can be installed via the NetLogo Extension Manager and its documentation is available online.[1]

We describe how the Q-Learning extension works by means of the classic *cliff walking* problem [15] shown in Figure 2. In this problem the environment is a grid, in which a subset of cells represents a cliff. The goal of a *Walker* agent is to learn how to go from the starting cell S to the goal cell G without falling off the cliff (gray cells). The agent can move up, down, right, and left. A learning episode ends when the agent reaches the goal cell or falls off the cliff. If the agent falls off, its reward is -100; otherwise, its reward is -1 for each cell it has visited.

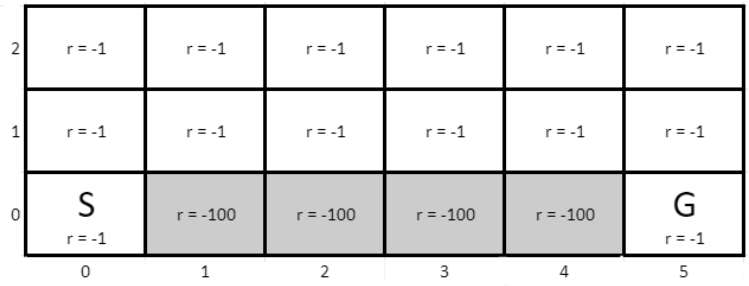


Figure 2. Cliff walking problem. Adapted from Sutton and Barto [15]

The NetLogo source code developed with the Q-Learning extension for the cliff walking problem is partially shown in Figure 3.[2] The `setup` procedure (lines 1–12) is where the simulation is set up. The `ask` block in lines 3–11 requests that all `Walker` agents execute a few commands provided by the Q-Learning extension in order to set up the Q-Learning algorithm. These commands are described next.

```
1  to setup
2      clear-all
3      ask Walkers [
4          qlearningextension:state-def ["xcor" "ycor"]
5          (qlearningextension:actions [goUp] [goDown] [goLeft] [goRight])
6          qlearningextension:reward [rewardFunc]
7          qlearningextension:learning-rate 0.4
8          qlearningextension:discount-factor 0.2
9          qlearningextension:action-selection "e-greedy" [0.8 0.99]
10     ]
11 end
12
13 to go
14     ask Walkers [
15         qlearningextension:act
16         qlearningextension:learn
17     ]
18 end
```

Figure 3. Cliff Walking Simulation Implemented with the Q-Learning Extension

The `qlearningextension:state-def` command (line 4) specifies the state representation. This command takes as argument a list of agent variables (attributes) whose values characterize a state. In the cliff walking, the state is characterized by the x and y coordinates of the agent position (stored by the `xcor` and `ycor` agent attributes).

To specify the actions considered by the Q-Learning algorithm, the extension provides the `qlearningextension:actions` command (line 5). This command takes as argument a list of NetLogo procedures, each of them corresponds to an action the agent can execute. In the cliff walking simulation, `goUp`, `goDown`, `goLeft`, and `goRight` procedures are implemented by the developer to move the agent towards these directions.

[2] We refer the reader to Kons and Santos [8] for the complete source code of the *cliff walking* simulation.

To specify the reward received by the agent whenever it acts, the extension provides the `qlearningextension:reward` command (line 6). The developer only needs to inform the procedure that computes and returns the reward value. To specify the learning rate and the discount factor, the extension provides the commands `qlearningextension:learning-rate` (line 7) and `qlearningextension:discount-factor` (line 8), respectively.

The extension provides two action selection policies: *random-normal* and ϵ-*greedy*. Both select an action at random with a given probability. However, in the ϵ-*greedy* policy such a probability is periodically reduced by a factor, as described earlier in Section 2.2. To specify the action selection policy, the extension provides the command `qlearningextension:action-selection`, that takes as argument the name of the policy and a list with its parameters (line 9).

The `go` procedure (lines 13–18) is where the behavior of the *Walker* agents is implemented. The extension provides two commands to activate the Q-Learning algorithm. The `qlearningextension:act` command (line 15) makes the agent choose and run an action according to the selection policy. The `qlearningextension:learn` command (line 16) makes the agent observe the new state and reward received, and update the *Q-table*, as described previously in Section 2.2.

As it can be seen, by using the commands provided by extension a developer does not need to implement the Q-Learning algorithm from scratch. However, it is worth to mention that there is no evaluation of whether this extension simplifies the development of agent-based simulations. We conducted such evaluation in the present paper.

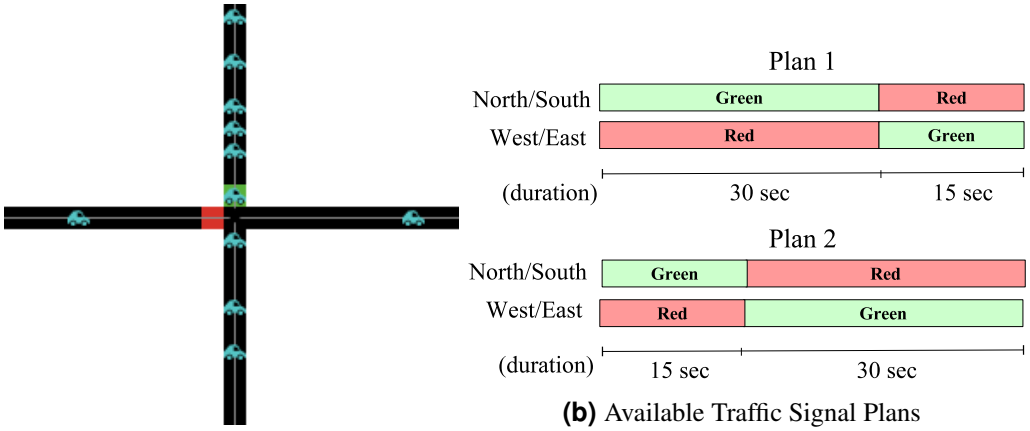
3. Quantitative Evaluation

The goal of the evaluation is to investigate whether the existing Q-Learning NetLogo extension simplifies the development of simulations. The evaluation considered two simulations: the *cliff walking* simulation described in Section 2.3, and an adaptive traffic signal control (ATSC) simulation. The latter was chosen because ATSC is a real-world problem in which successful applications of intelligent agents have been reported [2, 3]. This section describes the ATSC simulation, the evaluation procedure, and obtained results.

3.1. The ATSC Simulation with Q-Learning

In the area of traffic signal control, the goal is to develop traffic control systems that (i) maximize the overall capacity of the traffic network; (ii) maximize capacity of critical routes and intersection that represent bottlenecks; (iii) minimize negative impacts of traffic on the environment and energy consumption; (iv) minimize travel times; and (v) increase traffic safety [2]. In such systems, traffic signal devices (e.g., traffic lights) are used to control the traffic flow. In scenarios with complex traffic demands, traffic control systems should be able to *adapt* their policies to the current traffic conditions. Agent-based systems is an alternative that has been considered for creating these ATSC systems. By being endowed with learning capabilities, agents can learn traffic control policies in real time and thus optimize the overall traffic flow considering the existing infrastructure [10].

The environment of an ATSC simulation is a traffic network, which is composed of links and nodes that represent road lanes and intersections, respectively. In this paper, we adopted the model proposed by Oliveira and Bazzan [12], which specifies a traffic signal



(a) Traffic Network

(b) Available Traffic Signal Plans

Figure 4. Elements of the ATSC Simulation

controller (TSC) agent in charge of managing traffic light indicators. TSC agents are created at intersection and perceive the queue length on incoming lanes. The design of a TSC agent involves a set of concepts from the traffic control domain, which are described next. A *stage* describes a particular set of allowed traffic movements for vehicles in the lanes of the intersection. A *phase* is a period in which the indicators of the corresponding stage are green, allowing the traffic flow. Finally, a *plan* is a set of phases assigned to stages plus the sequence in which they are activated.

Figure 4a shows the traffic network considered in the evaluation. The network is composed of two one-way lanes. Vehicles on the vertical lane move from north to south, while on the horizontal lane they move from west to east. Each lane is 32 units long, which means that there can be at most 16 vehicles stopped on each direction at the intersection, waiting for the green light to cross it. The simplicity of this network allows focusing on the use of the Q-Learning algorithm, which is the goal of our evaluation. Figure 4b shows the two plans adopted in our simulation. Each plan has a total duration of 45 seconds. Plan 1 allows the traffic to flow north/south in the first 30 seconds, and then west/east in the last 15 seconds. In turn, plan 2 allows the traffic to flow north/south in the first 15 seconds, and then west/east in the last 30 seconds. Therefore, plan 1 gives priority to the north/south flow, and plan 2 to the west/east flow. In our simulation, a new vehicle is inserted on the vertical lane at every 4 seconds, while in the horizontal lane a new vehicle is inserted at every 15 seconds. Consequently, the traffic demand on the vertical lane is about 3.75 times higher than on the horizontal lane.

The Q-Learning specification for this ATSC simulation follows from Oliveira and Bazzan [12] and Santos et al. [14]. The goal of using Q-Learning in this simulation is to allow the TSC agent to learn which plan should be executed at every 45 seconds so as to minimizes the traffic jam (stopped vehicles) at intersections. The *state* definition is based on the queue length of the incoming lanes, and it is represented by a pair of values that corresponds to the number of stopped vehicles on each lane. Therefore, the state s at a particular instant is given by *(stopped north, stopped west)*. Each traffic signal plan is considered as an *action*, and therefore $A = \{plan1, plan2\}$. The *reward* is given by *0 - (stopped north + stopped west)*, which means that the higher the number of stopped vehicles at both lanes, the lower the reward received by the agent.

3.2. Procedure and Metrics

We adopted the following procedure to conduct the evaluation: (i) we implemented both simulations (cliff walking and ATSC) using the Q-Learning extension; and (ii) their source code was compared to other source codes implemented without the extension. The metric we selected for the comparison is the size of the simulation source code. More specifically, we use the number of lines of code (LoC), which is often used as a software size metric in cost estimation methods, such as Function Points [1] and COCOMO [4].

To assert that the implementations using the Q-Learning extension were consistent, we ran both simulations and inspected the policies learned by the agents. Regarding the Q-Learning parameters, we adopted $\alpha = 0.1$; $\gamma = 0.3$ for both simulations. For the cliff walking simulation, we adopted $random\text{-}normal = 0.7$. And for the ATSC simulation, we adopted $\epsilon = 0.7$ and $\epsilon\text{-}decay = 0.995$. The ATSC simulation results are based on 5 runs of 86400 ticks each.[3]. The cliff walking results are based on 5 runs of 350 ticks.

Table 1 shows the agent *Q-table* with the *Q-value* average for each pair *state-action* after running the cliff walking simulation. The maximum *Q-value* for each *state*, which correspond to the control policy learned by the agent, is emphasized with bold. As we can see, the *max* values are in the *state-action* pairs that form the optimal policy. For example, in the initial state *(0,0)*, the policy indicates to execute the *Up* action, so as to move towards the target without falling off the cliff. Then, the policy indicates to execute the *Right* action, to keep going towards the goal. Finally, the policy indicates to execute the *Down* action when the agent is located above the goal state.

Table 1. Q-Table of the Cliff Walking Simulation

States	**Actions**			
(x,y)	*Right*	*Down*	*Left*	*Up*
(0, 0)	-99,9999	-1,4281	-1,4282	**-1,4277**
(0, 1)	**-1,4259**	-1,4280	-1,4272	-1,4267
(0, 2)	**-1,4248**	-1,4255	-1,4254	-1,4252
(1, 1)	**-1,4207**	-99,8333	-1,4253	-1,4235
(1, 2)	**-1,4209**	-1,4217	-1,4237	-1,4214
(2, 1)	**-1,4075**	-99,5073	-1,4138	-1,4109
(2, 2)	**-1,4123**	-1,4134	-1,4174	-1,4146
(3, 1)	**-1,3727**	-96,9192	-1,3873	-1,3869
(3, 2)	**-1,3931**	-1,3941	-1,4004	-1,3991
(4, 1)	**-1,2776**	-91,2390	-1,3335	-1,3335
(4, 2)	**-1,3505**	-1,3535	-1,3697	-1,3548
(5, 1)	-1,1439	**-0,9911**	-1,1997	-1,2276
(5, 2)	-1,3006	**-1,2678**	-1,3006	-1,2976

Table 2 shows the *Q-table* for the ATSC agent. Only the states explored by the agent are shown. In this simulation, the agent learned to always choose the *Plan 1* action, given that the traffic demand in the vertical lane is higher than the horizontal lane, as previously detailed in Section 3.1.

[3]We assume 1 tick = 1 second, thus we simulate a 24h learning period for the TSC agent.

Table 2. Q-Table of the ATSC Simulation

States *stopped vehicles (north, west)*	Actions *Plan 1*	*Plan 2*	States *stopped vehicles (north, west)*	Actions *Plan 1*	*Plan 2*
(4, 2)	**-9,8587**	-13,9570	(10, 2)	**-9,6641**	-16,1368
(5, 2)	**-9,5479**	-13,5261	(11, 1)	**-2,5743**	-4,7753
(6, 2)	**-8,6826**	-10,6698	(11, 2)	**-9,8594**	-18,6116
(7, 2)	**-1,1338**	-1,7662	(12, 1)	**-2,1176**	-5,0304
(8, 2)	**-4,2193**	-7,3978	(13, 1)	**-7,9200**	-11,5943
(9, 1)	**-8,8922**	-18,0331	(14, 1)	**-13,7738**	-21,0598
(9, 2)	**-9,4045**	-15,6132	(15, 1)	**-14,2968**	-19,0880
(10, 1)	**-9,8652**	-19,4982	(16, 1)	**-15,6289**	-21,1684

Our evaluation is focused on the use of the Q-Learning algorithm. Therefore, only the LoC related to the agent's implementation were considered to compare the size of simulations source code. Blank lines, code comments, and code block delimiters were ignored. To avoid biases related to the source code indentation, we followed the conventions adopted in the simulations available in the NetLogo model library, in which each line contains a single code statement. The simulations source code is available online.[4]

3.3. Results and Discussions

Obtained results are shown in Table 3. To clarify the effects of using the Q-Learning NetLogo extension on the source code, we grouped the number of LoC by the following aspects regarding the simulation: *types and variables* (*T&V*), *setup*, and *execution*. The *T&V* aspect groups LoC for importing extensions and libraries, as well the definition of breeds and their variables. The *setup* aspect groups LoC related to the initialization of the simulation. In the ATSC simulation, such initialization consists of creating the traffic network, vehicles, and the TSC agent. In the cliff walking simulation, it consists of creating the cliff landscape and the Walker agent. Finally, the *execution* aspect groups LoC related to the behavior of agents.

Table 3. LoC of the Evaluated Simulations

Q-Learning Implementation	T&V	Setup	Execution	Total
Cliff Walking				
Original, without the extension	15	30	51	**96**
Using the extension	9	30	44	**83**
Adaptive Traffic Signal Control (ATSC)				
Original, without the extension [14]	27	23	119	**170**
Using the extension	16	18	63	**97**

[4] https://github.com/agentbasedsimulations/2021-wesaac-qlearning-netlogo-ext-evaluation-extras

With respect to *T&V*, by using the Q-Learning extension there is a reduction of 6 LoC in the cliff walking simulation (40%). In turn, the ATSC simulation using the extension required 11 (40.74%) few LoC. Without using the extension, both ATSC and cliff walking require extra LoC to specify additional agent variables and data structures to store all Q-Learning elements. These additional variables and data structures are encapsulated by the extension, so the developer does not need to declare them.

For the *setup* aspect, the size of the cliff walking source code is the same in both simulations. Regarding the ATSC simulation, the number of LoC was reduced by 21.74% (5 LoC). Such a reduction is because without the extension, additional LoC are required to initialize the Q-Learning algorithm, more specifically the *Q-table* data structure.

Concerning the *execution* aspect, the use of the extension produced source codes with fewer LoC in both simulations: 47.06% fewer in the ATSC simulation (73 LoC), and 13,72% fewer in the cliff walking (7 LoC). These reductions are due to the fact that by using the extension, the developer does not need to implement the Q-Learning algorithm previously described in Section 2.2. Instead, the developer only needs to invoke the *learn* and *act* commands, or the *learning* command, provided by the extension.

Overall, the agents source code developed using the extension was 13,54% (cliff walking) and 42.94% (ATSC) smaller than those developed without using it, which shows that the existing Q-Learning extension simplifies the development of simulations. Although our evaluation considered only these two simulations, which may raise questions regarding the generalization of the results for other domains, it is important to notice that the reduction in the number of LoC is due to the use of the commands provided by the extension. These Q-learning related commands are domain independent, which suggests that other simulations would also benefit from using the extension.

4. Conclusion

The agent-based modeling and simulation paradigm has been used to model and run simulations focused on the behavior of individuals and on the complexity that emerge from them. Agent-based simulations are usually developed in agent-based simulation platforms, as they provide features inherent to agents. NetLogo is a popular agent-based simulation platform, to which a reinforcement learning extension was recently made available. The extension provides commands for using the Q-Learning reinforcement learning algorithm. However, no evaluation on whether such extension simplifies the development of agent-based simulations was available yet.

In this paper, we conducted a quantitative evaluation to investigate whether the existing NetLogo Q-Learning extension simplifies the development of simulations. The evaluation considered two simulations: the classic cliff walking, and an adaptive traffic control simulation. Both simulation were implemented using the extension, and their source code were compared to existing implementations considering *lines of code* software size metric. The results showed that by using the extension the number of lines of code was reduced by 13.72% in the cliff walking simulation, and by 47.06% in the adaptive traffic signal control simulation. This gives evidence that the existing Q-Learning extension simplifies the development of simulations with NetLogo.

As future work, additional studies can be conducted with other simulations, to verify how our findings generalize to other agent-based simulation domains. Furthermore,

runtime might be a bottleneck in reinforcement learning applications, specifically when several agents learn simultaneously. Future studies should be conducted to investigate the scalability of the Q-Learning extension, given that execution time and memory consumption are out of the scope of this paper. Another possible work is to conduct an user study with humans, to evaluate if using the Q-Learning extension would also reduces the cognitive and development effort.

References

[1] Allan J Albrecht. Measuring application development productivity. In *Proceedings of the joint SHARE/GUIDE/IBM application development symposium*, volume 10, pages 83–92, 1979.

[2] Ana L. C. Bazzan. Opportunities for multiagent systems and multiagent reinforcement learning in traffic control. *Autonomous Agents and Multiagent Systems*, 18(3): 342–375, June 2009.

[3] Ana L. C. Bazzan and Franziska Klügl. A review on agent-based technology for traffic and transportation. *The Knowledge Engineering Review*, FirstView:1–29, 4 2013.

[4] Barry Boehm, Bradford Clark, Ellis Horowitz, Chris Westland, Ray Madachy, and Richard Selby. Cost models for future software life cycle processes: COCOMO 2.0. *Annals of Software Engineering*, 1(1):57–94, 1995.

[5] CoMSES. CoMSES Catalog, 2020. URL `https://catalog.comses.net/visualization/`. `https://catalog.comses.net/visualization/`, Acesso em: Jul/2020.

[6] Franziska Klügl and Ana L. C. Bazzan. Agent-based modeling and simulation. *AI Magazine*, 33(3):29–40, 2012.

[7] Kevin Kons. *Biblioteca Q-Learning para desenvolvimento de simulações com agentes na plataforma NetLogo*. Trabalho de conclusão de curso, Universidade do Estado de Santa Catarina (UDESC), 2019.

[8] Kevin Kons and Fernando Santos. Cliff walking with q-learning netlogo extension. CoMSES Computational Model Library, 2019. Retrieved from: `https://www.comses.net/codebases/b938a820-f209-4648-afc6-0946657c3484/releases/1.0.0/`.

[9] Charles Macal and Michael North. Introductory tutorial: Agent-based modeling and simulation. In *Proceedings of the 2014 Winter Simulation Conference*, WSC '14, pages 6–20, Piscataway, NJ, USA, 2014. IEEE Press.

[10] Patrick Mannion, Jim Duggan, and Enda Howley. An experimental review of reinforcement learning algorithms for adaptive traffic signal control. In Leo Thomas McCluskey, Apostolos Kotsialos, P. Jörg Müller, Franziska Klügl, Omer Rana, and René Schumann, editors, *Autonomic Road Transport Support Systems*, pages 47–66. Springer, 2016.

[11] S. T. Monteiro and C. H. C. Ribeiro. Desempenho de algoritmos de aprendizagem por reforço sob condições de ambiguidade sensorial em robótica móvel. *Revista Controle & Automação*, 15(3):320–338, 2004.

[12] D. de. Oliveira and A. L. C. Bazzan. Multiagent learning on traffic lights control: effects of using shared information. *IGI Global*, pages 307–321, 2009.

[13] Stuart Russel and Peter Norvig. *Inteligência Artificial*. Rio de Janeiro: Campus, 2 edition, 2004.

[14] Fernando Santos, Ingrid Nunes, and Ana L. C. Bazzan. Model-driven agent-based simulation development: a modeling language and empirical evaluation in the adaptive traffic signal control domain. *Simulation Modelling Practice and Theory*, 83: 162–187, April 2018.

[15] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2 edition, 2018.

[16] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine learning*, 33 (3–4):279–292, 1992.

[17] Uri Wilensky. NetLogo, 1999. URL `http://ccl.northwestern.edu/netlogo/`. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.

[18] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

A influência da dor no ritmo circadiano com base em modelo matemático, estatístico e sistema multiagente

Angélica T. Santos[1], **Catia M. Machado**[1], **Diana F. Adamatti**[1]

[1]Programa de Pós Graduação em Modelagem Computacional (PPGMC)
Universidade Federal do Rio Grande (FURG)
Caixa Postal 474 – 96.203.900 – Rio Grande – RS – Brasil

{theisangelica, catiamachado, dianaadamatti}@furg.br

Abstract. *The circadian rhythm controls the unconscious activities of living beings through the clock biological. External influences, such as pain, depression, anxiety, obesity, elevated body temperature and hormones can cause dysfunction in the synchronization and desynchronization of the circadian rhythm. In this context, it is proposed to present the parameters developed for mathematical, computational and statistical modeling that describe the circadian rhythm based on pain. For such modeling, a questionnaire, data collection and parameterization of the variables were carried out. Results show how the pain variable directly influences the quality of sleep, as well as the development of daily activities.*

Resumo. *O ritmo circadiano comanda as atividades inconscientes dos seres vivos através do relógio biológico. Influências externas, como a dor, depressão, ansiedade, obesidade, temperatura corporal elevada e hormônios podem causar disfunção na sincronização e dessincronização do ritmo circadiano. Nesse contexto, é proposto apresentar os parâmetros desenvolvidos para modelagem matemática, computacional e estatística que descrevem o ritmo circadiano com base na dor. Para tal modelagem, foi realizado um questionário, coleta de dados e parametrização das variáveis. Resultados mostram como a variável dor influencia diretamente na qualidade do sono, bem como no desenvolvimento das atividades do cotidiano.*

1. Introdução

A cronobiologia (crono = tempo, bio = vida, logia = estudo) é a área da Biologia que estuda o relógio biológico dos seres vivos. O relógio biológico de cada pessoa é sincronizado conforme suas atividades decorrentes do dia. Assim, a marcação horária interna eventualmente é precisa. Para que haja a regulação interna, é necessário que os mecanismos que permitem a sincronização sejam ajustados. Essa sincronização é realizada pelo fenômeno de ajuste, chamado de "arrastamento", que comanda o ajuste onde denominado "zeitgeber", atuando como sincronizador do relógio biológico. Assim, o ritmo circadiano e homeostático são sincronizados pelo "zeitgebers", de maneira que os mesmos estejam sempre interligados. Essa interligação é realizada por meio de "pacemaker" (marcapasso) [Daan et al. 1984].

O estudo do ritmo circadiano faz parte da cronobiologia, área pertencente às ciências biológicas, que tem como objetivo estudar os relógios biológicos que controlam os ritmos e são responsáveis por atividades dos seres vivos. Sobretudo, os ritmos que

estão associados a funções vitais como hormônios, sistema digestivo, sensação de sono e fome, e influências externas, como dor, ansiedade ou depressão [Bruna 2019].

Pesquisas cientificas relacionadas ao sono destacam-se porque ocupam parte da vida humana, conforme apresentado no referencial teórico de [dos Santos et al. 2019]. O sono é uma necessidade para todos os indivíduos e a qualidade do sono afeta diretamente o desenvolvimento das atividades do cotidiano. A necessidade de dormir varia em cada indivíduo, sendo que ter uma qualidade ruim de sono, desencadeia uma série de alterações comportamentais [Bergamasco et al. 2006].

O sono é modelado conforme os horários do indivíduo, neste estudo, é modelado através de um sistema multiagente, que é a área que estuda o comportamento de um conjunto independente de agentes com características diferentes, evoluindo em um ambiente [Wooldridge 2009]. A união dos modelos matemáticos com o sistema multiagente são úteis para representar situações reais, realizar previsões e auxiliar no suporte à decisão.

Modelando o ritmo circadiano através de um sistema multiagente, é possível analisar as funções vitais que estão ligadas diretamente ou indiretamente ao sistema biológico, podendo ser hormônios ou sistema digestivo, e influências externas, como a dor, ansiedade, depressão e temperatura corporal. À vista disso, percebe-se a necessidade de investigar sobre as influências externas, especificamente a dor devido a grande relevância que a dor tem na vida dos indivíduos. A dor afeta a qualidade de vida, o dia a dia, os afazeres.

Neste contexto, estudar e diagnosticar a dinâmica e funcionamento do ritmo circadiano é fundamental para a cronobiologia e para a ciência. Assim, o trabalho propõe apresentar os resultados da pesquisa que está em andamento sobre o modelo matemático de dois processos - ritmo circadiano e ritmo homeostático, que descreve as curvas do ritmo circadiano apresentado por [Daan et al. 1984], um modelo computacional baseado em sistema multiagente, partindo da implementação realizada por [Skeldon 2014], e a inclusão da variável dor, que afeta diretamente o ritmo circadiano.

Assim, o trabalho está organizado nas seguintes seções: na seção 2 o referencial teórico, na seção 3 a metodologia, na seção 4 a discussão dos resultados, seguido das conclusões e próximos passos.

2. Referencial Teórico

Nesta seção é apresentado o referencial teórico relacionado a Sistema Biológico, Ciclo Vigília/Sono, Ritmo Circadiano, Ritmo Homeostático, Dor, Modelo Matemático, Sistemas Multiagente e a ferramenta Netlogo.

2.1. Sistema Biológico

O sistema biológico do ritmo circadiano são mudanças cíclicas que se repetem ao longo de um determinado período e estão relacionados com as alterações dos processos fisiológicos do corpo, sendo que a atividade de dormir é um mecanismo de reparo das células que regulam os processos físicos, intelectuais e psíquicos [Borbély and Achermann 1999].

A regulação do sistema biológico ocorre por meio do ciclo vigília-sono, que é subdividido em ritmo circadiano e ritmo homeostático, caracterizado pela redução significativa da atividade motora e da percepção de estímulos sensoriais [Borbély and Achermann 1999].

A variação entre o sono e vigília é fundamental para a saúde mental e física de todos os seres. Além de tornar fundamental o equilíbrio social e profissional, as alterações do sono, provocam insônia, sonolência e anemia. O ciclo vigília-sono é controlado pelo sistema hipotalâmico e seus sistemas funcionais do sistema circadiano. O ritmo circadiano do ciclo vigília-sono é regulado pelos núcleos supraquiasmáticos (NSQ's) do hipotálamo. Os núcleos estão localizados na base do cérebro, sobre o cruzamento das fibras nervosas originárias dos olhos [Saper et al. 2001].

O ciclo vigília-sono tem ritmicidade de 24 horas, período de vigília ocorre durante o dia e o sono durante a noite, dando ênfase que o sono realizado durante o dia não tem a mesma qualidade do sono noturno, bem como a vigília que ocorre a noite não é igual a do dia.

Um dos mecanismos importantes da regulação do vigília-sono, é mostrado pelo processo $\tilde{S}$ (união do ritmo circadino e ritmo homeostático) [Daan et al. 1984], este por sua vez, é dependente da duração e qualidade do sono [Borbély and Achermann 1999]. A duração da vigília incrementa o processo $\tilde{S}$, aumentando assim o tempo de sono. O vigília-sono é regulado pelo modelo de dois processos (ritmo circadiano e homeostático).

O ritmo circadiano (circa = por volta de, dies = dia) e todos os demais ritmos têm características auto-sustentáveis, e são ajustados para o ciclo vigília-sono com período de 24 horas, influenciado por fatores internos e externos. Ele regula atividades químicas, físicas, psicológicas e psíquicas do organismo.

Já o ritmo homeostático é decorrente do vigília-sono que procede do modelo do Processo $\tilde{S}$. O ritmo homeostático é decorrente da pressão homeostática, na qual é máxima no início do mesmo e dissipa-se gradualmente ao longo da fase. Na vigília, a pressão homeostática é miníma, e aumenta ao longo do período [Beersma and Gordijn 2007].

2.2. Modelo Matemático

O função do sono é recuperar a fadiga decorrente da vigília, sendo o sono regulado por dois processos, já descritos anteriormente, o ritmo circadiano e ritmo homeostático. O ciclo vigília-sono é definido como o resultado da interação entre o ritmo circadiano e ritmo homeostático [Daan et al. 1984].

O modelo analisa um "pacemaker", localizado no núcleo supraquiasmático do hipotálamo, exercido pelos mecanismos do "zeitgeber". No trato retino-hipotalâmico são geradas as oscilações circadianas fisiológicas por meio do ciclo vigília sono.

A explicação das oscilações se dá por meio de dois limiares: H e L, denominado "S-Thresholds", sendo que o processo $\tilde{S}$ (tempo de interação entre o ritmo circadiano e ritmo homeostático) aumenta durante a vigília até atingir o limiar H (altura máxima para o início do sono), e, consequentemente diminui durante o sono até atingir o limiar L (momento em que o sono termina). Assume-se, por combinação que "S-Thresholds" decresce durante o sono e cresce durante a vigília.

O modelo matemático de dois processos [Daan et al. 1984] considera a pressão homeostática $H(t)$ que diminui exponencialmente durante o sono:

$$H(t) = H_0 e^{(t_0 - t)/X_s} \tag{1}$$

E consequentemente aumenta durante a vigília, período que o indivíduo está acordado:

$$H(t) = \mu + (H_0 - \mu)e^{(t_0 - t)/X_w} \tag{2}$$

O parâmetro μ é a "assíntota superior" (upper asymptote), pressão máxima que o ritmo homeostático H pode alcançar, sem interferências externas ou internas e a *assíntota inferior* (lower asymptote) é abaixo de zero. Alternando entre o tempo de dormir e acordar ocorre a pressão homeostática $H(t)$, onde atinge o limite superior $H^+(t)$, que consiste no valor médio H_0^+ da modulação do ritmo circadiano $C(t)$:

$$H^+(t) = H_0^+ + aC(t) \tag{3}$$

A oscilação entre o sono e vigília resulta quando $H(t)$ atinge o limite inferior (lower threshold), $H^- t$:

$$H^-(t) = H_0^- + aC(t) \tag{4}$$

Onde o $C(t)$ é o período de 24 horas, sendo descrito de maneira simplificada por:

$$C(t) = sin(\omega(t - \alpha)) \tag{5}$$

E na maneira mais complexa, que inclui alterações internas ou externas, por:

$$C(t) = \begin{cases} 0.97sin[\omega(t-\alpha)] + 0.22sin[2\omega(t-\alpha)] + 0.07sin[3\omega(t-\alpha)] + \\ 0.03sin[4\omega(t-\alpha)] + 0.001sin[5\omega(t-\alpha)] \end{cases} \tag{6}$$

Os parâmetros apresentados são decorrentes de pesquisas realizadas por [Daan et al. 1984, Borbély and Achermann 1999], e foram definidos com base em dados de adultos saudáveis, submetidos a programas de vigília sono natural ou alterados. Esses indivíduos foram expostos a esforços e reforços nos seus ambientes diários que auxiliam o processo sono vigília a permanecer em sincronia com os demais ritmos.

2.3. Sistemas Multiagente

A simulação é uma técnica que envolve a construção de um modelo de situação real para posterior experimentação. Os sistemas multiagente com base nas informações preliminares do ambiente, podem tomar decisões para atingir o objetivo do grupo e/ou pessoal de maneira independente [Lesser 1999]. Nesse meio é possível mostrar uma população real em uma forma artificial, onde cada indivíduo da população é apresentado por um agente e todos os agentes formam um grupo, sendo que cada grupo possui suas regras e comportamento.

O sistema de vários agentes, que trabalham em conjunto, e estes agentes podem ter características diferentes, atingindo metas, evoluindo o ambiente [Wooldridge 2009].

A partir de um problema real ou imaginário é realizado o estudo, a modelagem do problema, a implementação e a validação por meio de sistemas multiagente. Neste estudo, foi escolhido o Netlogo, para realizar a implementação do ritmo circadiano e a variável dor, por ser um software livre e pela implementação do modelo de dois processos de [Skeldon 2014] já ter sido realizado em Netlogo.

O NetLogo[1] é um ambiente de programação de modelagem para sistemas multiagente, com intuito de analisar o comportamento dos fenômenos naturais e/ou sociais [Tisue and Wilensky 2004].

2.4. Dor

De acordo com a Associação Internacional de Estudos da Dor (*International Association for the Study of Pain* - IASP), a dor é "uma experiência sensorial, emocional e desagradável, associada a fatores reais internos ou externos" [Moayedi and Davis 2013].

Dado que a dor é uma experiência sensorial desagradável, ela está relacionada com a medula espinhal que envolve circuitos neurais no cérebro. Os circuitos envolvem uma gama de sistemas neurotransmissores e cognitivos, e as respostas cognitivas influenciam sobre a percepção da dor [da Silva and Pinto 2011]. Assim, a modelagem do processamento da dor é relacionada com o todo do corpo humano.

3. Metodologia

A metodologia desta pesquisa está embasada em um questionário não invasivo[2]. Para cada dia da semana, o indivíduo responde sobre o horário de dormir, horário de acordar e nível de dor, ainda, questionado sobre local da dor e produtividade no trabalho. O horário de dormir e acordar, são definidos como hora fixa, devido ao tempo constante que o mesmo leva para dormir profundamente e despertar ao acordar. Caso o horário de dormir não conste no questionário, é porque o indivíduo é considerado fora dos padrões da pesquisa. A ordem da metodologia é composta da seguinte forma:

- coleta de dados;
- separação dos dados;
- simulação em Netlogo;
- análise estatística em RStudio;
- análise de clusterização k-means.

Após a coleta de dados, os mesmos foram preparados para posterior análise. De uma forma geral quanto maior o tamanho da amostra, menor a chance de os resultados serem apenas coincidência, e espera-se um desvio padrão menor, visto que se encontrará um comportamento mais estável. Estatisticamente, é necessária uma amostra mínima de 30 indivíduos [Morettin and Bussab 2017].

Os dados foram separados por indivíduos que têm relevância para a pesquisa (critérios de inclusão). Cada resposta foi simulada em Netlogo, conforme o modelo da simulação apresentado em [Dos Santos et al. 2020] que permite uma análise de cada indivíduo. Para os testes estatísticos foi utilizada a técnica de Principal Component Analysis (PCA) no software RStudio.

[1] https://ccl.northwestern.edu/netlogo/

[2] Plataforma Brasil e Comitê de Ética em Pesquisa na Área da Saúde (CEPAS) da Universidade Federal do Rio Grande - FURG, sendo este aprovado pelo Protocolo CAAE: 18147119.9.0000.5324

A Análise de Componentes Principais é um algoritmo que estuda a variação de um conjunto de dados, com o intuito de reduzir a dimensão dos dados. Nesta pesquisa, o PCA é utilizado para agrupamento de informações, delineamento e união de variáveis dependentes ou independentes, que permitem ser representadas em duas ou mais dimensões [Ringnér 2008].

O primeiro componente principal contém a maior variação dos dados, o segundo componente contém a segunda maior variação, e assim por diante. Vale ressaltar que o PCA foi aplicado para identificar direções com maiores variações de informações.

Ainda para análise dos dados foi aplicado a técnica de K-means, na qual é realizado clusterização dos dados e encontram-se as variáveis correlacionadas, variáveis dependentes e variáveis independentes.

4. Discussão dos Resultados

Para discutir os resultados, introduz-se o questionário, seguido de testes no Netlogo, e posteriormente análise na ferramenta RStudio.

Um total de duzentos e doze participantes responderam ao questionário, sendo: 1,4% não aceitaram participar da pesquisa; 0,5% tem menos de 20 anos; 71,9% tem entre 20 e 35 anos, e 26,2% tem mais de 35 anos. Definido no projeto do CEPAS o público-alvo desta pesquisa indivíduos entre 20 e 35 anos, visto que é nessa faixa etária que o sono sofre mais picos de dessincronização, ao final tem-se 151 indivíduos respondentes válidos da pesquisa.

É perceptível que os participantes têm mais dor em uma região do corpo, podendo ser membros superiores e membros inferiores, ou cabeça e coluna, ou ainda cabeça, coluna e membros inferiores. Levando em consideração a região da dor, deduz-se que a dor é resultante do tipo de trabalho desenvolvido.

Com relação aos 151 indivíduos que se enquadram nessa pesquisa, foram realizadas simulações em Netlogo, para verificar como o ritmo circadiano e a variável dor comportam-se. Aqui, são apresentados três casos com interferência da dor (indivíduos C, D e E), conforme a Figura 1, sabendo que:

- Amarelo - indivíduo deveria estar acordado, mas está tentando dormir;
- Rosa - indivíduo queria dormir, mas está acordado;
- Preto - ritmo homeostático;
- Cinza - ritmo circadiano.

O indivíduo (C) tem 34 anos, dorme todos os dias as 25 horas (1 hora da manhã seguinte), e acorda de sábado a quarta as 10 horas e na quinta e na sexta as 8 horas, dormindo em média 9 horas por noite. Seus níveis de dor variam de 0; 0; 0; 0; 4; 4 e 2. É visível que no domingo, segunda-feira, terça-feira e quarta-feira o mesmo não apresentou dor, mas apresentou momentos em que queria estar dormindo, mas estava acordado. Já na quinta e sexta teve um nível de dor quatro, seguido de dor dois no sábado. A dor quatro da sexta é mais forte que a dor quatro de quinta, pois este acumulou o cansaço da semana e também o processo de acordar cedo.

O indivíduo (D) tem 30 anos, dorme em média 7 horas por noite. O nível de dor varia, sendo: 0; 6; 4; 4; 6; 2 e 2. O nível 6 de dor na segunda-feira é diferente do nível

6 de dor na quinta-feira, pois na quinta-feira já está acumulado o cansaço dos demais dias, a falta de sono, devido à dor que está sofrendo todos os dias. Consequentemente, a dor vai acarretando na produtividade e deixando o indivíduo menos produtivo na sexta. Percebe-se que no sábado, mesmo tendo nível 2 de dor, o indivíduo consegue dormir e descansar.

Já o indivíduo (E) tem 27 anos, um ritmo circadiano sincronizado pois dorme todos os dias as 23 horas e acorda as 8 horas. Levando em consideração a variável dor o ritmo circadiano fica dessincronizado pois teve nível oito de dor no domingo e os demais dias da semana nível máximo de dor dez, consequentemente não consegue dormir.

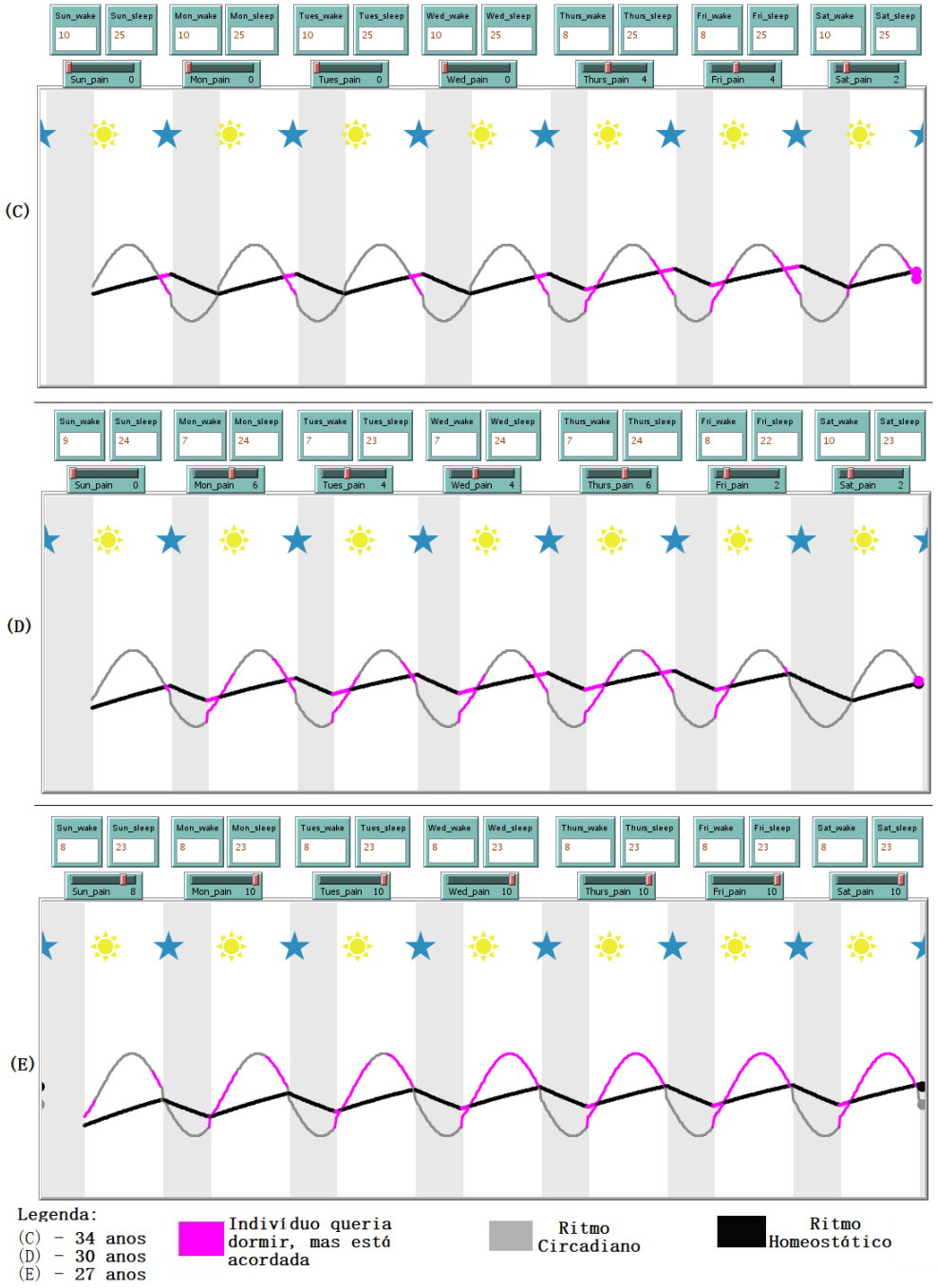


Figura 1. Comparação de Indivíduos.

Para uma análise mais aprofundada, ainda foi realizada a Análise de Componentes Principais no RStudio. O Software R oferece diferentes formas de analisar dados. Inicialmente é estudada a análise gráfica do PCA, seguida do agrupamento no K-means [Kodinariya and Makwana 2013].

O PCA é um algoritmo matemático que analisa a variação do conjunto de dados, auxiliando a reduzir a dimensionalidade dos dados e componentes [Ringnér 2008]. Foi utilizado o método de *Scree Test* e Autovalor, que indicam 3 componentes como principais e capazes de explicar as variáveis.

Desta forma, são escolhidos 2 componentes principais, de acordo com o método do Autovalor, visto que estes dois primeiros componentes principais têm uma proporção cumulativa de 92,25%, onde preserva a variância dos dados. Na Figura 2 é apresentada a relação entre os componentes principais de forma espacial.

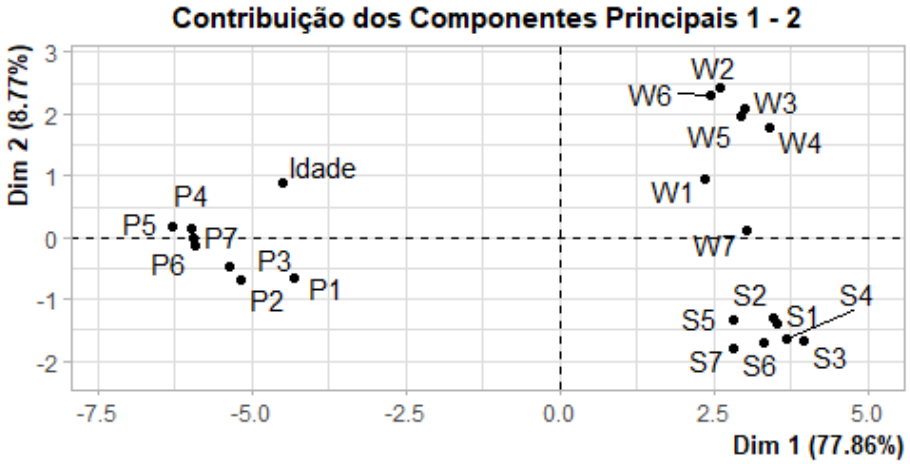


Figura 2. Componente Principal - Dim 1 x Dim 2, onde:
W_n representa hora de acordar (wakeup);
S_n representa a hora de dormir (sleep);
P_n representa a dor (pain).

Fonte: As autoras (2020).

Na Figura 2 é possível analisar que o horário de acordar (W1; W2; W3; W4; W5; W6; W7) sempre será positivo. É visível que a dor está mais relacionada com a idade, assim como a hora de dormir está mais relacionada com a hora de acordar. Para um melhor entendimento, é utilizada técnica de agrupamento (clusterização), com o algoritmo K-means.

O primeiro passo é encontrar a quantidade ideal de cluster (N). Existe uma variedade de métodos para encontrar a quantidade N de cluster. Neste trabalho foram aplicados três métodos Gap Statistic, Elbow e Silhoette [Kodinariya and Makwana 2013].

Aplicando os métodos Gap Statistic e Elbow encontra-se N = 3, como número ideal de cluster. A Figura 3 mostra como as variáveis estão agrupadas. Sendo que a hora de dormir (cluster azul) aparece no primeiro quadrante e com valores positivos. A hora de acordar (cluster verde) está em um quadrante negativo, mas próximo ao eixo x. Já a idade e o nível de dor (cluster vermelho), que são variáveis dependentes e estão diretamente relacionadas.

Já para o método de Silhouette, número ótimo de cluster foi N = 2. Utilizando-se este N, o resultado do agrupamento é apresentado na Figura 4, a qual o cluster vermelho refere-se a hora de dormir e acordar, sendo estas duas variáveis diretamente dependentes. Já o cluster azul, nível de dor e idade, continuam formando um cluster, como mostrado na Figura 3.

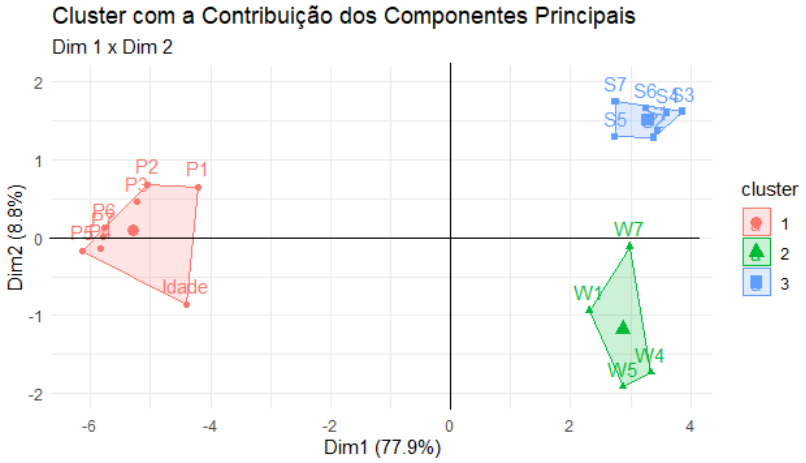


Figura 3. Cluster (N = 3) com o PCA na Dimensão 1 - 2
Fonte: As autoras (2020).

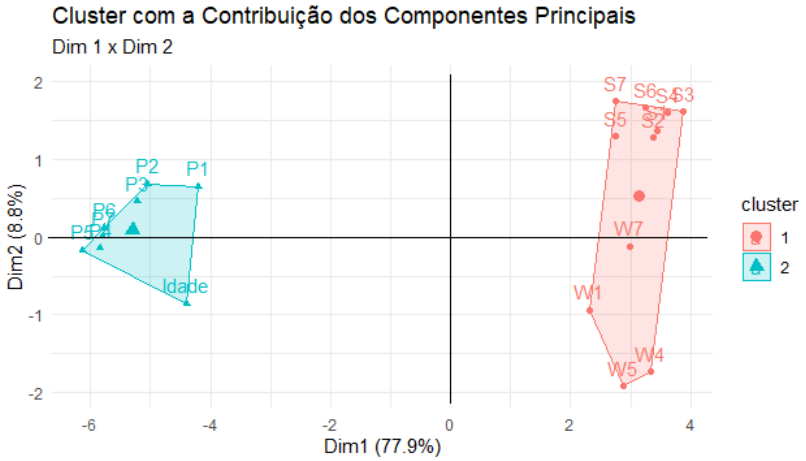


Figura 4. Cluster (N = 2) com o PCA na Dimensão 1-2
Fonte: As autoras (2020).

Analisando os clusters formados a partir das Figuras 3 e 4, é possível relacionar com a análise realizada no Netlogo (Figura 1), onde percebe-se que a sincronização independe da dor, pois quando o indivíduo dorme, em média, oito horas diárias, o ritmo circadiano fica sincronizado. Quando a dor interfere no sono, ou o indivíduo não dorme em média 8 horas, o ritmo fica dessincronizado.

5. Considerações Finais

Tendo em vista o objetivo proposto de apresentar os resultados da pesquisa que está em andamento sobre o modelo matemático de dois processos - ritmo circadiano e ritmo homeostático, que descreve as curvas do ritmo circadiano, modelo computacional baseado em sistema multiagente e a inclusão da variável dor, conclui-se que os modelos apresentados mostram de forma fidedigna a dor, diretamente relacionada com as horas de dormir, assim, como a dor influencia diretamente na qualidade do sono.

Gerando dois ou três clusters, a dor sempre vai estar diretamente relacionada com a idade, assim como, a hora de dormir sempre está relacionada com a hora de acordar.

Desta forma, a hora de dormir e acordar são variáveis dependentes, uma sempre irá depender da outra.

A uso do modelo multiagente e matemático traz vantagens para a pesquisa na qual é possível interligar modelos, unir e promover a interdisciplinariedade, neste têm-se o modelo multiagente,matemático e estatístico, além do banco de dados, conhecido como ciência de dados.

Como trabalhos futuros, pretende-se realizar mais testes, aprofundar a pesquisa, e provar o quanto a produtividade é afetada pela dor.

Agradecimentos

As autoras deste artigo agradecem à Fundação de Amparo a Pesquisa do Estado do Rio Grande do Sul - FAPERGS pelo recurso financeiro no desenvolvimento da pesquisa e a Universidade Federal do Rio Grande - FURG.

Referências

Beersma, D. G. and Gordijn, M. C. (2007). Circadian control of the sleep–wake cycle. *Physiology & behavior*, 90(2-3):190–195.

Bergamasco, E. C. et al. (2006). Alterações do sono: diagnósticos freqüentes em pacientes internados. *Revista Gaúcha de Enfermagem*, 27(3):356.

Borbély, A. A. and Achermann, P. (1999). Sleep homeostasis and models of sleep regulation. *Journal of biological rhythms*, 14(6):559–570.

Bruna, M. H. V. (2019). Relógios biológicos. `https://drauziovarella.uol.com.br/neurologia/relogios-biologicos-artigo/`. [Online; accessed 19-February -2020].

da Silva, J. A. and Pinto, N. (2011). A dor como um problema psicofísico. *Rev. Dor. São Paulo*, 12(2):138–151.

Daan, S., Beersma, D., and Borbély, A. A. (1984). Timing of human sleep: recovery process gated by a circadian pacemaker. *American Journal of Physiology-Regulatory, Integrative and Comparative Physiology*, 246(2):R161–R183.

dos Santos, A. T., Machado, C. M., and Adamatti, D. F. (2019). Ritmo circadiano e a variável dor: Revisões sistemáticas com a utilização de simulação multiagente. In *Anais do 13º Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicaç oes– IX WESAAC - Florianopolis*.

Dos Santos, A. T., Machado, C. M., and Adamatti, D. F. (2020). Circadian rhythm and pain: Mathematical model based on multiagent simulation. *Journal of Medical Systems*, 44(10):1–9.

Kodinariya, T. M. and Makwana, P. R. (2013). Review on determining number of cluster in k-means clustering. *International Journal*, 1(6):90–95.

Lesser, V. R. (1999). Cooperative multiagent systems: A personal view of the state of the art. *IEEE Transactions on knowledge and data engineering*, 11(1):133–142.

Moayedi, M. and Davis, K. D. (2013). Theories of pain: from specificity to gate control. *Journal of neurophysiology*, 109(1):5–12.

Morettin, P. A. and Bussab, W. O. (2017). *Estatística básica*. Saraiva Educação SA.

Ringnér, M. (2008). What is principal component analysis? *Nature biotechnology*, 26(3):303–304.

Saper, C. B., Chou, T. C., and Scammell, T. E. (2001). The sleep switch: hypothalamic control of sleep and wakefulness. *Trends in neurosciences*, 24(12):726–731.

Skeldon, A. (2014). Are you listening to your body clock? `http://personal.maths.surrey.ac.uk/st/A.Skeldon/sleep.html`. [Online; accessed 25-March -2020].

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

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

Modelo baseado em multiagentes para aplicação de estratégias de isolamento social para o combate da pandemia da Covid-19

Giulia T. Monteiro, Diana F. Adamatti

[1]Programa de Pós-Graduação em Computação (PPGComp)
Universidade Federal do Rio Grande (FURG)
Caixa Postal 474 – 96.203-900 – Rio Grande – RS – Brazil

{giuliatondinm, dianaada}@gmail.com

Abstract. *This paper presents a model based on multi-agents to visualize new strategies of social distancing to deal with the Covid-19 pandemic and demonstrates an alternative to the lockdown procedure widely adopted by several regions of the world. The achieved results showed the importance of complying with the isolation norms and the possible positive effects of a cyclical strategy of controlled reopening of different sectors of the economy on the rate of the disease transmission.*

Resumo. *Este artigo apresenta um modelo baseado em multiagentes para visualização de novas estratégias de distanciamento social para lidar com a pandemia da Covid-19 e demonstra uma alternativa para o procedimento de lockdown amplamente adotado por diversas regiões do mundo. Os resultados alcançados evidenciaram a importância do cumprimento das normas de isolamento e os possíveis efeitos positivos de uma estratégia cíclica de reabertura controlada de diferentes setores da economia sobre a taxa de transmissão da doença.*

1. Introdução

O mundo enfrenta atualmente uma grave crise sanitária: a pandemia de Covid-19, causada pelo novo coronavírus (SARS-CoV-2). Relatado pela primeira vez em Wuhan na China em dezembro de 2019, espalhou-se rapidamente por diversos países asiáticos, chegando a Europa e, posteriormente, aos demais continentes. O primeiro caso da doença no Brasil foi confirmado em fevereiro de 2020 e um ano após essa confirmação, o país já havia acumulado um número total de mortes de aproximadamente 250 mil pessoas [Saúde 2021].

Visto a gravidade dessa doença altamente transmissível, diversas medidas de prevenção foram tomadas em todos os países para diminuir os casos de Covid-19 e evitar o colapso dos sistemas de saúde. Uma das estratégias que foi amplamente adotada por várias regiões do mundo foi o chamado *lockdown* [Atalan 2020]. De modo simples, essa medida bloqueia a movimentação e as atividades econômicas quando um número limite de casos da doença é excedido e desbloqueia, quando os casos diminuem. Essa medida foi implementada com menor ou maior amplitude nessas diversas regiões e contribuiu para impedir o sobrecarreagamento dos serviços de saúde.

Porém, no estudo realizado por [Karin et al. 2020], observou-se que os procedimentos operacionais de sua implementação poderiam levar ao surgimento de novas ondas com o acúmulo de novos casos ao liberar a movimentação e as atividades, além de levar à incerteza econômica. Nessa pesquisa, a curva de disseminação da Covid-19 durante períodos de *lockdown* padrão foi comparada com a curva de disseminação resultante de uma nova estratégia cíclica adaptativa para essa medida. Nessa nova estratégia, é apontada uma adaptação cíclica do *lockdown* onde x dias a população realiza suas atividades no meio social e y dias, mantém-se em isolamento. A partir dessa comparação matemática, foi demonstrado que uma estratégia cíclica de *lockdown* seria mais eficiente do que a sua abordagem padrão no achatamento da curva de disseminação da doença, diminuindo a taxa de disseminação da doença ao longo do tempo.

Este trabalho tem como objetivo adaptar essa nova estratégia proposta de forma matemática e levar alguns de seus fatores para um modelo baseado em multiagentes. Busca-se, através da simulação de diferentes cenários de reabertura dos setores da economia, das diferentes combinações de medidas de prevenção e diferentes ciclos de isolamento social, demonstrar como esses fatores afetam a taxa de disseminação da doença.

2. Simulação

A simulação foi implementada na plataforma NetLogo[1], que possui uma linguagem de programação própria e um interface gráfica para o desenvolvimento de modelos baseados em agentes. O modelo desenvolvido permite realizar simulações com uma população de até 999 pessoas e até 100 pessoas já infectadas ao iniciar o cenário. Seu código fonte está disponível em um repositório na plataforma GitHub[2].

Para relacionar o nível de serevidade da doença sobre uma pessoa e como a mesma se comportaria durante a simulação, a população foi dividida em grupos etários. De modo geral, uma população pode ser dividida, a partir de dados demográficos, em três faixas: jovens, adultos e idosos [Sebastien et al. 2020]. Além disso, no ambiente proposto, a população pode ser segmentada em grupos familiares, compostos por uma média de 3 pessoas por casa. Essa média foi definida a partir do resultado do levantamento do número médio da composição familiar no Brasil realizado no estudo de [Ohana 2019].

Nesse modelo, diferentes cenários de reabertura dos setores da economia podem ser simulados. Os setores que constituem a simulação são: serviços, comércio, indústria, construção e escola. Cada pessoa da população faz parte de apenas um setor. Os jovens pertencem ao setor "escola" e a população adulta é distribuída no restante dos setores. Todas as pessoas idosas ou apenas uma parcela podem ser definidas como aposentadas. Os idosos não aposentados são divididos juntamente com a população adulta pelos setores.

A plataforma NetLogo utiliza-se de um contador, chamado de *tick*, para representar a passagem do tempo em uma determinada simulação. No modelo proposto, foi definido um relógio que realiza a contagem dos dias e horas desde o ínicio da simulação baseado nesses *ticks*. Cada dia é composto por 10 *ticks*. Ao iniciar o dia, cada pessoa da população afasta-se de seu grupo familiar e movimenta-se para o seu setor específico. Dentro da área do setor, a pessoa move-se aleatoriamente. Ao fim do dia, cada pessoa

[1]https://ccl.northwestern.edu/netlogo/

[2]https://github.com/giuliatondin/cyclic-lockdown-model

volta para o seu grupo familiar. Sempre ao iniciar um novo dia, é verificado quais setores estão abertos e apenas a parcela da população referente aos mesmos voltam a movimentar-se.

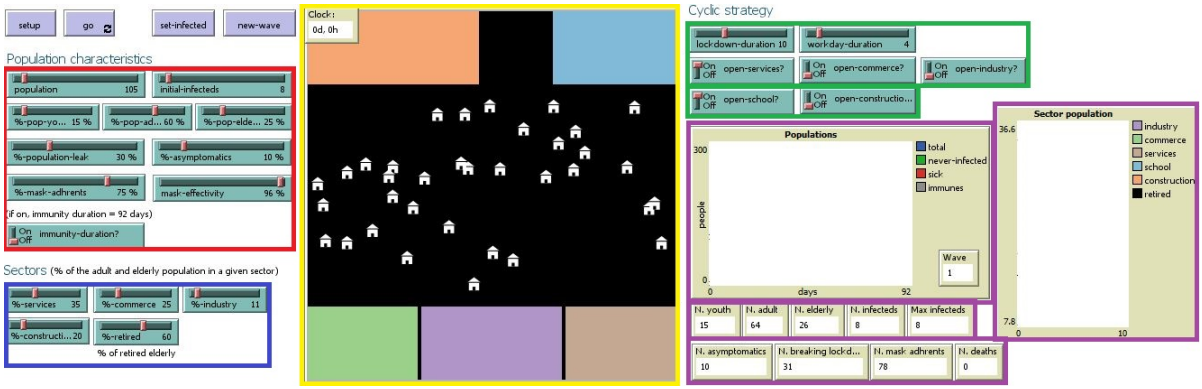
2.1. Ambiente

O ambiente deste modelo é fundamentado a partir de dois agentes: *Turtles*, agentes que se movem no mundo, e *Patches*, agentes que formam o ambiente bidimensional.

No atual modelo, as *Turtles* são divididas em 3 *breeds* (espécies). O primeiro *breed* denominado ***healthys*** contém todas as pessoas saudáveis, que não foram contaminadas ou estão imunes, enquanto que o *breed* ***sicks*** contém aquelas que estão doentes e que contaminam outras. Por fim, temos o *breed* ***houses***, representadas por imagens de casas no ambiente, onde são definidos os locais de moradia dos grupos familiares. A quantidade de casas é definida pelo número total da população divida por 3 (vide a média da composição familiar apresentada anteriormente), logo, cada casa é composta por uma média de 3 *turtles* das duas primeiras *breeds*.

Os *Patches* dividem-se em 6 grupos: a área alheia aos setores da economia é representada pela cor **preta**; o setor serviços é representado pela cor **marrom**; o setor comércio pela cor **verde**; a indústria pela cor **lilás**; o setor de construção pela cor **laranja**; e a escola pela cor **azul**. O ambiente de simulação pode ser observado na Figura 1.

Figura 1. Interface do modelo no NetLogo



Fonte: Autoria própria.

A área de destaque amarela da Figura 1 contém os *Patches* dos setores da economia e da área alheia aos mesmos, locais onde os agentes da simulação movimentam-se. As áreas de destaque vermelha, azul e verde contém os parâmetros da simulação que podem ser modificados para gerar cenários diferentes de comparação. A área destacada na cor roxa compreende os gráficos e campos de exibição dos valores obtidos durante uma simulação.

A área de detaque vermelha compreende os parâmetros relacionados a população, sendo eles: ***population***, refere-se ao número de pessoas no ambiente; ***initial-infecteds***, o total de pessoas que iniciam a simulação infectadas; ***%-population-leak***, a porcentagem de pessoas que não respeitam o *lockdown*; ***%-asymptomatics***, a porcentagem de

pessoas assintomáticas; ***%-mask-adhrents*** a porcentagem de pessoas que são adeptas ao uso da máscara; e ***mask-effectivity***, a efetividade de proteção da máscara utilizada pela população. Além disso, é possível definir a porcentagem de cada parcela da população através dos parâmetros ***%-pop-youth***, ***%-pop-adult*** e ***%-pop-elderly***. Por fim, também é permitido configurar através do parâmetro ***imunity-duration*** a possibilidade de uma pessoa que curou-se da doença perder essa imunidade e infectar-se novamente.

A área de destaque azul contém os parâmetros dos setores, sendo eles: ***%-services***, ***%-commerce***, ***%-industry***, ***%-construction***, que representam a porcentagem da população de adultos e não aposentados presente nos setores de serviço, comércio, indústria e construção respectivamente. Além disso, é possível definir a porcentagem da população de idosos aposentados através da variável ***%-retired***.

Enquanto que a área de destaque verde apresenta os parâmetros a estratégia adotada durante uma simulação, sendo eles: ***lockdown-duration*** e ***workday-duration***, que representam a quantidade de dias que compõe o ciclo de *lockdown* e liberação das atividades no meio social. Por fim, é possível definir se determinado setor estará aberto no ciclo da estratégia utilizando os parâmetros ***open-services***, ***open-commerce***, ***open-industry***, ***open-construction*** e ***open-school***.

Por fim, a área de destaque roxa abrange 9 componentes monitores, para verificação das variáveis percentuais definidas e para o acompanhamento do número total de infectados e mortos. Ademais, são apresentados dois componentes gráficos chamados *Populations* e *Sector population*, onde o primeiro apresenta a curva do número de pessoas doentes durante o período simulado, nunca infectadas ou imunes na simulação ao longo do tempo, e o segundo apresenta a quantidade de pessoas em cada um dos setores ao longo do tempo.

2.2. Comportamento

A simulação permite criar diversos cenários de estratégia para o *lockdown* e a circulação controlada de pessoas pelo ambiente. A população pode ser de três tipos:

- Pessoas saudáveis: movem-se pela área referente ao seu setor, caso este esteja em funcionamento na estratégia definida para o cenário;
- Pessoas doentes sintomáticas: ficam em *lockdown* mesmo que o seu setor esteja em funcionamento;
- Pessoas doentes assintomáticas: movem-se pela área referente ao seu setor, caso este esteja em funcionamento, podendo infectar pessoas saudáveis do ambiente.

A probabilidade de uma pessoa infectada contaminar pessoas saudáveis foi definida de acordo com o estudo de [Liu et al. 2020], variando conforme a relação entre as duas pessoas e as suas idades. Baseado nesse estudo, o risco de contágio entre pessoas que moram na mesma casa neste modelo é de 6,4% para aqueles que possuem menos de 20 anos, 17,1% para adultos e 28% para aqueles com 60 anos ou mais. A probabilidade de transmissão secundária, ou seja, entre pessoas que não moram juntas, é de apenas 2,4%.

No caso de pessoas do mesmo grupo familiar, a transmissão pode ocorrer, levando em consideração as probabilidades definidas anteriormente, quando os agentes retornam para a casa ao fim do dia. Enquanto que a transmissão secundária, pode ocorrer dentro da área do setor a cada *tick* da simulação no período de abertura do mesmo, a depender

da distância da pessoa contaminada de uma pessoa saudável, caso as duas se cruzem no mesmo *patch* é realizado o cálculo da probabilidade de contaminação.

A taxa de mortalidade entre as pessoas infectadas foi definida de acordo com o estudo realizado pelo Centro de Controle e Prevenção de Doenças [Team 2020]. Para isso, foram definidas faixas de idade para cada grupo etário: os jovens possuem idades entre 5 e 19 anos, adultos possuem idades entre 20 e 59, enquanto que todos acima de 60 anos são considerados idosos. As probabilidades de letalidade da doença de acordo com a idade, sem levar em consideração as variantes que surgiram recentemente, podem ser observadas na Tabela 1.

Tabela 1. Taxa de letalidade de acordo com a faixa etária

Faixa etária	Taxa de letalidade
Entre 5 e 9 anos	Probabilidade de 0,1% de morte
Entre 10 e 39 anos	Probabilidade de 0,2% de morte
Entre 40 e 49 anos	Probabilidade de 0,4% de morte
Entre 50 e 59 anos	Probabilidade de 1,3% de morte
Entre 60 e 69 anos	Probabilidade de 3,6% de morte
Entre 70 e 79 anos	Probabilidade de 8% de morte
Acima de 80 anos	Probabilidade de 14,8% de morte

No modelo proposto, também foi levado em consideração a possível parcela da população que não respeita as regras de distanciamento e de *lockdown*. Essas pessoas, mesmo que o seu setor não esteja em funcionamento ou esteja infectada e seja sintomática, circulam pela ambiente, podendo infectar outras pessoas. O tamanho da parcela da população que "vazam" o isolamento é definido através da interface.

2.3. Cenários

São considerados como cenários de simulação os testes realizados com diferentes valores para os parâmetros referentes a população e o ambiente, buscando comparar os resultados e verificar quais combinações possuem o melhor efeito sobre a diminuição da taxa de disseminação.

Nos cenários definidos, foi inicializada uma população de 225 pessoas, sendo composta 15% por jovens, 60% por pessoas adultas e 25% por pessoas idosas, além de 8 pessoas já iniciarem a simulação infectadas. Além disso, 10% da população foi definida como assintomática e 75% como aderente ao uso da máscara nos meios sociais. Preliminarmente, todas as máscaras possuem 96% de efetividade.

Em relação aos setores da economia, a divisão inicial foi definida da seguinte maneira: 35% da população foi dividida para o setor de serviços, 25% para o comércio, 10% para a indústria e 30% para a construção. Além disso, 60% da população idosa foi indicada como aposentada. Todas as simulações apresentadas inicialmente possuem um ciclo de 14 dias, onde a interação social ocorre durante 4 dias e o isolamento, durante 10 dias.

Os valores citados acima podem ser modificados através da interface, de modo que os parâmetros da simulação aproximem-se mais da realidade de cada região. Os

valores definidos nesse estudo possibilitam a comparação dos resultados obtidos a partir dos parâmetros chaves de cada uma das simulações que serão analisadas na seção 3.

3. Resultados

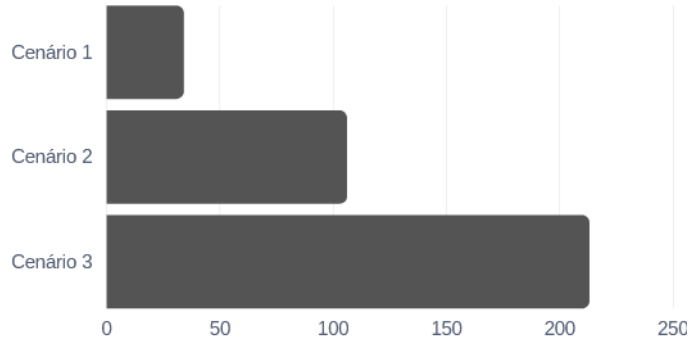
Para realizar a análise do funcionamento do modelo e os possíveis efeitos da estratégia cíclica sobre a disseminação da doença, foram definidos 11 cenários distribuídos em 3 comparações. Esses cenários utilizaram os valores definidos na subseção 2.3, modificando apenas os números e taxas dos parâmetros chaves de cada cenário a ser comparado, apontados adiante.

Através dos cenários definidos, busca-se comparar a quantidade de infectados em diferentes configurações. Na primeira comparação, serão definidos valores distintos para o vazamento do *lockdown*, enquanto que na segunda, serão analisados os efeitos de diferentes taxas de adeptos ao uso e efetividade das máscaras. Na terceira comparação, serão analisados os efeitos das diferentes combinações de aberturas de setores. Na última comparação, serão apresentados resultados obtidos ao testar com dias de *lockdown* e de trabalho diferentes dos testes anteriores. Os resultados de cada uma das comparações foram obtidos através do cálculo dos valores médios atingidos ao longo de 5 execuções em cada cenário.

3.1. Primeira comparação

Os primeiros três cenários comparados são apresentados para demonstrar a importância do cumprimento das normas de distanciamento social e do respeito ao *lockdown*. Nesses cenários, apenas dois setores foram setados como abertos, sendo eles: serviços e escola. Porém, no primeiro cenário o *lockdown* é respeitado completamente, com 0% de quebra da quarentena. Já no segundo cenário, existe um *vazamento* de 30%, sendo este considerado o limite ideal. E no terceiro cenário, existe um *vazamento* de 60%. Apenas nessa diferença de valores já é possível observar os grandes efeitos no aumento de disseminação da doença, como é demonstrado na Figura 2.

Figura 2. Média de infectados das diferentes quebras do *lockdown*



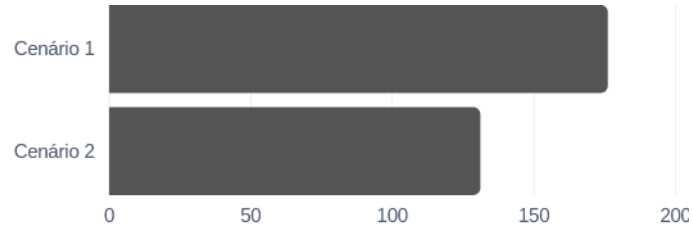
Fonte: Autoria própria.

No cenário 1, o total de infectados durante a simulação foi de apenas 34, enquanto no segundo cenário já aumentou para 106 e no terceiro e pior caso, aumentou para 213. Foi possível perceber durante a simulação que a disseminação da doença é muito mais rápida no cenário 3 do que a no cenário 2, que apesar de também ter um vazamento do *lockdown*, ele é mais contido.

3.2. Segunda comparação

Os seguintes cenários verificaram a importância da adoção do uso da máscara pela população no meio social. Para realizar essa verificação, todos os setores foram abertos. A taxa de vazamento do *lockdown* é de 30% em ambas as comparações. No cenário 1, a porcentagem de adeptos ao uso da máscara em uma população é de apenas 40%, enquanto que no cenário 2, a porcentagem é de 80%, como pode ser observado na Figura 3.

Figura 3. Média de infectados em relação ao uso de máscaras

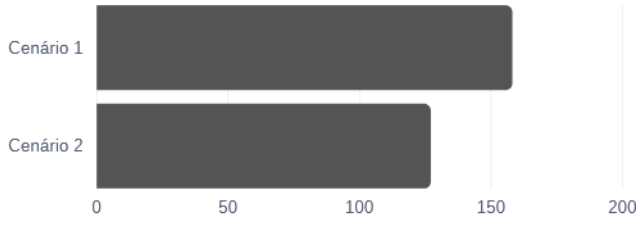


Fonte: Autoria própria.

A partir dessa comparação, pode-se perceber que apesar dos dois cenários possuírem a mesma taxa de vazamento do *lockdown*, a aderência ao uso da máscara é um fator significativo no número total de infectados. No cenário onde o uso da máscara por uma população é de apenas 40%, a média total de infectados foi de 176, enquanto que no cenário onde o uso da máscara foi de 80%, a média de infectados diminuiu para 131, ou seja, 45 pessoas infectadas a menos. Essa diferença é um número considerável, visto a rapidez da disseminação da doença e o número restrito de leitos para o tratamento dos casos mais graves.

O uso da máscara no meio social é muito importante mesmo que a pessoa não esteja sentindo sintomas, porém dependendo do material utilizado na sua confecção, a mesma pode não ter uma boa efetividade. Nas simulações a seguir, foi levado em consideração um número maior de pessoas assintomáticas, aumentando para 30%. Nesse caso, por mais que a pessoa esteja doente, ela continuará movimentando-se pelo ambiente. Para verificar como a disseminação da doença comporta-se ao utilizar máscaras, com diferentes taxas de efetividade, o total de adesão cresceu para 100%. No cenário 1, as máscaras utilizadas possuem efetividade de 80% e no cenário 2, de 96%, como pode ser observado na Figura 4. No cenário 1, uma média de 158 pessoas foram contaminadas com o vírus, diminuindo para 127 pessoas, no cenário 2, ao utilizar uma máscara com maior efetividade.

Figura 4. Média de infectados em relação a efetividade das máscaras



Fonte: Autoria própria.

3.3. Terceira comparação

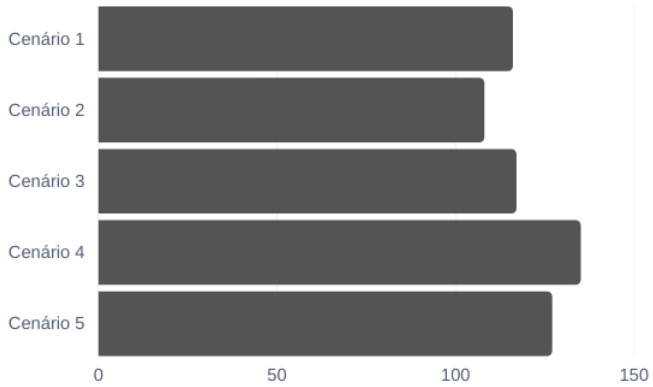
As próximas simulações utilizaram diferentes combinações de aberturas de setores para verificar os efeitos das mesmas na taxa de disseminação. Para isso, o total de assintomáticos durante essa comparação foi setado como 10%, o número de pessoas que aderiram ao uso da máscara foi de 80%, a efetividade da máscara utilizada pela população foi setada como 96% e a quebra do *lockdown* é de 30%. A divisão dessas diferentes aberturas nos cenários de simulação pode ser observada na Tabela 2.

Tabela 2. Cenários de diferentes aberturas de setores

Cenário	Setores abertos
1	Serviços, comércio e escola
2	Comércio, construção e indústria
3	Comércio, indústria e escola
4	Serviços, comércio e indústria
5	Construção, serviço e escola

É importante salientar que as distâncias dos setores no ambiente não afetam o número de contaminações. A média de infectados em cada um dos cenários pode ser visualizada na Figura 5.

Figura 5. Média de infectados em relação aos setores abertos



Fonte: Autoria própria.

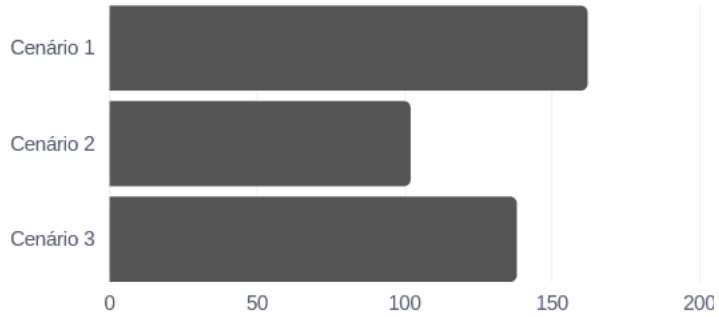
A menor média de disseminação foi obtida no cenário 2, com a abertura dos setores: comércio, construção e indústria. Isso pode ser justificado pois o ambiente de trabalho do setor de construção foi definido em uma área maior para representar um ambiente mais aberto e com menos contato pessoal. No caso da indústria, a sua área é um pouco menor que a da construção, porém maior do que as áreas dos outros setores, visto que na indústria o contato pessoal e divisão dos locais de trabalho também é menor do que nos outros setores. O maior número de infectados foi obtido no cenário de abertura dos setores: serviços, comércio e indústria, podendo ser justificado pela grande movimentação e contato nos setores de serviços e comércio.

3.4. Quarta comparação

Por fim, a última comparação será relacionada aos dias de *lockdown* e dias de trabalho utilizados na estratégia cíclica. Em todos os cenários anteriores, a divisão desses dias era

de 10 dias de *lockdown* e 4 dias de trabalho, sendo identificado nessa comparação como Cenário 1. No Cenário 2, o tempo de *lockdown* foi aumentado para 15 dias e apenas 3 dias de trabalho. Enquanto que no Cenário 3, o tempo de *lockdown* foi de 9 dias e de trabalho, 5. Além disso, os setores abertos nessa comparação são os que obtiveram a melhor média nos testes de abertura de setores apresentado anteriormente, sendo eles: comércio, construção e indústria. As médias de infectados para esses cenários podem ser observadas na Figura 6.

Figura 6. Média de pessoas infectadas em relação a quantidade de dias do ciclo



Fonte: Autoria própria.

No cenário 1, a média total de infectados durante os testes foi 162. No cenário 2, a média final foi de 102. Já no cenário 3, a média de infectados foi de 143. A partir disso, é possível perceber como a taxa de disseminação pode comportar-se ao adaptar a estratégia cíclica do *lockdown*. Apesar da comparação apresentada ser simples, a mesma demonstra que existem muitas possibilidades de testes para observação da melhor estratégia a ser aplicada em determinada população.

4. Conclusão

Uma simulação baseada em multiagentes permite estudar cenários do mundo real, comparar e observar possíveis soluções de uma determinada problemática. Nesse trabalho, foi apresentado um novo modelo para a visualização de novas estratégias de isolamento social para lidar com a pandemia causada pela Covid-19, levando em consideração fatores como a probabilidade de contaminação entre faixas etárias, regras de utilização de máscaras e distanciamento, assim como a quantidade de pessoas que não respeitam essas regras.

Foram apresentadas comparações que demonstraram as possibilidades existentes no modelo proposto. Os resultados obtidos evidenciaram a importância do cumprimento das normas de distanciamento social e do respeito ao *lockdown*, assim como do uso da máscara pela população no meio social. Além disso, esclareceram como a taxa de disseminação da doença pode obter resultados diferentes ao adaptar o *lockdown* para uma estratégia cíclica com diferentes reaberturas de setores. Os resultados alcançados foram apresentados de forma introdutória, assim como as possibilidades de combinações de reabertura de setores econômicos, de modo a demonstrar o potencial do modelo, sendo possível ajustar e analisar as diferentes variáveis do modelo para tornar o ambiente mais próximo da realidade.

Esse modelo busca contribuir para a comunidade no que se refere a modelos de propagação de doenças em uma população fechada, incorporando fatores específicos referentes ao novo coronavírus e estratégias de contenção, demonstrando através dele que a adaptação do cotidiano, com medidas de contenção e distanciamento, assim como a implantação de uma rotina de abertura e fechamento programada de setores da economia, pode obter resultados efetivos na contenção da transmissão da doença.

Como trabalhos futuros, pretende-se analisar de forma mais aprofundada os efeitos de cada uma das variáveis e adaptações do ciclo de *lockdown*, como a abertura parcial dos setores da economia e a taxa de circulação de pessoas na área. Além disso, é possível incluir em estudos futuros os efeitos da taxa de imunização através da vacina sobre a disseminação da doença, além de levar em consideração outras possíveis dinâmicas de transmissão e reinfecção da doença e os diferentes impactos sobre a população.

Referências

Atalan, A. (2020). Is the lockdown important to prevent the covid-19 pandemic? effects on psychology, environment and economy-perspective. *Annals of Medicine and Surgery*, 56:38–42.

Karin, O., Bar-On, Y., Milo, T., Katzir, I., Mayo, A., Korem, Y., Dudovich, B., Zehavi, A., Davidovich, N., Milo, R., and Alon, U. (2020). Cyclic exit strategies to suppress COVID-19 and allow economic activity. *medRxiv*, pages 1–24.

Liu, M.-J., Zhang, Z.-B., Fang, L.-Q., Yuan, J., Zhang, A.-R., Dean, N., Luo, L., Ma, M.-M., Longini, I., Kenah, E., Lu, Y., Ma, Y., Jalali, N., Yang, Z., and Yang, Y. (2020). Household secondary attack rate of COVID-19 and associated determinants in Guangzhou, China: a retrospective cohort study. *The Lancet Infectious Diseases*, 20.

Ohana, V. (2019). Ibge: 2,7% das famílias ganham um quinto de toda a renda no Brasil. `www.shorturl.at/dkU28`. Acesso em: 22-07-2021.

Saúde, S. (2021). Linha do tempo do Coronavírus no Brasil. `www.sanarmed.com/linha-do-tempo-do-coronavirus-no-brasil`. Acesso em: 22-07-2021.

Sebastien, R., Olivier, M., and Doncescu, A. (2020). Use of fuzzy sets, aggregation operators and multi agent systems to simulate COVID-19 transmission in a context of absence of barrier gestures and social distancing: application to an island region. *Proceedings of 2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 2298–2305.

Team, T. N. C. P. E. R. E. (2020). The epidemiological characteristics of an outbreak of 2019 novel coronavirus diseases (COVID-19) — china, 2020. *China CDC Weekly*, page 113.

Portabilidade dos Modelos Epidemiológicos disponíveis na MDD4ABMS para a Plataforma Repast

Fernando Santos[1], **Jéssica B. Petersen**[1]

[1] Departamento de Engenharia de Software
Universidade do Estado de Santa Catarina (UDESC)
Ibirama – SC – Brasil

fernando.santos@udesc.br, jessica041ster@gmail.com

Abstract. *MDD4ABMS is a model-driven approach for developing agent-based simulations. It provides ready-to-use, platform-independent building blocks for modeling aspects that are frequently used in agent-based simulations. One of these blocks provides epidemiological models for modeling diseases that spread among agents. MDD4ABMS offers portability of these epidemiological models by means of automatic code generation, but only for the NetLogo platform. This paper presents an extension of the MDD4ABMS approach to offer portability of epidemiological models to the Repast platform. A case study was conducted. The results obtained with the execution of the Repast and NetLogo simulations were similar, showing evidence that the portability of the epidemiological models provided by the MDD4ABMS approach is feasible.*

Resumo. *A MDD4ABMS é uma abordagem de desenvolvimento dirigido a modelos de simulações baseadas em agentes. Ela oferece blocos de construção prontos para uso e independentes de plataforma, permitindo especificar aspectos frequentemente utilizados em simulações baseadas em agentes. Um destes blocos oferece modelos epidemiológicos, para modelar doenças que se propagam através dos agentes. A MDD4ABMS oferece portabilidade destes modelos epidemiológicos através de geração automática de código, mas apenas para a plataforma NetLogo. Este artigo apresenta uma extensão da MDD4ABMS para oferecer portabilidade dos modelos epidemiológicos para a plataforma Repast. Um estudo de caso foi realizado. Os resultados obtidos com a execução das simulações Repast e NetLogo foram similares, evidenciando a viabilidade da portabilidade dos modelos epidemiológicos disponíveis na abordagem MDD4ABMS.*

1. Introdução

Simulações baseadas em agentes têm sido utilizadas para reproduzir e estudar comportamentos emergentes de sistemas complexos. Uma simulação baseada em agentes é constituída por agentes, entidades autônomas que possuem atributos e comportamentos próprios, e interagem entre si e com o ambiente. Desenvolver tais simulações é uma tarefa desafiadora, que tem sido investigada no contexto da modelagem e simulação baseada em agentes (*agent-based modeling and simulation*—ABMS) [Klügl e Bazzan 2012].

Simulações baseadas em agentes têm demonstrado sua relevância no atual contexto da pandemia de Covid-19, pois foram utilizadas para estudar a propagação do vírus

e avaliar estratégias de mitigação de infecções e mortes. Ainda no estágio inicial da pandemia em março de 2020, uma simulação baseada em agentes desenvolvida pelo Imperial College de Londres previu milhares de mortes no Reino Unido caso medidas não fossem tomadas [Ferguson et al. 2020]. Tais previsões fundamentaram a imediata adoção de medidas de contenção [Adam 2020], o que potencialmente salvou inúmeras vidas.

Recentemente foi proposta uma abordagem dirigida a modelos para desenvolver simulações baseadas em agentes denominada MDD4ABMS [Santos et al. 2018]. Esta abordagem é fundamentada na técnica de *model-driven development* (MDD), permitindo modelar simulações utilizando blocos de construção prontos para uso e independentes de plataforma. Estes blocos abstraem aspectos frequentemente utilizados em simulações, permitindo ao desenvolvedor focar *no que* simular, em vez de *em como* implementar tais aspectos. Um dos blocos de construção disponíveis na MDD4ABMS oferece modelos epidemiológicos, para simular doenças que se propagam nos agentes [Santos et al. 2020].

Por meio de geração automática de código, a MDD4ABMS oferece portabilidade de seus blocos de construção para a plataforma de simulação baseada em agentes NetLogo. Isto permite que o desenvolvedor execute a simulação especificada com a MDD4ABMS nesta plataforma. Recentemente, [Santos e Tenfen 2019] ofereceram portabilidade para a plataforma de simulação Repast, porém apenas de blocos que abstraem aspectos elementares da simulação (e.g., movimentação e sobrevivência dos agentes). Este artigo apresenta uma extensão da MDD4ABMS para oferecer portabilidade dos modelos epidemiológicos para a plataforma Repast, permitindo que desenvolvedores com alguma experiência em Repast também possam executar as simulações especificadas na MDD4ABMS. Um estudo de caso foi realizado, e os resultados obtidos com a execução das simulações Repast e NetLogo foram similares, evidenciando a viabilidade da portabilidade dos modelos epidemiológicos especificados com a abordagem MDD4ABMS.

O restante deste artigo está organizado da seguinte forma. A seção 2 apresenta a fundamentação teórica sobre modelagem e simulação baseada em agentes e a abordagem MDD4ABMS. A seção 3 apresenta a extensão desenvolvida neste artigo para suportar a portabilidade dos modelos epidemiológicos. Em seguida, a seção 4 descreve o estudo de caso realizado. Por fim, a seção 5 apresenta as conclusões e trabalhos futuros.

2. Fundamentação Teórica

2.1. Modelagem e Simulação baseada em Agentes

A modelagem e simulação baseada em agentes (ABMS) é um paradigma de simulação que utiliza agentes para analisar, reproduzir ou predizer fenômenos normalmente complexos e emergentes. Em ABMS, um modelo do sistema em estudo é construído em termos de agentes, que interagem com seus pares e também com o ambiente. A partir da simulação do modelo, através de repetida execução por determinado intervalo de tempo, pode-se obter informações significativas a respeito das propriedades do sistema ou fenômeno analisado, bem como sobre sua própria evolução [Garro e Russo 2010].

No paradigma ABMS, o desenvolvedor não possui restrições quanto a complexidade do agente que deseja especificar, sendo livre para definir técnicas sofisticadas para raciocínio e interações (por exemplo, técnicas de aprendizagem e redes neurais disponíveis na área de inteligência artificial), bem como para compor a população de agentes de forma

heterogênea. Desta forma, o pesquisador pode incorporar explicitamente na simulação a complexidade e diversidade de comportamento e interações dos indivíduos observada em cenários reais [Macal e North 2014].

De acordo com [Klügl e Bazzan 2012], uma simulação baseada em agentes é formada por três elementos: um conjunto de agentes autônomos; a especificação das interações entre os agentes e também com o ambiente, responsáveis por produzir a saída geral do sistema; e o ambiente simulado, que contém todos os demais elementos de simulação, como recursos e outros objetos sem comportamento ativo.

Simulações baseadas em agentes são frequentemente desenvolvidas em plataformas específicas, que disponibilizam recursos inerentes a agentes visando simplificar o desenvolvimento. De acordo com [Klügl e Bazzan 2012], entre as principais plataformas estão: NetLogo [Wilensky 1999], Repast [North et al. 2013] e MASON [Luke et al. 2005].

O NetLogo foi projetado para o usuário final, tendo uma interface para trabalhar com os parâmetros da simulação, e uma extensa biblioteca de modelos existentes. Além disto, NetLogo oferece uma linguagem de programação própria para desenvolver simulações. A plataforma Repast, por sua vez, permite desenvolver simulações utilizando programação Java. Através de bibliotecas Java existentes, o desenvolvedor pode incorporar na simulação quaisquer recursos utilizados de modo geral na indústria de software, como por exemplo integração com serviços web ou aprendizagem de máquina. Por fim, a MASON também é uma plataforma baseada em Java que visa facilitar a programação de simulações em larga escala para ganhar desempenho, sendo muito atrativo para simulações que demandam performance.

2.2. Abordagem MDD4ABMS para Desenvolvimento de Simulações com Agentes

A engenharia de software fornece suporte para o desenvolvimento de software profissional por meio de técnicas de especificação, projeto e evolução de sistemas. Uma destas técnicas é o desenvolvimento dirigido a modelos (*model-driven development*—MDD).

Em MDD são utilizados artefatos de modelagem que abstraem aspectros frequentemente presentes em softwares, para impulsionar o desenvolvimento de artefatos de baixo nível como códigos-fonte [Mernik et al. 2005]. O MDD facilita o desenvolvimento pois permite ao desenvolvedor focar em *quais* aspectos o software deve ter, em vez de focar em *como* desenvolver estes aspectos. A modelagem torna-se o papel chave no processo de desenvolvimento, pois é a partir do modelo que o código fonte será gerado automaticamente. Isto torna o desenvolvimento mais produtivo e eficiente, permitindo a portabilidade do software para diferentes plataformas [Mohagheghi et al. 2009].

Recentemente foi proposta uma abordagem MDD para desenvolver simulações baseadas em agentes chamada MDD4ABMS [Santos et al. 2018]. A MDD4ABMS foi elaborada a partir da análise de simulações existentes, onde foram identificados aspectos que são frequentemente utilizados em diferentes domínios. Estes aspectos foram abstraídos em um metamodelo de simulações com agentes.

A MDD4ABMS oferece a linguagem de modelagem ABStractLang, que fornece blocos de construção para os aspectos abstraídos no metamodelo de simulações. Além disso, a MDD4ABMS disponibiliza uma ferramenta de modelagem, cha-

mada ABStractme, que permite especificar simulações de acordo com a linguagem ABStractLang. A Figura 1 apresenta a ferramenta ABStractme. Na parte central está o diagrama com a especificação de uma simulação de propagação de doença, elaborado de acordo com a linguagem de modelagem ABStractLang. À direita observa-se uma paleta com os blocos de construção disponíveis para modelagem.

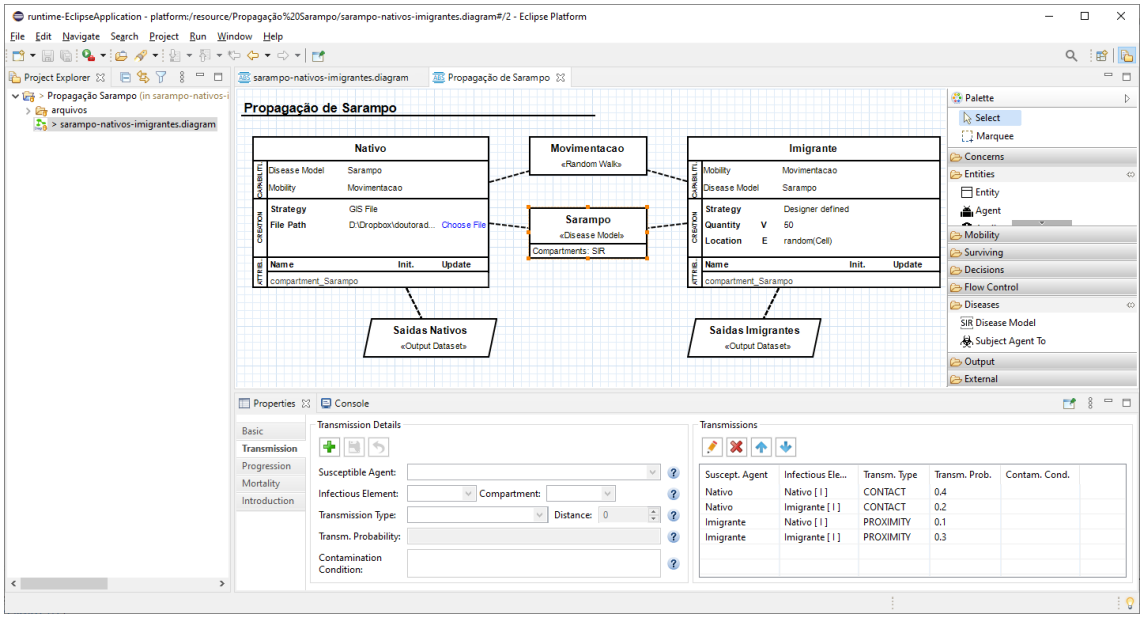


Figura 1. Simulação de propagação de doença especificada na MDD4ABMS

Os aspectos de simulações com agentes abstraídos no metamodelo da MDD4ABMS incluem: ambientes do tipo grade com inicialização a partir de arquivos georreferenciados; entidades e agentes com atributos, que podem ser inicializados a partir de dados fornecidos; coleta de dados para exibição e análise de resultados; habilidades de agentes, tais como mobilidade, sobrevivência, e aprendizagem por reforço; e modelos epidemiológicos de propagação de doenças [Santos et al. 2020].

Os modelos epidemiológicos permitem especificar a dinâmica de propagação de uma doença entre os agentes da simulação. Eles são fundamentados no modelo compartimental de [Kermack e McKendrick 1932], utilizado por epidemiologistas para prever e entender a propagação de doenças.

Estes modelos compartimentais classificam os indivíduos em compartimentos, de acordo com o estado de saúde. Por exemplo, o modelo epidemiológico elementar `SIR` considera os seguintes compartimentos: suscetível (`S`) para indivíduos ainda não infectados pela doença; infectado (`I`) para indivíduos contaminados pela doença e que podem manifestar sintomas e transmitir para outros agentes; e recuperado (`R`) para indivíduos curados e que eventualmente desenvolveram imunidade. A dinâmica da propagação da doença ocorre com a transição dos indivíduos por estes compartimentos. Essa transição é governada por parâmetros que caracterizam a doença, como por exemplo taxas de transmissão e de recuperação.

Variações do modelo `SIR` existem na literatura, sendo o modelo `SEIR` uma delas [Keeling e Rohani 2008]. No `SEIR` existe um compartimento adicional, exposto (`E`),

para indivíduos em que a doença está em período de incubação. Para doenças em que a imunidade é temporária, há modelos que especificam o retorno do indivíduo ao compartimento suscetível (S) após um período de tempo, como por exemplo os modelos SIRS e SEIRS. Os modelos SIR, SEIR, SIRS e SEIRS são contemplados pela abordagem MDD4ABMS.

A MDD4ABMS provê geração automática de código para simulações especificadas a partir de sua linguagem de modelagem, conferindo agilidade e produtividade ao desenvolvimento de simulações. Através da geração de código, os aspectos da simulação são portados para artefatos de plataformas de simulação baseada em agentes (e.g., códigos fonte e arquivos de configuração). É possível gerar código NetLogo para todos os aspectos contemplados pela abordagem MDD4ABMS [Santos et al. 2020]. Recentemente, [Santos e Tenfen 2019] ofereceram portabilidade (geração de código) para a plataforma Repast, mas apenas de aspectos elementares das simulações (ambientes do tipo grade, entidades, agentes, e as habilidades de mobilidade e sobrevivência). Este artigo apresenta uma extensão da abordagem MDD4ABMS para oferecer portabilidade dos modelos epidemiológicos para a plataforma Repast.

3. Portabilidade dos Modelos Epidemiológicos da MDD4ABMS

Para obter portabilidade dos modelos epidemiológicos para a plataforma Repast, este trabalho estendeu o gerador de código desenvolvido por [Santos e Tenfen 2019]. Este gerador é formado por regras de produção de código, que transformam os elementos do metamodelo MDD4ABMS (agentes, ambiente, etc) para elementos e blocos de comandos da plataforma Repast. As regras são descritas em Xpand,[1] uma linguagem para especificação de *templates* de geração de código. O restante desta seção descreve, de forma geral, as modificações realizadas no gerador de código Repast para contemplar os modelos epidemiológicos suportados pela MDD4ABMS.

Inicialmente, foi necessário projetar como os modelos epidemiológicos poderiam ser implementados na plataforma Repast. Tendo em vista que na Repast as simulações são desenvolvidas em linguagem Java, optou-se por adotar um projeto orientado a objetos. A Figura 2 apresenta o diagrama de classes do projeto de implementação dos modelos epidemiológicos. Na Repast, por definição, o desenvolvedor precisa implementar uma classe Java para cada tipo de agente existente na simulação. No diagrama, isto está representado pela classe AgentX (na implementação concreta seria chamada por exemplo de Humano).

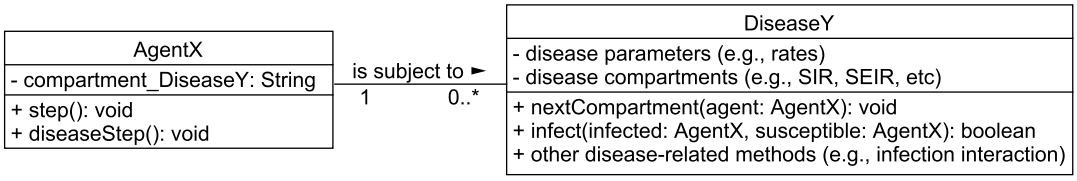


Figura 2. Projeto de implementação dos modelos epidemiológicos em Repast

Assumimos que, ao utilizar um modelo epidemiológico na simulação, o desenvolvedor está especificando uma doença que acometerá um ou vários tipo(s) de

[1] http://www.eclipse.org/modeling/m2t/?project=xpand

agente(s). Neste sentido, a doença é representada por uma nova classe, `DiseaseY`. Na implementação concreta, o nome da classe é substituido pelo nome da doença, por exemplo `Sarampo`. Durante a execução, um ou mais objetos de `DiseaseY` estarão associados a cada agente que estiver sujeito à(s) doença(s). O comportamento do agente será afetado pela dinâmica de propagação e parâmetros desta(s) doença(s).

O gerador de [Santos e Tenfen 2019] já estava gerando o código Java da classe `AgentX`. Modificações foram realizadas para gerar atributos relacionados à doença, como por exemplo o compartimento no qual o agente se encontra atualmente, e também para gerar um novo método `diseaseStep()` no agente, para ativar a dinâmica de propagação da doença no objeto `DiseaseY` associado. Além disso, a geração do método `step()` foi estendida. Este método, que implementa o comportamento do agente durante um ciclo de execução da simulação, agora ativa o método `diseaseStep()` caso o agente esteja sujeito a alguma doença.

A classe `DiseaseY` encapsula os atributos e métodos que gerenciam a dinâmica da doença. Para os parâmetros que caracterizam a doença (e.g, taxas de transmissão, recuperação, e mortalidade) são gerados atributos. Também são gerados atributos para identificar os compartimentos do modelo compartimental adotado na doença.

Dentre os métodos gerados para implementar a dinâmica da doença na classe `DiseaseY`, destacam-se: `nextCompartment()`, que determina o próximo compartimento do agente de acordo com a evolução da doença; e `infect()`, responsável por fazer com que um agente infectado contamine um suscetível quando ocorrer interação entre eles.

Outros métodos são gerados para implementar aspectos relacionados à dinâmica da doença, como por exemplo identificar as interações entre os agentes onde poderá ocorrer a infecção. Estes aspectos relacionados à dinâmica dos modelos compartimentais estão documentados em [Santos et al. 2020].

Para gerar o código da classe `DiseaseY` foi desenvolvido um novo módulo Xpand, com a implementação de regras para gerar código Java dos atributos e métodos que realizam a dinâmica da doença. A Figura 3 apresenta a principal regra Xpand, que define e gera a estrutura da classe da doença. Inicialmente a regra gera os elementos Java *package* e *imports* nas linhas 2 e 3, respectivamente. Na linha 4 é gerada a declaração da classe Java da doença. Conforme mencionado anteriormente, o nome `DiseaseY` apresentado na Figura 2 é substituído pelo nome da doença especificada no diagrama da simulação. Em seguida, são invocadas diversas regras Xpand responsáveis por gerar código para os outros aspectos da doença. Nas linhas 5 a 7 são invocadas regras para gerar os atributos, o construtor, e os métodos *getters*/*setters*. Os métodos `nextCompartment()` e `infect()` são gerados pelas regras invocadas nas linhas 8 e 11, respectivamente.

As regras ativadas nas demais linhas são responsáveis por gerar códigos que implementam os outros aspectos relacionados à doença, como por exemplo a transição entre os compartimentos do modelo epidemiológico adotado (na MDD4ABMS estes compartimentos são chamados de estados) e a identificação das interações entre agentes onde ocorre a infecção (e.g., por contato físico ou proximidade entre agentes).

```
1  «DEFINE DiseaseClasses (mm::DiseaseModel diseaseModel) FOR mm::AgentBasedSimulation»
2  package «GET_PACKAGE_NAME(this)»;
3  «EXPAND importsDiseaseClass ((DiseaseModel)diseaseModel) FOR this»
4  public class «EXPAND getClassName ((DiseaseModel)diseaseModel)  FOR this» {
5    «EXPAND diseaseAttributes ((DiseaseModel)diseaseModel) FOR this»
6    «EXPAND diseaseConstructor ((DiseaseModel)diseaseModel) FOR this»
7    «EXPAND diseaseGettersSetters ((DiseaseModel)diseaseModel) FOR this»
8    «EXPAND methodNext ((DiseaseModel)diseaseModel) FOR this»
9    «EXPAND resetDurations ((DiseaseModel)diseaseModel) FOR this»
10   «EXPAND transitionBetweenStates ((DiseaseModel)diseaseModel) FOR this»
11   «EXPAND infect ((DiseaseModel)diseaseModel) FOR this»
12   «EXPAND contactTransmission ((DiseaseModel)diseaseModel)  FOR this»
13   «EXPAND getTransmissionByInfected  FOR this»
14   «EXPAND returnAgents ((DiseaseModel)diseaseModel) FOR this»
15   «EXPAND proximityTransmission ((DiseaseModel)diseaseModel) FOR this»
16   «EXPAND returnValueOfEdgeDimension ((DiseaseModel)diseaseModel) FOR this»
17   «EXPAND getTypeTransmission FOR this»
18   «EXPAND transitionValue ((DiseaseModel)diseaseModel) FOR this»
19 }
20 «ENDDEFINE»
```

Figura 3. Regra Xpand para Gerar a Classe Java do Agente Repast

4. Estudo de Caso

Esta seção descreve o estudo de caso conduzido para demonstrar a viabilidade da portabilidade dos modelos epidemiológicos e os resultados obtidos.

4.1. Materiais e Métodos

Duas simulações de propagação de doenças foram especificadas para o estudo de caso. Uma delas utilizou o modelo epidemiológico `SIR`, e a outra o modelo `SEIR`. A partir do modelo da simulação, especificado a partir da ferramenta ABStractme da MDD4ABMS, foram gerados código fonte para as plataformas Repast e NetLogo. As simulações foram executadas, e os resultados obtidos nestas plataformas foram comparados. O resultado obtido com a NetLogo é considerado *baseline*, tendo em vista que a consistência da sua geração de código já foi abordada por [Santos et al. 2020]. A portabilidade será obtida com sucesso caso os resultados das simulações Repast e NetLogo sejam similares quando executados com os mesmos valores de parâmetros.

As simulações desenvolvidas consideram a propagação de Sarampo entre dois tipos de agentes, arbitrariamente denominados de Nativos e Imigrantes. Estes agentes podem se movimentar em um ambiente no formato de grade. A modelagem desta simulação está retratada na Figura 1, apresentada como exemplo de modelagem na seção 2.2.

A Tabela 1 apresenta a quantidade de agentes por tipo, e também os valores dos parâmetros utilizados nas simulações. Valores arbitrários foram utilizados para as quantidades de agentes e os parâmetros que caracterizam a disseminação, recuperação, e mortalidade da doença simulada, pois o objetivo do estudo não é analisar a propagação da doença em uma situação real de epidemia, mas sim avaliar a consistência do gerador de código desenvolvido e consequentemente a viabilidade da portabilidade para a plaforma Repast. Neste sentido, optou-se por introduzir a doença e estabelecer uma taxa de mortalidade apenas para um tipo de agente.

Com relação às taxas de transmissão, também foram utilizados diferentes valores para cada possível interação entre os agentes. Na Tabela 1, estas interações e taxas

Agente	Qtd.	Infectados*	Taxas de transmissão	Taxa mortalidade	Modelo	Durações
Nativo	600	nenhum	Nativo: 0.4 (contato)	0.2	SIR	I: 20
			Imigrante: 0.3 (proximidade: 2)		SEIR	I: 20 E: 3 R: 25
Imigrante	100	5	Imigrante: 0.2 (proximidade: 1)	nenhuma	SIR	I: 15
			Nativo: 0.1 (contato)		SEIR	I: 15 E: 5 R: 30

* Infectados no início da simulação

Tabela 1. Parâmetros utilizados nas simulações

devem ser interpretadas da seguinte maneira: o agente da coluna *Agente* transmite para os agentes (interage) da coluna *Taxas de transmissão* de acordo com a taxa informada nesta coluna. Na MDD4ABMS esta interação pode ser por contato, quando os agentes ocupam a mesma posição no ambiente, ou por proximidade, quando estão situados até determinada distância. Por fim, a tabela apresenta as durações dos compartimentos para os modelos SIR e SEIR utilizados nas simulações. Estas durações correspondem à quantidade de passos da simulação que o agente permanecerá nos compartimentos.

As simulações Repast e NetLogo geradas a partir da especificação em MDD4ABMS foram executadas 20 vezes cada. As quantidades de agentes suscetíveis, infectados, expostos e recuperados, bem como a quantidade total de agentes, foram coletados a cada passo da simulação, e por fim foram calculadas a média e desvio padrão destas quantidades. A simulação com o modelo SIR foi executada por 90 passos. Já a simulação com o modelo SEIR foi executada por 210 passos para verificar a reincidência da doença nos agentes, pois nesta simulação foi especificada uma duração para o compartimento R, tornando a imunidade temporária.

4.2. Resultados com o Modelo Epidemiológico SIR

As Figuras 5 e 4 apresentam a média e desvio padrão dos resultados obtidos para a simulação SIR para cada tipo de agente. Os resultados obtidos na Repast são apresentados em linhas com cores intensas, e os obtidos na NetLogo em linhas com cores claras (e.g., verde intenso e verde claro).

Nos agentes Imigrantes observa-se um pico de infecções por volta do passo 20, que decai a medida que os agentes se recuperam da doença. A quantidade total de agentes Imigrantes permanece constante, pois para estes agentes não há taxa de mortalidade.

Por outro lado, no caso dos agentes Nativos, o pico de infecções ocorre por volta do passo 37 em razão da maior quantidade de agentes deste tipo, o que faz com que a doença circule e infecte maior quantidade de agentes. Como agentes Nativos estão sujeitos a uma taxa de mortalidade, sua quantidade decai ao longo da simulação.

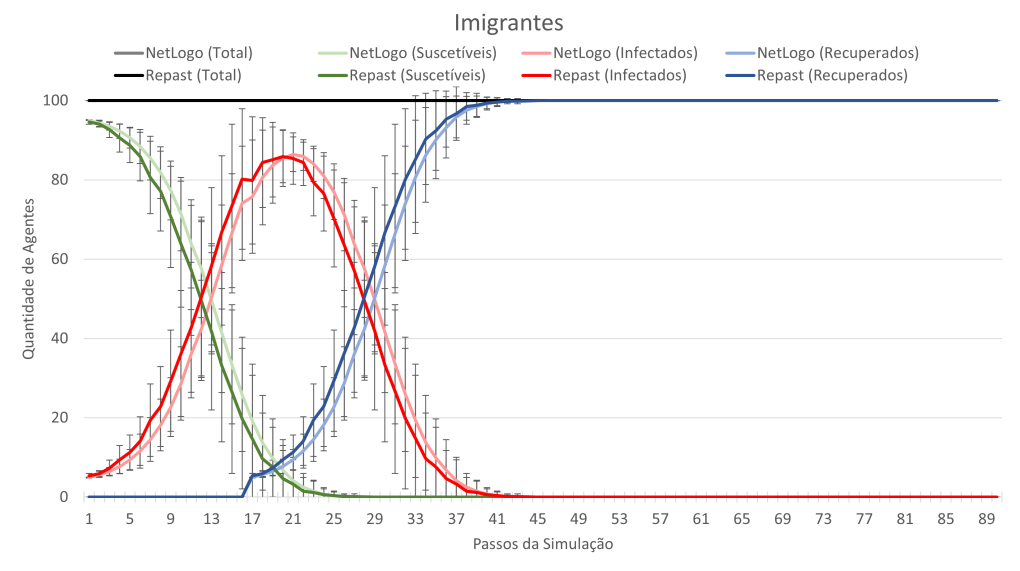


Figura 4. Resultados com o modelo epidemiológico SIR e agentes Imigrantes

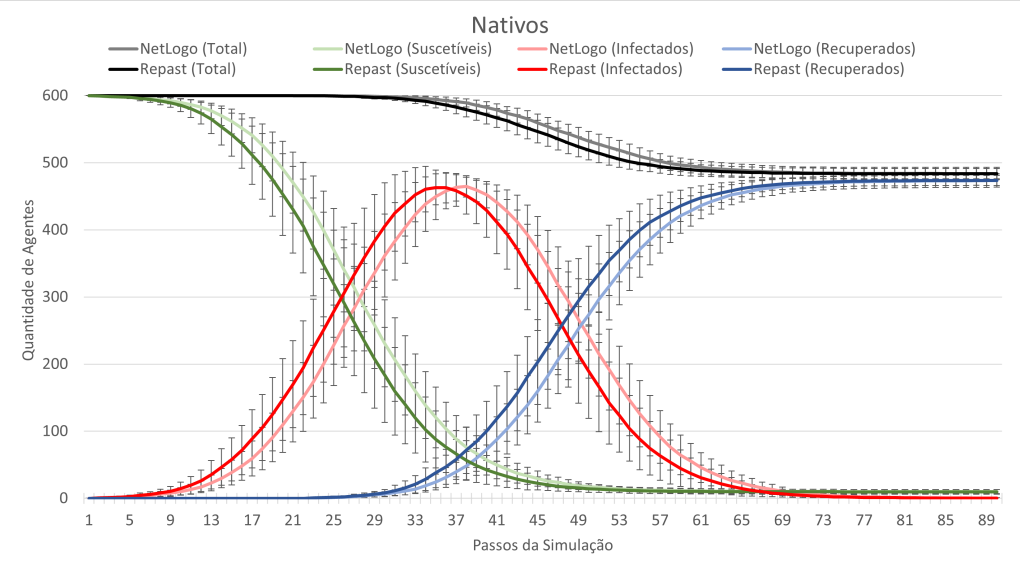


Figura 5. Resultados com o modelo epidemiológico SIR e agentes Nativos

4.3. Resultados com o Modelo Epidemiológico SEIR

As Figuras 7 e 6 apresentam a média e desvio padrão dos resultados obtidos para a simulação `SEIR`, utilizando também a representação de cores intensas e claras para Repast e NetLogo, respectivamente.

Em relação à simulação anterior, notam-se duas principais diferenças. A primeira é a existência de agentes expostos, que se encontram no compartimento `E` (linhas amarelas). Os agentes contaminados pela doença permanecem neste compartimento por um período de tempo até evoluirem para o compartimento `I`. A segunda difença é a oscilação nas quantidades de agentes em cada compartimento. Isto ocorre pois nesta simulação

ambos agentes possuem imunidade temporária, retornando ao compartimento S após permanecer certo tempo no compartimento R. Estas oscilações representam as *ondas* de contaminação por uma doença. Nos agentes Imigrantes, as ondas possuem amplitude similar ao longo da simulação pois não há mortalidade de agentes. Já no caso dos agentes Nativos, as amplitudes são diferentes pois há mortalidade de agentes.

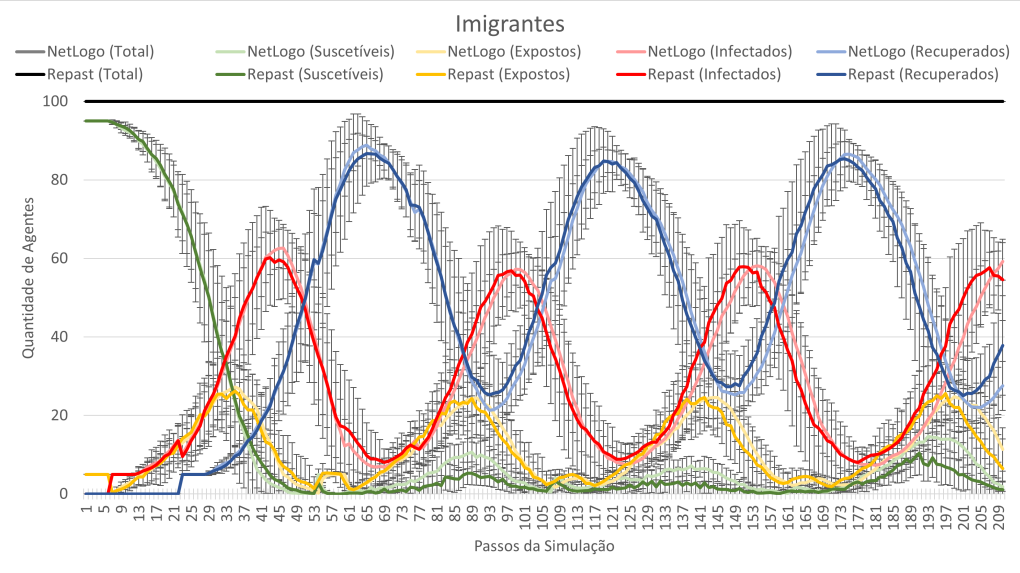


Figura 6. Resultados com o modelo epidemiológico SEIR e agentes Imigrantes

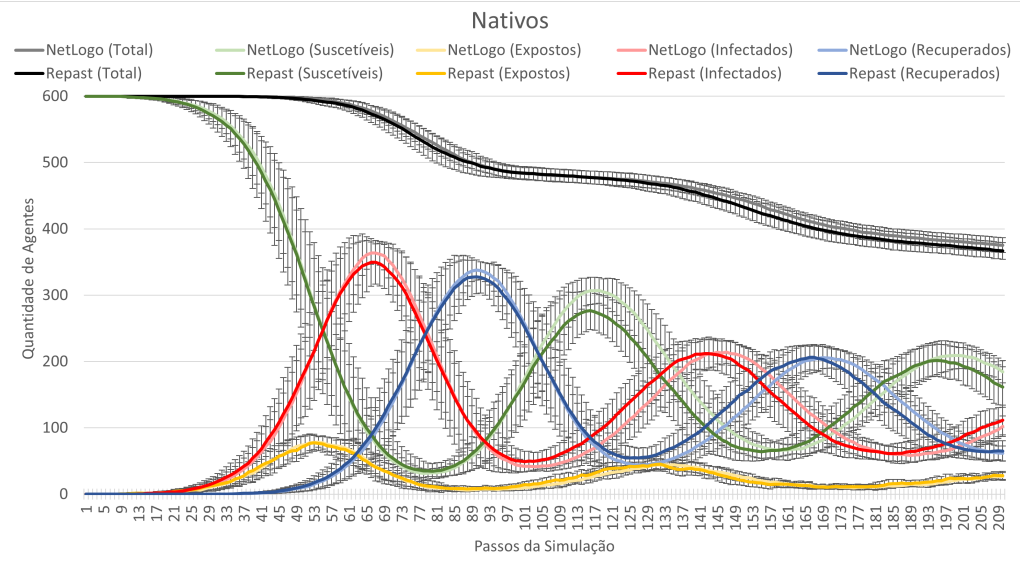


Figura 7. Resultados com o modelo epidemiológico SEIR e agentes Nativos

Em ambas as simulações, observa-se que os resultados obtidos na Repast coincidem, dentro das margens indicadas pelos desvios padrão, com os resultados obtidos na NetLogo (*baseline*). Isto evidencia sucesso na portabilidade dos modelos epidemiológicos disponíveis na plataforma MDD4ABMS para a plataforma Repast.

5. Conclusão

Este trabalho estendeu a abordagem MDD4ABMS para suportar a portabilidade dos modelos epidemiológicos para a plataforma Repast. A portabilidade foi obtida por meio de modificações no gerador de código Repast existente, além do desenvolvimento de um novo módulo para gerar código dos aspectos relacionados à dinâmica dos modelos epidemiológicos. Essa portabilidade pode ser útil em meios acadêmicos e profissionais, pois oferece maior liberdade aos interessados em simulações com agentes, fomentando a adoção do paradigma de modelagem e simulação baseada em agentes.

Um estudo de caso foi realizado, considerando a modelagem de duas simulações de propagação de doenças. O gerador de código desenvolvido foi utilizado para gerar código fonte Repast destas simulações. O resultado da execução destas simulações foi similar ao resultado obtido na plataforma NetLogo, evidenciando a viabilidade da portabilidade dos modelos epidemiológicos especificados com a abordagem MDD4ABMS para a plataforma Repast.

Como trabalhos futuros, sugere-se melhorias no gerador de código para suportar modelos epidemiológicos customizados. Nestes modelos customizados o desenvolvedor pode definir novos compartimentos para customizar a dinâmica da propagação da doença, para atender casos em que seja necessário a adoção de modelos compartimentais ainda não contemplados pela MDD4ABMS. Ainda que a abordagem MDD4ABMS suporte a especificação de modelos epidemiológicos customizados, tanto o gerador de código Repast quanto NetLogo não suportam a sua geração de código. Além disso, outra sugestão de trabalho futuro é evoluir a própria abordagem MDD4ABMS para suportar a especificação de vetores de transmissão de doenças. Vetores de transmissão são agentes que não estão sujeitos aos sintomas ou mortalidade, apenas estão infectados e transmitem o patógeno causador da doença. Exemplos de vetores são mosquitos (dengue) e carrapatos (febre maculosa).

Referências

Adam, D. (2020). Special report: The simulations driving the world's response to COVID-19. *Nature*, 580(7803):316–318.

Ferguson, N. M., Laydon, D., Nedjati-Gilani, G., Imai, N., Ainslie, K., Baguelin, M., Bhatia, S., Boonyasiri, A., Cucunubá, Z., Cuomo-Dannenburg, G., Dighe, A., Dorigatti, I., Fu, H., Gaythorpe, K., Green, W., e Hamlet, A. (2020). Report 9: Impact of non-pharmaceutical interventions (NPIs) to reduce COVID19 mortality and healthcare demand. Technical report, MRC Centre for Global Infectious Disease Analysis, London.

Garro, A. e Russo, W. (2010). easyABMS: A domain-expert oriented methodology for agent-based modeling and simulation. *Simulation Modelling Practice and Theory*, 18(10):1453–1467.

Keeling, M. J. e Rohani, P. (2008). *Modeling infectious diseases in humans and animals*. Princeton University Press.

Kermack, W. O. e McKendrick, A. G. (1932). Contributions to the mathematical theory of epidemics. ii.—the problem of endemicity. *Proc. R. Soc. Lond. A*, 138(834):55–83.

Klügl, F. e Bazzan, A. L. C. (2012). Agent-based modeling and simulation. *AI Magazine*, 33(3):29–40.

Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., e Balan, G. (2005). Mason: A multiagent simulation environment. *Simulation*, 81(7):517–527.

Macal, C. e North, M. (2014). Introductory tutorial: Agent-based modeling and simulation. In *Proceedings of the 2014 Winter Simulation Conference*, WSC '14, pages 6–20, Piscataway, NJ, USA. IEEE Press.

Mernik, M., Heering, J., e Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344.

Mohagheghi, P., Dehlen, V., e Neple, T. (2009). Definitions and approaches to model quality in model-based software development – a review of literature. *Information and Software Technology*, 51(12):1646–1669.

North, M. J., Collier, N. T., Ozik, J., Tatara, E. R., Macal, C. M., Bragen, M., e Sydelko, P. (2013). Complex adaptive systems modeling with repast simphony. *Complex Adaptive Systems Modeling*, 1(1):1–26.

Santos, F., Nunes, I., e Bazzan, A. L. (2020). Quantitatively assessing the benefits of model-driven development in agent-based modeling and simulation. *Simulation Modelling Practice and Theory*, 104:102–126.

Santos, F., Nunes, I., e Bazzan, A. L. C. (2018). Model-driven agent-based simulation development: a modeling language and empirical evaluation in the adaptive traffic signal control domain. *Simulation Modelling Practice and Theory*, 83:162–187.

Santos, F. e Tenfen, R. (2019). Extensão da abordagem de desenvolvimento dirigido a modelos de simulações com agentes (MDD4ABMS) para suportar portabilidade de simulações. In *Anais da III Escola Regional de Engenharia de Software*, pages 33–40, Porto Alegre, RS, Brasil. SBC.

Wilensky, U. (1999). NetLogo. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.

A Brief Overview of Social Dilemmas: From Social Sciences to Multiagent Based Simulations

Rafael M. Cheang[1,2], **Anarosa A. F. Brandão**[1], **Jaime S. Sichman**[1]

[1]Laboratório de Técnicas Inteligentes (LTI)
Escola Politécnica (EP)
Universidade de São Paulo (USP)

[2]Centro de Ciência de Dados – Universidade de São Paulo

{rafael_cheang, anarosa.brandao, jaime.sichman}@usp.br

Abstract. *Social dilemma is the name given to a set of games with conflicting individual and communal incentives and characterized by the existence of deficient equilibria. This area of research has for a long time drawn the attention of the social sciences, but not as much from the multiagent community. This paper aims to introduce this multidisciplinary area of research to computer scientists and engineers with a particular interest in multiagent systems. To this end, we present an overview of how the field has evolved over the last 40 years that includes: an introduction, key learning points coming from the social sciences, how social dilemmas are being simulated in multiagent Markov game settings, and the road ahead; difficulties and perspectives.*

1. Introduction

Great attention has recently been given to reinforcement learning (RL) algorithms and their prowess to learn from experience in complex environments. Milestones such as beating the best human players in competitive games[1] with huge state spaces, such as Go [Silver et al. 2016] and chess [Silver et al. 2018], are just a glimpse to show how much AI, and especially, RL, can help humans to solve problems that involve decision making under uncertainty.

Less attention, however, has been given by the multiagent systems community to an important subset of games that are not solely competitive but rather mixed motive. This collection of games, often called social dilemmas, has drawn considerable interest from social sciences‘ scholars [Dawes 1980] once it maps well into real-world social problems.

The importance of solving this class of problems is epitomized in one sentence by Dafoe et al.: ”There are multiple problems of global cooperation with stakes in the trillions of dollars or millions of lives, including those of disarmament, avoiding nuclear war, climate change, global commerce, and pandemic preparedness” [Dafoe et al. 2020].

However, dealing with these problems is not trivial. The mixed incentives nature of social dilemmas poses difficulties in guiding decision-making by pulling incentives to different and opposed directions: players are exposed to a short-term incentive to act greedily, which in turn, may compromise a long-term incentive to cooperate that would be beneficial to all involved.

[1]Also known as zero-sum games.

This paper aims to turn the attention of the multiagent systems community to this non-zero-sum set of games that we encounter with some frequency in our everyday lives. To achieve our goal, we first introduce the idea behind social dilemmas and how research on the domain progressed, especially through the 80s and 90s, based on a high-level abstraction of the problem — a matrix and a reward function. We then follow with a short review of studies based on a lower-level formalism that uses modern RL algorithms and pixel environments. We present the differences between models, difficulties encountered when switching from high-level to low-level formalisms, and finish by discussing some open problems in social dilemmas.

2. Delving deeper into social dilemmas

A social dilemma may be described as a game that exposes its players to simultaneous and conflicting cooperative and competitive incentives. The game is cooperative in the sense that a high collective reward can be achieved by mutual cooperation, and it is competitive because every player has an individual incentive not to cooperate, either by fear of the opponent not cooperating or greed. Every social dilemma is marked by at least one *deficient equilibrium* [Kollock 1998]; selfish incentives lead participants to a state where no individual is better off changing its behavior, even though a more fortunate state for everyone exists and is reachable.

Consider a 2-player symmetric game — a game where players share the same set of legal actions and reward function — where players can choose between cooperate (C) and defect (D). For every combination of actions $\{C_1C_2, C_1D_2, D_1C_2, D_1D_2\}$ there exists a corresponding reward that we shall name, from player 1‘s perspective and in order, reward (R), sucker (S), temptation (T) and punishment (P). In these settings, it is possible to define a social dilemma by adjusting the relative values of the rewards.

Let‘s take the most famous social dilemma as an example. The Prisoner‘s dilemma (PD) is a 2-player symmetric game, formally defined by the relative reward values such that the inequalities $T > R > P > S$ and $2R > S + T$ are satisfied. The background story accompanying this formal definition is a tale of two outlaws caught by the police and sent to different rooms for interrogation. Each prisoner has the option to snitch the partner (defect), for which he would receive a lesser punishment at the cost of a more severe punishment to his accomplice, or keep quiet (cooperate). Since the punishment for being snitched is greater than the penalty reduction for snitching, mutual cooperation is preferable to mutual defection.

If we use the values $R = 2$, $T = 3$, $S = 0$ and $P = 1$, it is straightforward to see how the dilemma plays out. Both players have the individual incentive to defect regardless of what the other will do, since the value they get for defecting is greater, either in case the other chooses to cooperate — $T = 3 > R = 2$ — or defect — $P = 1 > S = 0$. We call defect in the Prisoner‘s dilemma game a *dominating strategy*, that is, a strategy that yields greater reward than any other [Axelrod 1984]. In case both players choose the dominating strategy, they both get a reward of 1, even though there exists a more advantageous scenario for both players; one where they would have chosen to cooperate, and for that, would have gotten a reward of 2.

It is possible to create two other social dilemmas by changing the relative payoffs of R, T, S, and P [Kollock 1998]. If they satisfy the inequality $R > T > P > S$ we have

the game of *assurance* or *stag hunt*, while if they satisfy the inequality $T > R > S > P$ we have the game of *chicken*. Note that both games are still considered social dilemmas since they both have at least one deficient equilibrium.

3. Axelrod‘s tournaments

We call the one-time iteration of a Prisoner‘s dilemma a *one shot* PD, and there is not much to learn from this atomic interaction; as we have seen, the optimal strategy is to defect. The one-shot PD paints a bleak picture for our hopes of achieving mutual cooperation, but it certainly does not paint the whole picture.

In case the relationship between two players is extended, by incrementing the number of future encounters between them, the choice to defect cease to be trivial. We call the successive iterations of a PD game a *repeated* Prisoner‘s dilemma (rPD) [Gotts et al. 2003]. The difference between the one-shot and repeated games comes from the fact that, in the repeated case, any player can use previous interactions to guide future decisions. For instance, player 1 may punish player 2 in a future round for not cooperating in this current round. This ”shadow of the future” effect [Axelrod 1984] changes the dilemma in its core; the decision to C or D has to be made considering the effect it might have on the opponent‘s decision in future iterations.

The natural question that emerges is: what is the dominating strategy to guide one‘s decision in such a scenario? There is no dominating strategy that is independent of the opponent‘s strategy [Axelrod 1984]. We can quickly build a counterexample that works as proof: Suppose we play against an opponent that always defects (ALL-D). The best strategy we can use is to mirror our opponent‘s strategy and also always defect. Now, suppose we play against a player that always cooperates until defected against. In this case, our best option is to cooperate for at least a while before we think about defecting — if we think about defecting at all.

However, to say that there is no best strategy independent of the opponent‘s strategy does not mean that all strategies are equally good. In a seminal body of work, political scientist Robert Axelrod promoted two tournaments where game theorists could send strategies to play against one another [Axelrod 1980a] [Axelrod 1980b]. The study was an attempt to shed light on the question of how to properly behave in an rPD so as to maximize individual return [Axelrod 1980a].

The first tournament was run in a round-robin format, such that each of the fourteen entries would play against one another, against a RANDOM strategy, and against itself. Each game consisted of 200 iterations. Surprisingly, the winner of that tournament was the simplest of all strategies submitted, called TIT-FOR-TAT (TFT), submitted by professor Anatol Rapoport [Axelrod 1980a]. TFT cooperates in the first round and copies the action taken by the opponent in the previous round henceforth. Phrasing it differently, TFT is a purely responsive strategy, and it matches cooperation with cooperation and defection with defection, the response delayed by one round.

The second tournament had a similar format to the first, with the exception that the game had a small probability of ending in each given move, so strategies couldn’t exploit the fact that they knew when the game was going to end and throw sneaky end-game defections. This tournament had 62 entries from 6 different countries, ranging from a 10-year-old computer hobbyist to professors of computer science, economics, psychology,

mathematics, political science, and evolutionary biology [Axelrod 1980b]. Once again, TFT claimed first place.

3.1. Key learning points

As one can imagine, it was not by fate or luck that TFT did so well in both tournaments. Axelrod points out four properties shared by successful strategies, including TFT [Axelrod 1984]:

- *Niceness:* Successful strategies are never the first to defect. Nice strategies do especially well against other nice strategies since the threat of exploitation does not exist.
- *Forgiveness:* Successful strategies can forgive the opponent and cooperate after being defected against. Forgiveness is of interest to the forgiving part since not forgiving might mean missing out on future mutual cooperation.
- *Provocable:* Successful strategies are provocable, that is, they can retaliate against an opponent‘s act of defection immediately after the fact. Retaliation may signal to others low tolerance towards defection.
- *Clear:* Successful strategies are clear and simple to understand.

These four properties present a general summary of what is needed for an all-around strategy. Note that for a strategy to be successful, it needs to perform well against different types of strategies, including those that are exploitative, overly-forgiving, overly-provocable, etc. TFT has all four of these properties and proved to be robust in all match-ups it encountered [Axelrod 1980b].

4. N-player social dilemmas: public goods and the tragedy of commons

The 2-player social dilemma framework is a good primer to the problem of conflicting individual and collective incentives. However, it is not sufficient to represent the whole range of dilemmas we face in the real world, especially those at the communal level. A natural extension to it is the n-player social dilemma framework, most often represented by two metaphorical stories of ”mythic” proportions [Kollock 1998]: Public Goods and The Tragedy of the Commons. Similar to the 2-player case, both games may be represented as a reward tensor that maps every combination of cooperation and defection to a reward for each participant.

4.1. Public goods

The public goods dilemma is named after a set of problems where a group of individuals needs to pay an upfront cost to utilize a public resource. Once again, every individual has the incentive not to pay the fee and free-ride, though, if everyone does so, there will be no public good to enjoy from [Kollock 1998].

A relatable example of a public good dilemma comes from the municipal tax system. The local government employs the money from our taxes, among other expenditures, to improve public spaces that every resident has the right to use. The lack of payment from one resident will most likely not affect the maintenance of our roads and parks, but if tax evasion becomes a norm, the community — especially those who paid their taxes — will be punished by having public spaces that are not well maintained [2].

[2]We are simplifying the problem to make our point and not accounting for the fact that if you don‘t pay your taxes, you are likely to be punished, which in the real world changes the payoff matrix.

4.2. Tragedy of the commons

The tragedy of the commons is a term introduced by Garret Hardin [Hardin 1968] to describe a set of social problems that cannot be solved solely by technological advances; instead, a behavior change is needed. Like the public goods dilemma, the tragedy of the commons is a group dilemma. Still, unlike the former, it is associated with individuals being incentivized to increase own short-term payoff at the cost of inflicting a punishment to everyone in the group.

Hardin himself describes a didactic example in his original article. A group of herders, having access to a common piece of land, can allow as many of their cows graze on it. Every herder has the individual incentive to let as many cows in, but if all herders behave accordingly, the grass will soon be depleted, and the cows will have nothing to eat [Hardin 1968].

4.3. Theoretical predictions and empirical evidence of n-player dilemmas

The theoretical findings regarding the group dilemmas described in this section point to a non-endogenous resolution to this set of problems [Hardin 1968, Ostrom 2000]. The theoretical framework is based on three assumptions *a)* Resource users are norm-free utility maximizers with no bounded rationality; *b)* Designing rules to change incentives is an easy task; *c)* The resolution to these problems demands intervention from a central authority [Ostrom 1999].

Nonetheless, empirical work has provided evidence that these three assumptions don‘t conform with reality. In practice, many experimental studies have shown instances of n-player dilemmas being solved by local communities, without the need of a regulatory central authority, and that social norms play a significant role in solving them, as shown by Economics Nobel Prize winner Elinor Ostrom [Ostrom 1999, Ostrom 2000]. The theoretical endeavor that tries to bridge the gap between the expected defection of participants and empirical evidence taps from evolutionary theories and points towards a complex system framework, with agents of multiple personalities interacting and being able to adjust behavior based on outcome [Ostrom 1999, Ostrom 2000].

5. Agent-based simulations of social dilemmas

The 2-player and n-player social dilemmas presented so far are well structured — they are a simple and deterministic mapping from actions to rewards — which makes them good candidates for agent-based modeling. This section presents a small sample of studies that made use of agent-based simulation to examine some properties of social dilemmas.

Deadman used the multiagent framework Swarm to simulate adaptive agents with limited rationality in a common-pool resource experiment [Deadman 1999]. Agents in the simulation were conceived as having two components: a model of the environment that consisted of state transitions given past actions, and a model of themselves, that contained instructions for generating behavior. Significant variance in strategies‘ performance was noted as a function of agents‘ recent actions, and no single strategy emerged as dominant.

Izquierdo et al. used a linear stochastic model of reinforcement learning to study the dynamics of two-player, stochastic-strategy social dilemmas [Izquierdo et al. 2008]. In each round, players had to choose whether to cooperate or defect and revise their current probabilities based on a threshold value called *aspiration level*. The revision would

be upwards if the payoff stayed above the aspiration level and downwards otherwise. They report two types of attractors within these systems; a self-reinforcing equilibrium, when agents enter a virtuous cycle of mutual cooperation, and a self-correcting equilibrium, loosely defined as a stable equilibrium in the vicinity, i.e., trajectories that get sufficiently close will be drawn to it as time approaches infinity.

Guerberoff et al. studied the effects of language representation expressiveness in spatial rPD simulations [Guerberoff et al. 2011]. Cooperation between agents increased as expressiveness grew in complexity. Queiroz and Sichman used parallel computing to extended this work and tested the effects of environment variables such as grid size, mutation rate, and error rate [Queiroz and Sichman 2016].

Barbosa et al. employed RL techniques and agents with varying cognitive capabilities — defined as the size of the state space over which strategies can be learned — in an n-player Prisoner‘s dilemma game [Barbosa et al. 2020]. They tested the effects of environment variables, learning variables, and agents‘ cognition on the outcome of an n-person Prisoner‘s dilemma simulation. We highlight the results on the impact of cognition on cooperation rates. Surprisingly, there is no positive correlation between an increase in cognition, as defined, and the rate of cooperation after the cognitive capacity surpasses a small threshold.

6. Modelling social dilemmas as Markov Games

We refer to the matrix formalization of social dilemmas presented thus far as the high-level model (HLM). The justification behind this label comes from the series of abstractions needed to fit real-world problems into this formalism. Leibo et al. summarizes some key differences: *a)* In the real world, the set of actions is not a clear dichotomy made of cooperation and defection. Cooperation and defection are rather implicit in one‘s actions and may occur in a continuum. *b)* In the real world, actions are not perfectly synchronous, and minimal asynchronies may signal one‘s intentions to others, which may, in turn, change their behavior. *c)* The world is a complex place, and hardly ever all information needed to guide one‘s decision is available [Leibo et al. 2017].

More recently, a model that circumvents the issues above has been proposed to tighten the gap between HLMs and the real world. Hereafter, we refer to it as the low-level model (LLM), since it captures with greater detail the process of a human dealing with a real-world social dilemma. The model is based on the Markov game framework, which is an extension of the Markov Decision Process (MDP) for the multiagent setting [Littman 1994].

A Markov game is generally defined by a set of states S, a collection of action sets $A_1, A_2, ...A_n$ one for each agent, a transition function $T : S$ x A_1 x A_2, ..., $A_n \longrightarrow$ PD(S), that maps combinations of states and actions to a probability distribution over S, and a reward function $R : S$ x A_1 x A_2, ..., $A_n \longrightarrow \mathbb{R}^n$ that yields an immediate reward to each agent. Each agent acts so as to maximize its expected long term reward, discounted by a factor of γ at each time step, that is, $\mathbf{E}\left[\sum_{j=0}^{\infty} \gamma^j r_{t+j}\right]$.

From now on, we turn our attention to a series of studies done using the LLM to characterize the dilemmas. The work reviewed in this section present how mixed-motive games are being simulated with the advent of RL algorithms and reveal a new and exciting

avenue of research. They are mainly divided into two categories: *a)* studies that try to test some hypothesis about a social dilemma by exploring the dynamics of the system, and *b)* studies that propose a new agent architecture and most often try to induce reciprocity to achieve mutual cooperation. Before we start the review, we introduce the environments used in the simulations as we believe it will facilitate the main ideas present in a LLM.

6.1. Environments

The first environment we describe, named *commons* [Pérolat et al. 2017], mimics a real-world tragedy of the commons, and consists of a 2D grid world where agents harvest apples, represented by colored cells, to earn rewards. Each cell has an apple spawn rate proportional to the number of apples left in its vicinity, so the fewer apples, the lower the spawn rate. Agents within this environment choose actions such as: moving up, down, left, right, rotating left and right, and collect apples by stepping into cells that contain one. They can also inflict a cost to other agents by shooting a laser beam that tags others out of the game for a predefined number of time steps. In order to maximize own reward, agents need to manage their greed so as not to deplete the environment completely.

The second environment used in simulations, named *coins* [Lerer and Peysakhovich 2018], mimics an n-player Prisoner‘s dilemma. The environment consists of a 2D grid world where agents have to collect coins that spawn in random cells at a fixed rate to receive an immediate reward. Agents and coins in the environment have a color property, and each agent receives a reward of +1 every time it collects a coin and a punishment of -1 every time another agent collects a coin that shares its color property. Actions are agent-centered and very similar to the commons game. Cooperation in the coins game means only collecting coins of the same color as its own as this behavior maximizes total utility.

The third environment, named *wolfpack* [Leibo et al. 2017], resembles the stag hunt game briefly described in section 1.1. The game once again occurs within a 2D grid world, and the players‘ action set is similar to the previous games. The n-players are divided into two groups: the wolves and the prey, and the game unrolls as the wolves try to catch the prey. Once the prey is caught, rewards are distributed to the wolves proportional to the number of wolves in the region the catch took place. This game is different in its payoff structure than the others; here, the reward received for mutual cooperation is greater than the one obtained for unilateral defection.

Finally, the *clean-up* environment [Leibo et al. 2017] mimics the public goods game, and similar to the commons game, agents earn rewards by harvesting apples. The apples‘ spawn rate in this environment is inversely proportional to the pollution of a nearby river and drops to zero once a threshold value is surpassed. Pollution increases with fixed probability over time, demanding agents to clean it up every now and then so apples keep spawning. Every time an agent cleans the river, it spends time it could otherwise be harvesting apples, thus the dilemma.

6.2. Dynamics of the system

Leibo et al. proposed a framework entitled Sequential Social Dilemmas, based on empirical game-theoretic analysis, that is built on top of the Markov game framework [Leibo et al. 2017]. The framework considers cooperation and defection as policy properties instead of atomic actions and preserves the payoff structure seen in the matrix-based

formalism for each game. A policy is deemed to be cooperative or defective based on a *social behavior metric* value, which is independent of the state value function. The study used the framework to analyze the learning dynamics of the system in two environments: wolfpack and commons. In the commons game, the social behavior metric used was the frequency of laser beams shot. Results pointed to an escalation in conflict as the resources became scarcer and conflict costs increased. The experiment on the wolfpack environment used wolves‘ closeness as its social metric. Two different cooperative policies emerged from the simulations: one where wolves looked for each other before hunting the prey and the other, where a wolf would first find the prey and then wait for the other wolf to arrive.

Perolat et al. also tested the learning dynamics of simulations in the commons environment and analyzed the emergent social outcomes in each learning stage [Pérolat et al. 2017]. Agents in the simulation learned through self-play via a Q-learning algorithm with function approximation. The social outcomes considered were: utility (U), sustainability (S), equality (E) and peace (P), and measured the aggregate effects of individual actions. The authors describe three distinct learning phases: *naïvety*, *tragedy* and *maturity*. Phase 1, *naïvety*, occurs at the beginning of training and is characterized by high commons‘ stocks. Due to agents‘ lack of efficiency at harvesting apples, stocks are never depleted, and U and S are relatively high. P is also high since agents see no advantage in tagging others. Phase 2, *tragedy*, is characterized by the rapid depletion of stocks. Agents in this phase become too good at harvesting apples, and because of that, U and S sharply fall. Finally, phase 3, *maturity*, is characterized by an increase in conflict and a decrease in P. Agents at this point realize tagging can be useful in scenarios of scarcity, and because of that, the other three social outcomes increase.

Starting from the premise that many humans have inequity-averse preferences, Hughes et al. trained agents with these social preferences on the commons and clean-up environments [Hughes et al. 2018]. Both advantageous and disadvantageous inequity aversion were considered, which are, respectively, reductionist models of guilt and envy. This social value is modeled as a reward layer on top of each environment‘s extrinsic payoff structure and as the name suggests, penalizes agents as the difference between their rewards grows further apart. Results show the differences in behavior between advantageous and disadvantageous inequity averse agents. Advantageous inequity aversion drives the difference in behavior from the self, whereas disadvantageous inequity aversion induces mutual cooperation by punishing defection. Furthermore, it was found that advantageous inequity aversion is particularly effective for resolving public goods dilemmas, whereas disadvantageous inequity aversion performs better in the commons dilemma.

Finally, McKee et al. studied the effects of social diversity in mixed-motive games [McKee et al. 2020]. Similar to the work of Hughes et al., they embedded social preferences in agents as an intrinsic reward layer on top of the extrinsic rewards coming from the environment. Preferences were modeled following the Social Value Orientation (SVO) framework from interdependence theory. Simulations were conducted in the commons and clean-up environment. As expected, homogeneous altruistic populations of agents performed well compared to other homogeneous populations in both the commons and clean-up games but scored surprisingly low on the equality metric. Populations with diverse SVO, on the other hand, performed well in both cases while maintaining a higher

equality score.

6.3. Novel architectures

Lerer and Peysakhovich proposed an agent‘s architecture named amTFT that emulates the behavior of TFT in a two-player, extended Markov game setting, the coins environment [Lerer and Peysakhovich 2018]. The amTFT architecture comprises two policies: a purely cooperative and a purely defective. Cooperative policies were defined as those that, starting at an initial state s, maximize the joint expected reward $V^1(s, \pi_1, \pi_2) + V^2(s, \pi_1, \pi_2)$. The cooperative and defective policies were trained by means of self-play in two reward schedules: a purely cooperative and a purely selfish. An amTFT agent works by switching from cooperation to defection for k time steps — k being a function of the agent‘s tolerance towards defection — every time it detects defection by its counterpart. Cooperation/defection is measured at each time step in terms of the partner‘s relative Q-values; the Q-value given the partner‘s action versus the Q-value of the action the partner would have taken if it had cooperated.

Eccles et al. also proposed an agent‘s architecture that implements reciprocity in Markov Games [Eccles et al. 2019]. Three points of improvement were listed compared to Lerer and Peysakhovich‘s work: *a)* reciprocity is not defined only for fully cooperative or fully defective actions, it can be defined in the cooperation-defection continuum; *b)* reciprocity is not only defined for the 2-player case; *c)* reciprocity should not depend on observing the rewards of others; *d)* reciprocity can be learned online. The work accomplishes the proposed improvements by implementing two types of agents: innovators and imitators. An innovator optimizes for purely selfish rewards while an imitator measures and matches the sociality level of others. This diversity creates an interesting dynamic; innovators are incentivized to cooperate since their sociality will be matched by imitators. Sociality, as proposed, is measured in terms of a niceness function that maps the effects of one player‘s actions onto others, similar to the notion of a value function.

7. Discussion

7.1. Differences between models and the issue with cooperation

LLMs and HLMs differ in complexity, the former having a greater grasp of real-world dilemmas than the latter. It is important to state two key differences between them that bring the LLM formalism closer to reality: *a)* HLM is stateless and represented by a simple mapping from actions to rewards, while LLM encompasses state transitions. In practice, it means the rewards in LLM are dependent on the agent‘s sequence of state-action pairs. *b)* There exists a one-to-one mapping between actions and the notion of cooperation. That is to say that agents within HLM opt explicitly to act more or less cooperative, while in LLM cooperation or defection are implicit in one‘s sequence of actions. This has implications to cooperation‘s continuity and synchrony (see 1st. paragraph, items *a* and *b* of Section 6).

An arguably deeper problem emerges when switching from HLMs to LLMs, and it has to do with cooperation changing from being an explicit action to an implicit meta-action. How does one measure cooperativeness? Leibo et al. [Leibo et al. 2017] and Perolat et al. [Pérolat et al. 2017] measured cooperation through a domain-dependent social behavior metric. The problem with this approach is that it is bias-inducing prone

and is only generalizable to problems with a social metric that can change the game's payoff structure as it varies. Furthermore, using a social behavior metric to measure cooperation may lack meaning. For instance, both authors in [Leibo et al. 2017] and [Pérolat et al. 2017] used the frequency of tagging as a proxy for cooperativeness in the commons environment; should an agent that does not tag others but completely depletes the environment be considered truly cooperative?

Conversely, Lerer and Peysakhovich measured cooperativeness based on expected outcome [Lerer and Peysakhovich 2018]. This approach fails to generalize to the n-player case as pinpointing defectors becomes impossible. Furthermore, in stochastic processes — such as many real-world dilemmas —, we can think of scenarios where cooperativeness may yield bad results.

In Eccles et al., the authors use a sociality function that better approximates the meaning of cooperation in the real world [Eccles et al. 2019]. Still, this approach is not generally agreed upon, i.e., a theory of cooperation is yet to be fully developed.

7.2. How to better reciprocate?

As we have seen throughout this paper, reciprocity, embodied as TFT in the early days of Axelrod's tournaments, provides a suitable mechanism for solving social dilemmas. Other attempts at simulating reciprocal behavior, such as the Tit for 2 Tats strategy did not achieve the same level of robustness [Axelrod 1980b]. But the question of how to best reciprocate is still open, and experimenting with Markov games may provide us an answer. It is natural to stick to what has worked in the past, but could strategies such as Tit for 0.9 Tats or Tit for 1.1 Tats perform better in certain situations? These scenarios are open to investigation.

8. Conclusion

The importance of cooperating in mixed-motive settings is highlighted by [Dafoe et al. 2020]. The authors discourse about future challenges we will face as a society, all orbiting around the ill-defined notion of cooperation in mixed-motive environments and which they termed Cooperative AI. To illustrate with an example, how will an autonomous vehicle interpret the fast-walking pedestrian crossing the street looking at his/her phone? Cooperating with this pedestrian might mean something different than cooperating with the honking driver behind who is late for work. As we can see, many of these issues are immersed in social norms, and solving them is far from trivial.

Advancing this newly conceived field is of great concern for many areas of science. In the 80s and 90s, analyzing how artificial agents interacted in these situations could provide insights into acting in real-world societies. As of today, besides guiding us in some of our social interactions, solving these issues could represent a significant gain in productivity as the population of artificial agents acting on behalf of humans grows in size.

9. Acknowledgements

This work was carried out with the support of Itaú Unibanco S.A., through the scholarship program Programa de Bolsas Itaú (PBI), linked to the Centro de Ciência de Dados (C^2D) of Escola Politecnica da USP.

References

Axelrod, R. (1980a). Effective Choice in the Prisoner‘s Dilemma. *Journal of Conflict Resolution*, 24(1):3–25.

Axelrod, R. (1980b). More Effective Choice in the Prisoner‘s Dilemma. *Journal of Conflict Resolution*, 24(3):379–403.

Axelrod, R. (1984). *The Evolution of Cooperation*. Basic, New York.

Barbosa, J. V., Costa, A. H. R., Melo, F. S., Sichman, J. S., and Santos, F. C. (2020). Emergence of Cooperation in N-Person Dilemmas through Actor-Critic Reinforcement Learning. *Adaptive Learning Workshop (ALA), AAMAS 2020 - Autonomous Agents and Multi-Agent Systems*.

Dafoe, A., Hughes, E., Bachrach, Y., Collins, T., McKee, K. R., Leibo, J. Z., Larson, K., and Graepel, T. (2020). Open problems in cooperative ai.

Dawes, R. M. (1980). Social Dilemmas. *Annual Review of Psychology*, 31(1):169–193.

Deadman, P. J. (1999). Modelling individual behaviour and group performance in an intelligent agent-based simulation of the tragedy of the commons. *Journal of Environmental Management*, 56:159–172.

Eccles, T., Hughes, E., Kramár, J., Wheelwright, S., and Leibo, J. Z. (2019). Learning reciprocity in complex sequential social dilemmas.

Gotts, N. M., Polhill, J. G., and Law, A. N. (2003). Agent-based simulation in the study of social dilemmas. *Artificial Intelligence Review*, 19:3–92.

Guerberoff, I., Queiroz, D., and Sichman, J. (2011). Studies on the effect of the expressiveness of two strategy representation languages for the iterated n-player prisoner's dilemma. *Revue d'Intelligence Artificielle*, 25:69–82.

Hardin, G. (1968). The tragedy of the commons. *Science*, 162(3859):1243–1248.

Hughes, E., Leibo, J. Z., Phillips, M., Tuyls, K., Dueñez Guzman, E., García Castañeda, A., Dunning, I., Zhu, T., McKee, K., Koster, R., Roff, H., and Graepel, T. (2018). Inequity aversion improves cooperation in intertemporal social dilemmas. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Izquierdo, S., Izquierdo, L., and Gotts, N. (2008). Reinforcement learning dynamics in social dilemmas. *Journal of Artificial Societies and Social Simulation*, 11.

Kollock, P. (1998). Social dilemmas: The Anatomy of Cooperation. *Annual Review of Sociology*, 24(1):183–214.

Leibo, J. Z., Zambaldi, V., Lanctot, M., Marecki, J., and Graepel, T. (2017). Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, page 464–473, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Lerer, A. and Peysakhovich, A. (2018). Maintaining cooperation in complex social dilemmas using deep reinforcement learning.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, ICML'94, page 157–163, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

McKee, K. R., Gemp, I., McWilliams, B., Duèñez Guzmán, E. A., Hughes, E., and Leibo, J. Z. (2020). Social diversity and social preferences in mixed-motive reinforcement learning. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '20, page 869–877, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Ostrom, E. (1999). Coping with tragedies of the commons. *Annual Review of Political Science*, 2(1):493–535.

Ostrom, E. (2000). Collective action and the evolution of social norms. *Journal of Economic Perspectives*, 14(3):137–158.

Pérolat, J., Leibo, J. Z., Zambaldi, V., Beattie, C., Tuyls, K., and Graepel, T. (2017). A multi-agent reinforcement learning model of common-pool resource appropriation. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Queiroz, D. and Sichman, J. (2016). Parallel simulations of the iterated n-player prisoner's dilemma. In Gaudou, B. and Sichman, J. S., editors, *Multi-Agent Based Simulation XVI*, pages 87–105, Cham. Springer International Publishing.

Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

Racism in Agent Societies: A Model Based on the Concept of Capability-Based Social Control Mechanism

Antônio Carlos da Rocha Costa[1]

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

Abstract. *This paper builds both on S. Haslanger's conceptions of* ideology *and* racism*, and on Carmichael & Hamilton* typology of racism*, to formally introduce a model for* racism *in agent societies.* Racism *is modeled as a system of* racist capability-based social control mechanisms*, each composed of* practices*,* attitudes*,* meanings*, and* material and power conditions *that a variety of social groups adopt and handle, regarding some particular set of social groups, to the effect of* disempowering *the members of the latter groups, on the basis of* racist criteria*, with respect to the possibility of their participation in some part of the organization and functioning of the agent society they inhabit. Two main forms of racism are considered:* overt racism *and* institutional *(or* structural*, or* systemic*)* racism*. A case study formally models the* racist foundation *of the religious system of prototypical* Brazilian colonial plantations*.*
Keywords: *Agent societies. Ideological systems. Social capabilities. Social control mechanisms. Racism. Race-based religious ideologies.*

1. Introduction

This paper builds both on Sally Haslanger's *practice-theoretic* conceptions of *ideology* and *racism* [Haslanger 2017], and on the two basic forms of racism characterized by Stokely Carmichael and Charles V. Hamilton, *overt racism* and *institutional racism*, to formally introduce the concept of *racism* in agent societies.

For that purpose, the paper defines the concept of *capability-based social control mechanism* in agent societies, making use of the concepts of *ideology* and *organizational capability*, introduced in [Costa 2015] and [Costa 2020], respectively. *Racism* is formally defined, then, as a system of *racist capability-based social control mechanisms*.

The paper is structured as follows. Section 2 first reviews the concepts of *ideology* and *organizational capability*, as they apply to *agent societies*. Section 3 introduces *capability-based social control mechanisms* and its two main types of components: *capability check mechanisms* and *capability distribution mechanisms*.

Section 4 summarizes Haslanger's conceptions of *ideology* and *racism*, and extends accordingly our previous concept of *ideology*. Section 5 defines the concept that is the focus of the paper, *racism in agent societies*, in two types: *overt* and *systemic racism*.

Section 6 presents, as a case study, the formal characterization of the *racist foundation* of the religious system present in prototypical *Brazilian colonial plantations*. Section 7 is the Conclusion.

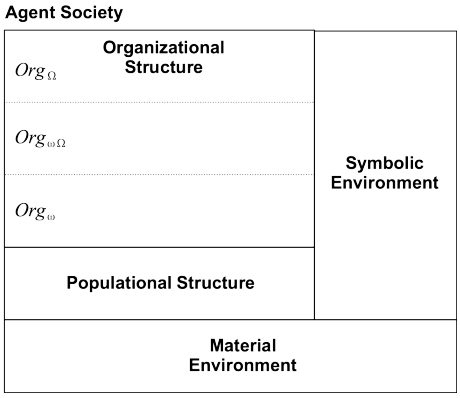


Figure 1. The main components of the *Agent Society* model.

2. Agent Societies

We call *Agent Society* the model of multiagent systems that we have been working on [Costa 2019]. The main components of the model are depicted in Figure 1, where:

- Org is the *organizational structure*, divided in three levels:
  - Org_ω, the micro-organizational level;
  - Org_μ, the mes-organizational level;
  - Org_Ω, the macro-organizational level;
- Pop is the *populational structure*;
- $SEnv$ is the *symbolic environment*;
- $MEnv$ is the *material environment*.

Table 1 lists the *elements* contained in the main components of the *Agent Society* model.

Table 1. The main elements of the architecture of agent societies.

Component		Main Elements
Pop		agents, agent networks
Org	Org_ω	organizational roles
	Org_μ	organization units
	Org_Ω	social subsystems
$MEnv$		material objects
$SEnv$		symbolic objects

where the *organizational units* are structured sets of functionally connected *organizational roles* and the *social subsystems* are structured sets of functionally connected *organizational units*, so that the *micro-organizational level* is composed of a network of *organizational roles*; the *meso-organizational level*, of a network of *organizational units*; and, the *macro-organizational level*, of a network of *social subsystems*.

2.1. Social Groups and Organizations

A particular type of *Organizational Unit*, located in the meso-organizational level Org_μ, are the *social groups*.

We call *social group* a set of *organizational roles*, together with enough *structural and functional specifications* (e.g.: admissible behaviors and interactions, mandatory individual and/or collective goals to be achieved, internal functionalities to be maintained, etc.) that a set of *agents* of the populational structure may implement on the basis of their behaviors and interactions, each agent implementing one or more of those roles.

We distinguish the two main types of organization units, *social groups* and *organizations*, by fact that *organizations* are required to have their set of roles, and structural and functional specifications, formally stated in a *chart* that is independent of the identities and idiosyncrasies of the agents that implement the organization, while *social groups* are not submitted to such requirement, usually being highly dependent on those identities and idiosyncrasies.

Most clearly, as the internal complexities of the *social groups* increase, they may progressively structurally and functionally tend to become *organizations*, by progressively stabilizing their structure and functioning, by making them progressively independent of the identities and idiosyncrasies of the agents that implement them.

Social groups, *organizations*, and *agents* are the main generators, adopters, maintainers, and diffusers, of the *ideologies* that are effective in an agent society at any time.

2.2. Ideologies and Ideological Systems

We call *ideological system* [Costa 2015] the component of the *Symbolic Environment* of an agent society that stores and manages the set of *ideologies* that agents and organization units make use of in the decision processes regarding their social behaviors and interactions.

An *ideology* is a (not necessarily consistent) set of ideological frameworks, and an *ideological framework* is a set of ideological envisagements with which an agent or organization unit may classify and qualitatively identify and assess bodily and/or social or cultural features of other agents or organization units in order to decide on how to behave toward them.

The basic types of *ideological envisagements* introduced in [Costa 2015] are:

- *segmentation envisagements*, which allow for the segmentation of a population of agents and organization units into different social and organizational segments;
- *qualifying envisagements*, which allow for the hierarchization of the different populational and organizational segments according to their supposed competence for the performance of some activity;
- *valuation envisagements*, which allow for the hierarchization of the different activities that are performed in the agent society;
- *normative envisagements*, which allow for the imposition of norms (prohibitions, obligations, etc.) on agents and organization units regarding the different activities performed in the society.

These are envisagements of essentially *organizational* type. Other types of envisagements, like *ethical* or *political* ones, may be added to this set, as required by the applications.

We denote by $IdeoFrmwrks$ the set of all *ideological frameworks* that can possibly be adopted by the agents and organization units of a given agent society.

3. Capability-Based Social Control Mechanisms

In general, the *social control* of a society indicates "*how people define and respond to deviant behavior*" [Black 1990, p. vii].

> More specifically:
>
> "*even in the most complex and differentiated societies of the modern world, legal activity constitutes only a small fraction of the social control in everyday life. All settings—whether families, organizations, occupations, neighborhoods, friendships, or gatherings of strangers—have their own forms of social control. It includes everything from rebukes to homicides, avoidance and exclusion, gossip, negotiation, and various modes of third-party intervention such as mediation, psychotherapy, and adjudication.*" [Black 1990, p. viii]
>
> On the other hand, a *social mechanism* is:
>
> "*a constellation of entities and activities, typically actors and their actions, that are linked to one another in such a way that they regularly bring about the type of phenomenon we seek to explain.*" [Hedström 2005, p. 2].

Thus, we take the *social control* of an agent society to be performed by a system of *social control mechanisms*, each constituted by an articulation of two types of *social mechanisms*, namely, *capability check mechanisms* and *capability distribution mechanisms*, which are articulated in the form that we explain presently.

We call *capability-based social control system* such system of capability-based social control mechanisms.

3.1. Capability Check Mechanisms

We call [Costa 2020] *capacity* of an agent any *behavior* or *interaction* that the agent is capable of performing or participating in, and *capability* any object (material or symbolic) that publicly certifies that a given agent is empowered with some determinate capacity.

Both types of organization units, *social groups* and *organizations*, make use of *organizational capabilities* to discriminate access to their membership process and to the social resources they control. That is, *organizations* and *social groups* often check the *capabilities* that are owned by an agent in order to *accept* it as one of its members or to *give it access* to some resource that they control.

Figure 2 shows the way *capability check mechanisms* operate in agent societies: an organization unit (*Organization unit k*) makes use of a *capability check mechanism* (*CC*) to check if an agent (*Ag*) has, in its capability list (*CL*) the capability that allows it to rightly enter the organization unit to implement one of its roles (capability *Cn* for *Role n*, for instance) in order to use some of the resources controlled by the organization unit (capability *Cj* for *Resource j*, for instance)[1].

Thus, *ideology-driven capability check mechanisms*, when checking the required capabilities according to the set of *ideological frameworks* adopted by the organization units, may privilege some agents over others regarding the access to the organizational roles or resources they control.

[1] See [Costa 2020] for more additional features of *capability-based* social control mechanisms.

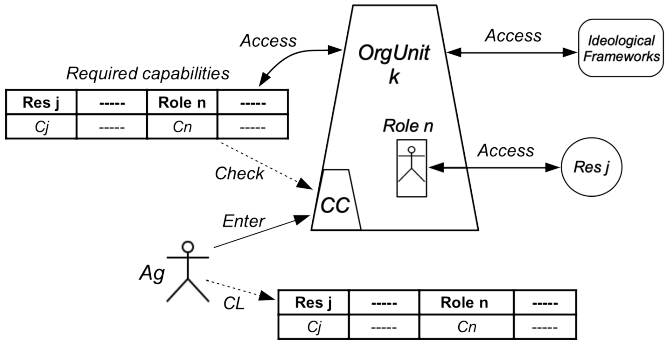


Figure 2. The way *capability check mechanisms* operate in agent societies.

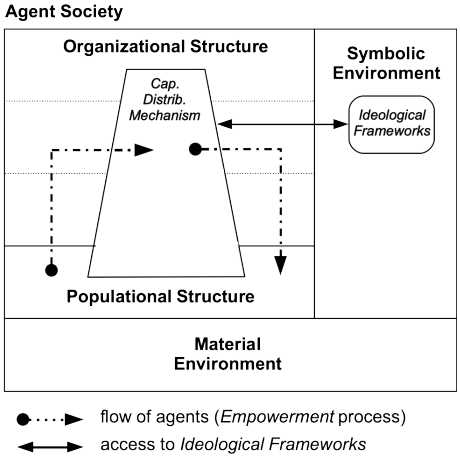


Figure 3. The way *capability distribution mechanisms* operate in agent societies.

3.2. Capability Distribution Mechanisms

Building on [Costa 2018] and the *agent-based version* of Bourdieu & Passeron's model of *class-oriented* schooling system [Bourdieu and Passeron 1990] that it sketched, we call *capability distribution mechanism* any *social control mechanism* that operates in such a way that, allowing a *flow of agents* through itself, assigns some capabilities to some of the agents in the flow while, at the same time, refusing them to others, to the effect of empowering the former and disempowering the latter with respect to the *capability-based access* to some particular type of organizational role or resource.

The *capability distribution mechanisms* are also *ideology-driven* in sense that they operate on the agents that flow through them on the basis of a set of *ideological frameworks* that drive them to privilege some of thosee agents over others, regarding the distribution of capabilities.

Figure 3 shows the general way in which *capability distribution mechanisms* operate in agent societies. Notice that, usually, the control of the very access to the *incoming agent flow* of a capability distribution mechanism is itself capability-based, in a functionally recursive way.

3.3. Systems of Capability Distribution Mechanisms

Clearly, the *recursive functional dependence* between interconnected *capability distribution mechanisms* implies the availability of *primitive capabilities* in the agent societies,

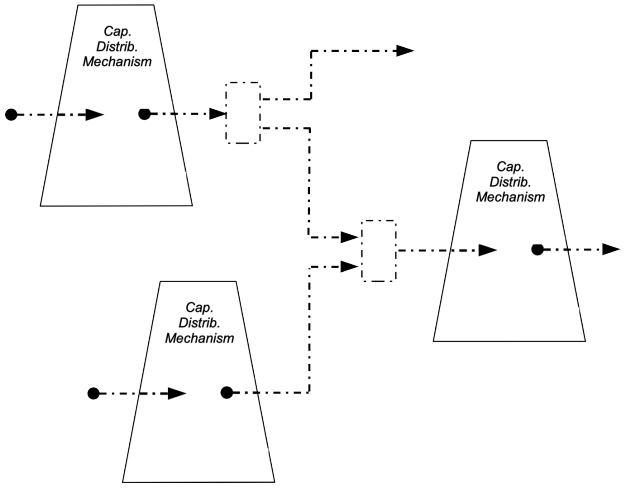


Figure 4. A sample *system of capability distribution mechanisms*.

i.e., features that are *not assigned* to the agents by any capability distribution mechanism but that can be used as *capabilities* [2].

We call *system of capability distribution mechanism* any set of capability distribution mechanisms that are functionally interconnected so that:

- the access to the incoming agent flow of a capability distribution mechanism depends on capabilities that are either *primitive* in the agents or *assigned* to them by other capability distribution mechanisms;
- there are *capability distribution mechanisms* that control their incoming agent flows only on the basis of *primitive capabilities*;
- the latter capability distribution mechanisms operate as *basic capability distribution mechanisms* in the recursive functional structure of the system.

Figure 4 shows a sample *system of capability distribution mechanisms*. The *dash-dotted rectangles* denote unspecified parts of the agent society through which the agents may flow while passing from one capability distribution mechanism to another.

3.4. Capability-Based Social Control Systems

We define the *capability-based social control system* of an agent society as the articulation of the *system of capability distribution mechanisms* of the society, denoted as $CapDistrMechs$, with the *set of capability check mechanisms* of the society, denoted as $CapCheckMechs$, which operate together on the *population* of the agent society in the manner indicated in Figure 5.

4. Haslanger's Conceptions of Ideology and Racism

Sally Haslanger view that "*ideology is part of what gives people their tools of reasoning in the first place*" [Haslanger 2017, p.7] is fully compatible with the functional view of ideology that we have been adopting since [Costa 2015].

Adhering to Haslanger's conception that *racism*, in particular, and *ideology*, in general, *is not* a set of beliefs, but "*is better understood as a set of practices, atti-*

[2] Which is the case, for instance, of *skin color* or *physiognomical traces*, for *overt* race-oriented social control mechanisms, as we consider in the next subsection.

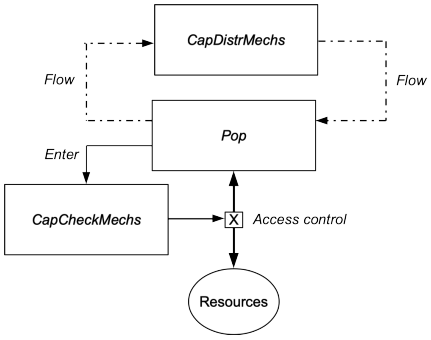


Figure 5. The structure of the *capability-based social control system* of agent societies.

tudes, social meanings, and material conditions, that systematically reinforce one another" [Haslanger 2017, p.1], we add to our previous notion of *ideology* the component of *practices*, which was lacking [3].

We do that by formally construing *practices* as *scripts* [Schank and Abelson 1977], i.e., sets of *behavioral and interactional schemes* that agents and organizational units may have to follow when behaving and interacting with each other.

Thus, we define an *ideology* in an agent society to be a *non-empty* structure of the form [4]:

$$ideology \in \wp(IdeoFrmwrks) \circlearrowleft \wp(Practices) \circlearrowleft \wp(MatConds)$$

where $X \circlearrowleft Y$ is the (cartesian product-like) operation of *mutual reinforcement* between structures X and Y (an operation that we leave formally undefined, for now).

5. Racism in Agent Societies

5.1. The General Concept

As indicated in the *Introduction*, we model *racism* as a *racist capability-based social control mechanism*. That means a capability-based social control mechanism that is driven by *racist ideologies*.

We define a *racist ideology* as an ideology containing at least one *racist ideological framework*, i.e., an ideological framework containing at least one of the following *envisagements* that adopt *race* as a decisive criterion [5]:

- a *qualifying envisagement* that takes *race* as the criterion for qualifying the agents or organization units regarding their competence for performing certain activities;
- a *valuation envisagement* that takes *race* as the criterion for valuating activities that are typical of certain segments of agents or organization units.

Additionally, we define a *capability-based social control mechanism* to be *racist* if it is driven by a *racist ideology*.

[3] In particular, "*practices depend on our ability to coordinate using shared meanings.*" [Haslanger 2017, p.14], which is precisely the coordinating function that we assign to *ideological frameworks*.

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

[5] Notice that we do not consider to be *racist* any *segmenting* or *normative* envisagement that takes *race* as its decisive criterion. Only envisagements that distinguish *qualifications* and *values* on the basis of *race* are considered to be *racist*. This allows for the non-racist character of *affirmative action* initiatives - see, e.g. https://en.wikipedia.org/wiki/Affirmative_action.

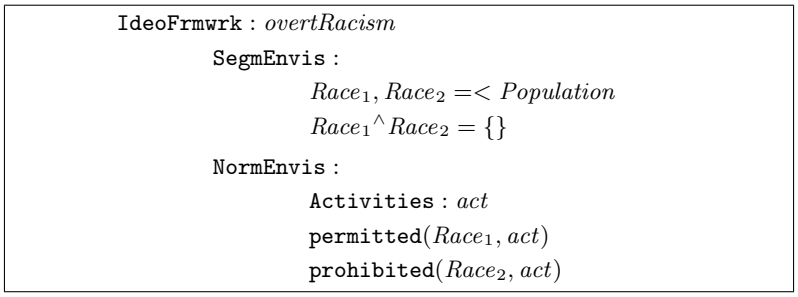


Figure 6. The *TinyIML* ideological framework that regulates the basic form of *overt racism*.

Racism in agent societies is defined, then, as a *system of racist capability-based social control mechanisms*, that is [6]:

$$Racism \subseteq RacCapSocContrMechs \times RacIdeos$$

where:

- $RacCapSocContrMechs$ is the set of *racist capability-based social control mechanisms* that may operate in the agent society;
- $RacIdeos = \wp(RacIdeoFrmwrks) \circlearrowleft \wp(RacPractices) \circlearrowleft \wp(MatConds)$ is the set of *racist ideologies* that may drive the racist social control mechanisms of $RacCapSocContrMechs$, with:
  - $RacIdeoFrmwrks$ is the non-empty set of *racist ideological frameworks*;
  - $RacPractices$ is the non-empty set of *racist scripts* [7];
  - $MatConds$ is a non-empty set of *material conditions*, i.e., a non-empty set of networks of interacting material objects present in the *Material Environment* of the agent society;

In the following, we make use of the *TinyIML* [8], the ideology modeling language introduced in [Costa 2015], to characterize two general forms of racism: *overt racism* and *institutional* (or *structural*, or *systemic*) racism.

5.2. Overt Racism

Overt racism is racism writ large, in open air, essentially by *overtly prohibiting* people of certain races to perform certain behaviors or interactions, like: accessing certain public resources, addressing people of certain other races, etc.

Formally, we characterize the basic form of *overt racism* in TinyIML by means of the ideological framework in Figure 6.

Notice that, in *TinyIML*, *population segments* are *sets*, and the following are *set operations and relations*:

$=<$	the relation of *set inclusion*;
$\wedge$	the operation of set *intersection*;
$\{\}$	the *empty set*.

[6] Compare this *ideology-based* conception of racism with the *polity-oriented* conception of racism by Carmichael and Hamilton:

> *By "racism" we mean the predication of decisions and policies on considerations of race for the purpose of* subordinating *a racial group and maintaining control over that group*. [Ture and Hamilton 1992, p.16].

[7] We leave undefined, here, the concept of *racist script*.

[8] We assume that the reading of *TinyIML* is well intuitive, so that no general account of its syntax and semantics is necessary here.

Remark that *agents* and *organization units* that adopt the ideological framework $overtRacism$ take that:

- the *population* of the agent society is segmented into two segments, $Race_1$ and $Race_2$;
- that there is no agent that belongs simultaneous to the two segments;
- that an activity act is performable in the agent society;
- that agents of $Race_1$ are allowed to perform act, while those of $Race_2$ are not.

Remark also that, from the *functional point of view* of the ideological system of the agent society, the capability *race*, which segments the population and allows some of them to perform the activity act, *is not* a a capability that was ideologically assigned to the agents, but a capability that is innate in them, and so operates as a *primitive capability*.

Finally, remark that the realization of *overt racism* can be directly modeled with the *capability check mechanism* of Figure 2, assuming that it has adopted the ideological framework $overtRacism$ and that:

1. Any agent that attempts to perform the activity act from within the *Organization Unit k* (e.g., playing role *Role n* or accessing resource *Res j*) is checked by *CC* regarding the value of its capability `Race`.
2. If `Race` is valued $Race_1$, the agent is allowed to perform act.
3. If `Race` is valued $Race_2$, the agent is not allowed to perform act.

5.3. Institutional (or Structural, or Systemic) Racism

The term *Institutional racism* was introduced, in 1967, by Stokely Carmichael (aka Kwame Ture) and Charles V. Hamilton [Ture and Hamilton 1992] to denote the general form of *covert racism* embedded in the *institutions* of a given society. We take the terms *structural racism* and *systemic racism* to be equivalent to it [9].

In the original formulation:

> *Racism is both overt and covert. It takes two, closely related forms: individual whites acting against individual blacks, and acts by the total white community against the black community. We call these individual racism and institutional racism. [...] The second type originates in the operation of established and respected forces in the society [...].* [Ture and Hamilton 1992, p.16]

The *connection* between *institutional racism* and *overt racism* was established by the recognition that the *racist attitude*:

> *permeates the society, on both the individual and institutional level, covertly and overtly.* [Ture and Hamilton 1992, p.17]

which we interpret, as indicated in Section 3, as the recognition of the *recursive dependence* between the *ideology-driven* racist social control mechanisms and the *capability-based* ones, the latter serving as *primitive* mechanisms.

In terms of the *TinyIML* ideological constructs, *institutional racism* can be formalized as in Figure 7, where the expression with the form "$X <_{act} Y$" means that, in general, the members of the populational segment X are *less capable* of performing the activity act then the members of the populational segment Y.

[9] See, e.g., https://en.wikipedia.org/wiki/Institutional_racism. See also [Almeida 2019] (in Portuguese).

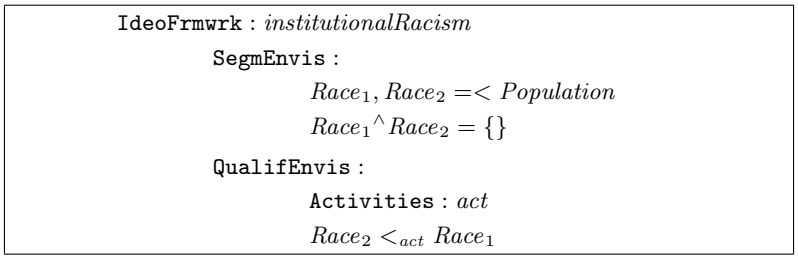


Figure 7. The ideological framework that regulates the basic form of *overt racism*.

The realization of this particular case of *institutional racism* can be directly modeled with the combination (in the way shown in Figure 5) of a *capability distribution mechanism* of the form shown in Figure 3 with a *capability check mechanism* of the form shown in Figure 2, assuming that both have adopted the ideological framework $institutionalRacism$ and that the *capability check mechanism* operates as follows:

1. Any agent that attempts to perform the activity act from within the *Organization Unit k* (e.g., playing role *Role n* or accessing resource *Res j*) is checked by *CC* regarding the value of its capability `Race`.
2. If the value of capability `Race` of the agent is $Race_1$, the agent ag is allowed to perform act.
3. Agents with capability `Race` valued $Race_2$ are allowed to perform act only if there are no agents with `Race` valued $Race_1$ attempting to perform act.

Notice that a traditional and most important type of *institutional racism* is that embedded in the *legal systems* of societies, to the effect that the legal systems accept that individuals of a given race make implicit or explicit use of reasons based on *race differences* as legal defenses, when charged of some crime against an individual of a different race.

6. Case Study: The Racist Foundation of the Religious System of a Prototypical Brazilian Colonial Plantation

In this section, we formally model the racist foundation of the religious system of a prototypical slavery-based Brazilian colonial plantation.

Figure 8 summarizes the main parts of the *ideological framework* that composes the *religious system* of the prototypical slavery-based Brazilian colonial plantation that we have analyzed in [Costa 2016].

The `plantRelig` ideological framework states that:

- the $Population$ of the plantation is composed of $Masters$ and $Slaves$;
- $Masters$ are better qualified to pray for $Saints$ than $Slaves$;
- $Slaves$ are better qualified to pray for $Orishas$ than $Masters$;
- prays directed to $Orishas$ are much less valuable than prays directed to $Saints$;
- $Masters$ and $Slaves$ are allowed to pray for $Saints$ but not for $Orishas$.

The ideological framework $plantRelig$ supports the *institutional racism* that impacts the religious activities in the plantation, to the effect that `Slaves` are not permitted to practice their religion, only that of the `Masters`.

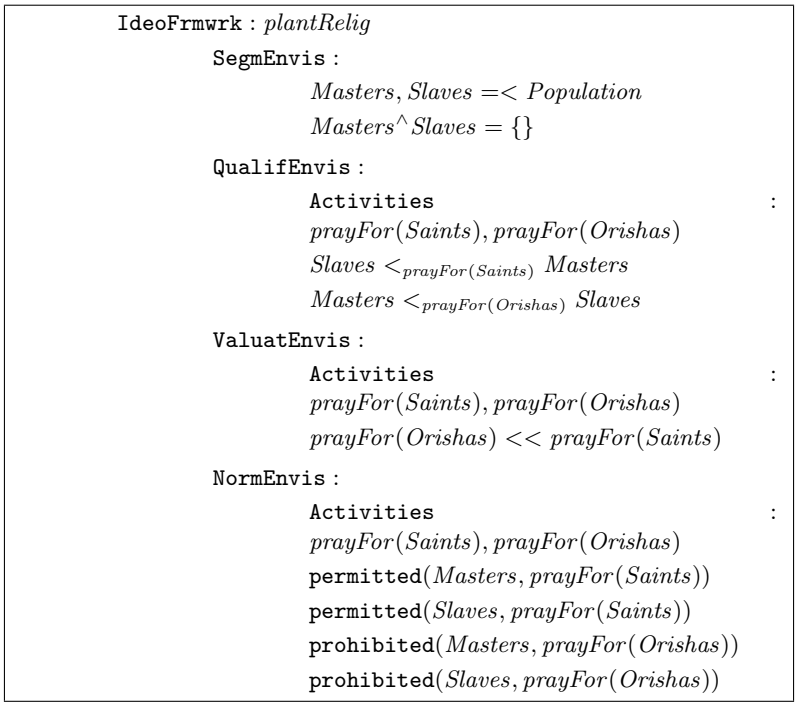


Figure 8. The *religious ideological framework* of the plantation.

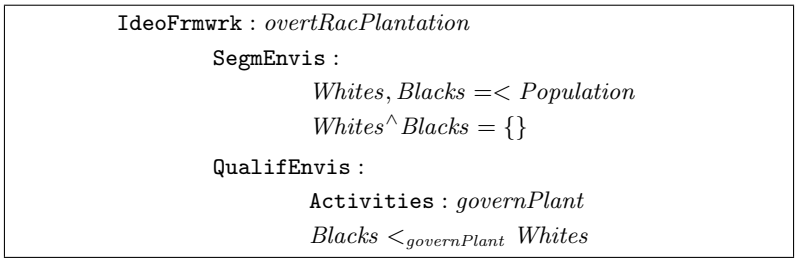


Figure 9. The *overt racism* underlying the *religious ideological framework* of the plantation.

Clearly, $plantRelig$ is consistent and may even be considered sensible by those that accept the proposition stated by the *valuating envisagement*, namely, that:

$$prayFor(Orishas) << prayFor(Saints)$$

which is implied by the assumption that the $Orishas$ of the Afro-Brazilian cults are less valuable than the Catholic $Saints$.

But that proposition may be challenged. Here, we take that the *foundational reason* for the general acceptance of such *institutional* (religion oriented) form of racism in the prototypical Brazilian colonial plantations is the general acceptance in them of the *overt* form of racism given in Figure 9.

The Ideological framework $overtRacPlantation$ states bluntly that $Masters$ are more capable of *governing plantations* than $Slaves$, which implies that it is up to them to determine the activities (including religious activities) that are admissible in plantations.

7. Conclusion

This paper illustrated the viability of the use of the *agent society* model as a *formal semantic model* for social theories and concepts, as proposed in [Costa 2019]. It introduced an ideology-based model for *racism in agent societies*, making use of the previously defined concepts of *ideology* and *organizational capability* in agent societies. The basic ideological issues implied in *racism* were fully exposed through the use of the ideology modeling language *TinyIML*. The concept of *practices* was considered central to that model and *scripts* were proposed as the means to formalize them (which was let as a subject for future work). A case study illustrated the applicability of the proposed ideas.

Acknowledgments

The author is grateful to the anonymous reviewers for their suggestions and comments. This work was partially supported by CAPES.

References

Almeida, S. (2019). *Racismo Estrutural*. Pólen, São Paulo.

Black, D. (1990). Foreword. In Allan V. Horwitz. *The Logic of Social Control*. Springer Science + Business Media, New York.

Bourdieu, P. and Passeron, J.-C. (1990). *Reproduction in Education, Society and Culture*. Sage Publications.

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). Symbolic environments and the cultural aspects of augmented worlds. Slides presented at ECAI-ALAW 2016: 1st International Meeting on Agents Living in Augmented Worlds. Online at: https://apice.unibo.it/xwiki/bin/view/ALAW2016/.

Costa, A. C. R. (2018). Exchange process-based social mechanisms and social functions: an operational approach to the macro functional aspects of agent societies. *Computational and Mathematical Organization Theory*, 24(2):188–223.

Costa, A. C. R. (2019). *A Variational Basis for the Regulation and Structuration Mechanisms of Agent Societies*. Springer, Cham.

Costa, A. C. R. (2020). *Organizational Capabilities in Agent Societies*. Talk given at WESAAC 2020: Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações. Curitiba, UTFPR. Slides available at https://sites.google.com/site/foundationsofagentsocieties/Papers/2020.

Haslanger, S. (2017). Racism, ideology, and social movements. *Res Philosophica*, 94(1):1–22.

Hedström, P. (2005). *Dissecting the Social - On the Principles of Analytical Sociology*. Cambridge University Press.

Schank, R. C. and Abelson, R. P. (1977). *Scripts, Plans, Goals and Understanding: An Inquiry into Human Knowledge Structures*. Lawrence Erlbaum Associates, Hillsdale.

Ture, K. and Hamilton, C. V. (1967/1992). *Black Power: The Politics of Liberation in America*. Vintage, New York.

Implementing a purpose model of status-functions through ontologies to support the social reasoning of agents

Rafhael R. Cunha[1,2], **Jomi F. Hübner**[2], **Maiquel de Brito**[2]

[1]Federal Institute of Education, Science and Technology of Rio Grande do Sul (IFRS)

[2]Federal University of Santa Catarina (UFSC)

rafhael.cunha@posgrad.ufsc.br,jomi.hubner@ufsc.br,maiquel.b@ufsc.br

Abstract. *In multi-agent systems (MAS), the agents may have goals that depend on others and on shared interpretation about the facts that occur in the system. These goals are thus social goals. Artificial institutions provide such a social interpretation by assigning statuses to the concrete elements that compose the system. These statuses enable the assignee element to perform functions that are not exclusively inherent to their design features. However, the enabled functions are not explicit in the existing models of artificial institutions. This limits the agents in reasoning to achieve their social goals in institutional contexts. Considering this problem, this paper proposes a model based on ontologies to express the functions associated with status-functions. We illustrate the model through some examples implemented in the JaCaMo framework, highlighting the benefits that agents acquire when using purposes for reasoning about the satisfaction of their social goals.*

1. Introduction

Consider a scenario where (i) the agent *Bob* has the goal of having a book and (ii) the agent *Tom* has the goal of selling a book. To this end, (i) *Bob* needs to execute an action that means *giving a value in exchange for a good*, and (ii) *Tom* waits for such action to then deliver the book. Both goals are *social goals* because they can not be achieved alone and depend on a *common interpretation* involving certain facts. Without such common interpretation, *Bob* might not know which action to perform to give a value in exchange for the book. Even this would not be the case, *Tom* might not acknowledge the action of *Bob*, refusing thus to deliver the book.

Inspired by human societies, some proposed models and tools provide this kind of interpretation to computer systems and, in particular, to MAS [Fornara 2011]. They usually consider that the elements involved in the interaction among the agents *constitute* (or *count as*) institutional concepts, that are the common interpretation of those concrete elements [Cliffe et al. 2006a, Cardoso and Oliveira 2007, De Brito et al. 2018, Fornara 2011]). These institutional concepts are referred in the literature as *status-functions*: they are *status* that assign *functions* to the concrete elements [Searle 1995, Searle 2010]. For example, the status *buyer* assigns to an agent some functions such as perform payments, take loans, etc. Artificial Institutions are the component of the MAS that is responsible for defining the conditions for an agent to become a *buyer*, or an action to become a *payment* [De Brito et al. 2018].

The existing work on Artificial Institutions is more concerned about (i) specifying and managing the constitution of status-functions and (ii) supporting the regulation of the

system. They focus more on the status than the function. While the status is explicit, *the function is usually implicit*. The current proposals, as far as we know, *do not provide the means for the institution to explicitly express the functions of the status*. The main disadvantage of this limitation on the agents' perspective is that they cannot reason about the functions performed by the elements that carry the status. This limitation has some implications. First, agents may not exploit the functions enabled by the institutions to achieve their social goals. For example, in the *Book store* scenario, *Bob* has the goal of *having the book*. *Bob* can execute an action (e.g., transfer) that is interpreted by an institution such as *payment*. However, if *Bob* does not know the functions associated with *payment*, he does not know that an action that counts as a *payment* leads the system to a state where his social goal is achieved. Second, the agents may not reach their social goal in institutions where different status-functions have the same function. For example, consider a *library* scenario where agents obtain books by constituting status-functions other than *payment*. Consider that *Bob* is coded to achieve his social goal exclusively by performing actions that are institutionally interpreted as *payment*. When *Bob* enters this new scenario (i.e., library), *Bob* is unable to achieve his social goal, because it is unable to exploit the functions associated with status-functions and there is no status-function *payment* in this institution.

Considering the difficulties discussed previously, the main contribution of this paper is the specification through an ontology of a *purpose model* that makes the functions of status-functions explicit, simplifying the agent reasoning towards the achievement of their social goals in the institution. It is inspired by the philosophical theories called "Construction of the Social Reality" by John Searle [Searle 1995, Searle 2010] and "Documentality" by Maurizio Ferraris [Condello 2018, Condello et al. 2019]. Both theories help to understand the concepts of social reality used in this work.

This paper is organized as follows. Sec. 2 introduces the main background concepts necessary to understanding our proposal and its position in the literature. It includes philosophical theories, related work and ontologies. Sec. 3 presents the proposed model and its interfaces. Sec. 4 presents a generic algorithm for agents to find ways to achieve social goals and the specification of the model through an ontology. Sec. 5 illustrate the proposal based on some examples that allow us to identify some limitations and advantages that the model offers on the agent perspective. Finally, Sec. 6 presents some conclusions about this work and suggests future work.

2. Background

This section presents the essential background of this work, which includes philosophical concepts (Sec. 2.1), artificial institutions (Sec. 2.2) and ontologies (Sec 2.3).

2.1. Institutions according to philosophical theories

An institution is composed of institutional facts [Searle 1995, Searle 2010]. These are based on status-functions and constitutive rules. Status-functions are statuses that have associated functions. These statuses enable concrete elements to perform functions (associated with the statuses) that cannot be explained through their physical virtues. Constitutive rules specify the assignment of status-functions to concrete elements with the following formula: X count-as Y in C, where X represents the concrete element, Y the

status-function and C the context where that attribute is valid. For example, a piece of paper (X) count-as money (Y) in a bank (C).

Statuses are imposed on objects when the status-related functions meet some *Purpose*. The functions are called *agentive functions* because they are assigned from *practical interests of the agents* [Searle 1995, p.20]. These practical interests of agents are called *Purposes*. Since the institution is formed by people (i.e., agents) and their collective agreements [Searle 1995], it is possible to say that the agents themselves (through their purposes) assign meaning to the status-functions. In other words, a purpose is the interpretation of a function performed by an element that carries a status from the agents' perspective. For example, someone has the purpose of winning a chess game when it leads the chessboard to a circumstance that count-as a checkmate. This purpose does not occur naturally. It is attributed through the practical interests of the agents playing the game (i.e., under that context). Fundamentally, both agents involved in the game must have the same understanding of these facts (i.e., about the function and their purpose). Otherwise, none of them achieve their social goal. Searle asserts that someone must be *capable of understanding what the thing is for*, or the function could never be assigned [Searle 1995, p.22]. Understanding a function requires to understand for what it serves (i.e., its purpose). In the case of chess game, the purpose of moving the piece to a checkmating position is to win the game. This purpose is in line with the interests of the agents who are playing the game (i.e., its is understood by the people involved in the institution).

While Searle suggests that status-functions are a consequence of collective intentionality, their origin remains, at least in part, unexplained. For example, throughout history, human societies agreed on assigning the function of money to a piece of paper, a shell or a portion of salt. However, the function seems not having a genesis. It is hard to establish when money or any other social objects were invented. It is also even more difficult to explain the nature of the collective intentionality that motivates people to act in different ways when they have contact with a concrete element constituted with a status. Indeed, Searle considers status-functions as a unique abstraction whose function depends on the individual interpretation of each individual, without worrying about how these functions are represented and shared.

To address the mentioned issues, Ferraris [Condello 2018, Condello et al. 2019] proposes to ground the social reality on structures called Documentality. These structures of documents store informations that not only describe or prescribe, but they actually build social objects. Such a structure makes it possible to explain the staying and persistence of functions (and his purposes) associated with status over time. The speech acts that gave origin to functions and status were written and stored in documents that run through time. These documents allow people to learn theses structures through study, perception, etc. For example, the money exercises its functions in the individual intentions only based in the memory (and consequently in the set of functions) that the social objects recover of the individuals with the base on the recording.

To summarize, social objects are used to externalize the set of recordings that allow individuals to remember the functionalities that a *status* (e.g., money) makes available by being assigned to a social object (e.g., paper note). From these theories, it is possible to conclude that an additional system of elements is required so that functions and status can persist and have value recognized over time within the social reality [Condello 2018].

A similar system can be applied to MAS to make explicit the functionalities of the status-functions that compose the institutions. It will permit to improve the agents' reasoning about the satisfaction of their social goals and overcome the difficulties that motivate the realization of this work.

2.2. Institutions in MAS

The main idea of using artificial institutions as a counterpart of human institutions in computer systems has inspired work in MAS. In different ways, these work use the *count-as* relationship, established through the constitutive rules proposed by Searle, to support the regulation of the system [De Brito et al. 2018]. The purpose of this section is to review state of the art on artificial institutions with respect of the explicit representation of the functions associated with the institutional concepts.

Works on Artificial Institutions are usually inspired by the theory of John Searle [Searle 1995, Searle 2010]. Some works present functional approaches, relating brute facts to normative states (e.g., a given action counts as a violation of a norm). These works do not address ontological issues, and, therefore, it becomes even more difficult to support the meaning of abstract concepts present in the institutional reality. Other works have ontological approaches, where brute facts are related to concepts used in the specification of norms (e.g., sending a message counts as a bid in an auction). However, these works have some limitations that are discussed below.

Some approaches allow the agents to reason about the constitutive rules [Cliffe et al. 2006b, Fornara 2011, De Brito et al. 2018, Cardoso and Oliveira 2007, Viganò and Colombetti 2008, Aldewereld et al. 2010]. However, generally the *status-function* (Y) is just a *label* assigned to the concrete element (X) and used in the specification of the regulative norms. Therefore, Y does not seem to have any other purpose than to serve as a basis for the specification of stable regulative norms [Vázquez-Salceda et al. 2008, Aldewereld et al. 2010]. Some exceptions are (i) in [Fornara and Colombetti 2009, Fornara 2011] where Y represents a class formed with some properties as roles responsible for executing actions, time to execute them, condition for execution, etc.; (ii) in [Vázquez-Salceda et al. 2008] where Y is a general concept, and X is a sub-concept that can be used to explain Y. Although the exceptions contain more information than just a label in the Y element, these data are somehow associated with regulative norms. There are no models that make explicit what the constituted elements (i.e., the status-functions) perform in the institution. From the perspective of agents, this can be made explicit through the purposes associated with status-functions. Thus, the status-functions' name would be less relevant to the system's correct functioning. Also, new agents could enter the system and understand the purposes of carrying out some functions that have institutional interpretation, resolving themselves to satisfy their social goals. [Rodriguez-Aguilar et al. 2015] corroborate this conclusion by stating that institutions have not yet considered how to help agents in decision-making, helping them to achieve their own goals.

2.3. Web Ontology Language

Web Ontology Language (OWL) 2 is a practical realization of a Description Logic (DL) system known as SROIQ(D). It allows one to define classes, properties, and individuals. Classes can be viewed as formal descriptions of sets of objects, and individuals can be

viewed as names of objects of the universe [Fornara and Colombetti 2009]. Properties can be either object properties or data properties. The former describe binary relations between objects of the universe; the latter, binary relationships between objects and data values. An OWL ontology consists of: (i) a set of class to describe the modeled domain, which constitute the Terminological Box (TBox); (ii) a set of properties to describe relationships, which constitute a Role Box (RBox); and (iii) a collection of assertions to describe individuals, which constitute an Assertion Box (ABox).

Inspired by the simplified proposal to represent ontologies (available in [Fornara and Colombetti 2009]), the following notation is used to define classes, properties and individuals: $p : C \rightarrow_o D$ to specify an object property named *p*, indicating a relationship between the class *C* and the class *D*. The notation contains the class *C* as the point of origin and the class *D* as the point of destination. It is important to be clear that *p* in this formalism is a property and not a function. We use capital initials for classes, and lower case initials for properties and individuals.

3. The purpose of status-functions

The model proposed in this work is composed of *agents*, *institutions*, and *purposes* (see Figure 1). *Agents* are autonomous entities that pursue their goals in the system [Boissier et al. 2020]. Through MAS definition (cf. Sec. 1), we can see that *goal* is a fundamental concept to understand and program MAS. The literature presents several definitions of *goal* that are different but complementary to each other. In this work, *goals* are something that agents aim to achieve (e.g. the holding of a certain state, the performance of an action, etc.).

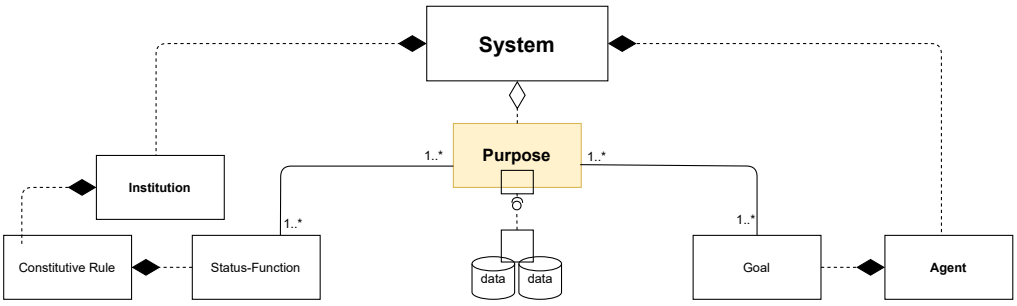


Figure 1. Overview of the model.

Institutions provide the social interpretation of the environmental elements of the system. The several models of artificial institutions consider that constitutive rules specify the assignment of status to those elements, enabling them to perform functions in a certain social context. These statuses with their associated functions are then called status-functions. The assignment of status-functions to the environmental elements is specified through constitutive rules. These rules are generally expressed as *X count-as Y in C* where *X* represents an environmental element (i.e., a brute fact), *Y* represents a status-function to be assigned to X, and *C* represents the context under which the constitution takes place. It is beyond the scope of this paper to propose a model of artificial institution. Rather, it considers this general notion of institution as the entity that constitutes status-functions, that is adopted by several models in the field of MAS.

While *agents* and *institutions* are known concepts, *purposes* are introduced in the proposed model. The *purposes* are an external representation of the practical interests of the agents that can be satisfied by the functions associated with the status-functions (cf. Sec. 2.1). If a goal is something *desirable* by the agent and a purpose of a status-function is a external representation of the practical interests of the agents, then we propose to link them. In this work, we are focusing on *agent's social goals that are satisfied by the purposes of the status-functions*. Therefore, the purpose can be seen as the consequence of the constitution of a status-function (e.g., a state of world) that is aligned with the agent's social goals. For example, the purpose of having a book (i.e., a state of the world reached by the constitution of some status-function) may be associated with agents' social goals such as having a library, having a beautiful bookshelf, having a door weight, etc. In other words, the social goals of the agents can match with the purposes of the status-functions.

In the model, (A) the purpose is connected with the institution through the concept of status-function and (B) the connection between purposes and agents, on the other hand, occurs through the purposes themselves. Shortly, *we are stipulating a relationship between functions of status-functions and purposes and between these purposes and social goals of agents*. Thus, if (i) an agent has a social goal with an assigned purpose, (ii) this purpose is associated with a status-function, and (iii) a constitutive rule specifies how a status-function is constituted, then it is explicit how the agent should act to achieve its social goal (i.e., acting to constitute the status-function that is associated with the same purpose as the social goal). In the previous example, Bob can know that he satisfies his social goal consulting the purpose of *payment* status-function and subsequently consulting the institutional specification to understand what concrete action is constituting the *payment* status-function.

4. A model for purposes of status-functions

This section presents the implementation of the model using ontologies (Sec. 4.1) and an algorithm (Sec. 4.2) to find concrete actions that satisfy social goals.

4.1. Ontology of the model

We specify the proposed model in an OWL 2 ontology. The model represents the *Tbox* and *Rbox* parts discussed in Sec. 2.3. The choice to implement the model using ontologies is justified by the ease of finding and coupling other ontologies available on the web that can represent the *Abox* (domain) part of the application.

The two main classes of the model are *Status-Functions* and *Purposes* (see Figure 2). The first represents the status-functions that may exist in the institution. Each status-function is represented by an individual that pertains to the status-function class. For example, the assertion $pay \in \text{status-function}$ indicates an individual called *pay* that belongs to the *Status-Function* class. The second class (Purpose) generalizes the interests of the agents that are represented through subclasses of that class. For example, *haveBook* is a specialization of class *have*. It can be represented by a axiom $HaveBook \subset Have$. To represent a specific purpose, the developer can create an individual belonging to a class. For example, the assertion $haveBook1 \in HaveBook$ indicates that the individual called *haveBook1* belongs to the *HaveBook* class. When new purposes will exist in the

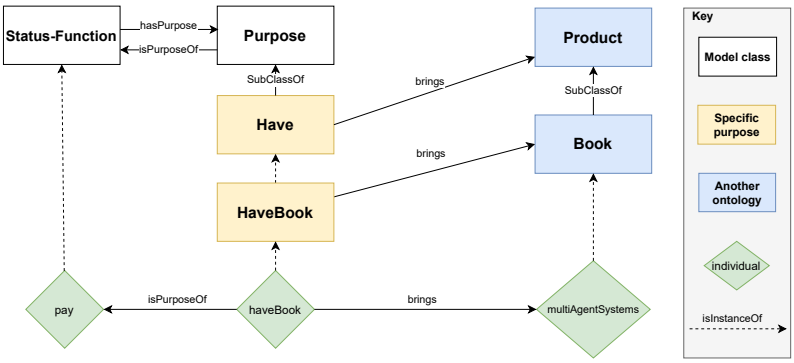


Figure 2. Ontology implementing the model.

system, it is enough to create new individuals to represent them. In this way, the taxonomy of classes and relations are reused. If necessary, the institution's ontology can be related to the ontology of the application domain.

There is a relationship between the classes *Purpose* and *Status-Function* in the model. This relation supports the step 2 of the algorithm 1. The relation is represented through the *hasPurpose* property. The property axiom used to represent this relationship is: $hasPurpose : Status - Function \rightarrow_o Purpose$. These relations have an individual (i.e., a domain) that belongs to the *Status-Function* class and an individual (i.e., a range) that belongs to the *Purpose* class. This relationship means that through the constitution of the status-function, it is possible to change the institution where the purpose is satisfied. We can define a relationship between two individuals belonging to these two classes through assertion: $hasPurpose : pay \rightarrow_o haveBook1$. The example of this relationship means that through the *pay* status-function, the *havebook1* purpose is satisfied.

SPARQL query:

```
PREFIX ont: <http://www.semanticweb.org/rafhaelrc/ontologies/2020/8/semantic_sf_4#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?sf ?book

WHERE
{
 ont:haveBook ont:isPurposeOf ?sf .
  ont:haveBook ont:brings ?book
}
```

sf	book
pay	multiAgentSystems

Figure 3. SPARQL query.

The relation *isPurposeOf* is the inverse of the property axiom *hasPurpose*. This relationship is necessary when someone wants to find out the status-functions associated with a particular purpose (i.e., someone doesn't know the status-function name, but someone already knows the purposes). Through this relationship, considering the purpose of *haveBook1*, the *pay* status-function can be obtained. From these definitions, we can specify purposes and relate them to status-functions. This represents the social knowledge that is present in an environment where agents can interact.

From the specification of the model through an ontology and through a language for querying ontologies (e.g., SPARQL), we can obtain answers regarding step 2 of the algorithm (see Figure 3). This query seeks (i) which status-function is associated with the *haveBook* purpose through the *isPurposeOf* relationship and (ii) which concrete element the *haveBook* purpose is associated with through the *brings* property. The results obtained are *pay* for the first query and *multiAgentSystems* for the second query. The queries carried

out in SPARQL allow us to observe that the ontology can represent the model's concepts.

4.2. Algorithm to reach agent's social goal

The algorithm 1 can be summarized in three steps: (1) Find a purpose associated with the agent's social goal (line 4); (2) find a status-function associated with that purpose (line 5); and finally (3) find a concrete action that can constitute this status-function (lines 6-7). It is assumed that the actions performed by the agents are taken as events by the institution.

Algorithm 1 Find a concrete action that satisfies a social objective

```
1:  Input: agent's social goal
2:  Output: concrete action'
3:  L ← {}                                          ▷ starts the empty set.
4:  if there is a purpose associated with agent's goal then
5:      for each status-function σ associated with that purpose do
6:          if there is a concrete action α that count-as σ then
7:              L ← L ∪ (σ, α)                      ▷ add the pair in set L.
8:          end if
9:      end for
10: end if
11: return L
```

Regarding the first step of the algorithm, the specification of the relationship between these two concepts is outside of the scope of this work. The step 2 requires a representation of the social knowledge of the context in which the agent is inserted (cf. Sec. 4.1). Finally, in the step 3 of the algorithm, we assume that there are works that implement the institutional reality and maintain a constitutive specification that the institution must explain [Cliffe et al. 2006b, Fornara 2011, De Brito et al. 2018]. This specification contains rules that allow the agent to understand what concrete actions (i.e., events in the system) it can use to constitute status-function.

5. Using the model in a Multi-Agent System

To exploit the proposed ontology and the algorithm 1, consider an open Multi-Agent System composed of agents, environment and institution. The agents Bob, Alice, François, and João aim to have a book. Different programmers have developed the different agents' specifications, that are slightly different from each other. The goal of agent *Bob* is identified by the term *have a book*. Alice has the goal of *get a book*. François has the goal of *obtenir a book* and João has the goal of *ter a book*. The intended outcome of all these goals is the same. We consider an environment where all agents are located in a book store (Figure 4). This system is instrumented with an institution that contains a constitutive rule stating that the concrete action *transfer count-as pay*. We focus only on this constitutive rule to illustrate the main features of the model proposed (cf. Sec. 3).

The example is implemented with the JaCaMo framework[1] [Boissier et al. 2020] (see Figure 5). The agents (*Bob*, *Alice*, *François* and *João* in Figure 4) are coded in Jason and the environment in CArtAgO. To implement the artificial institution (*Institutional*

[1]https://github.com/rafhaelrc/psf_model

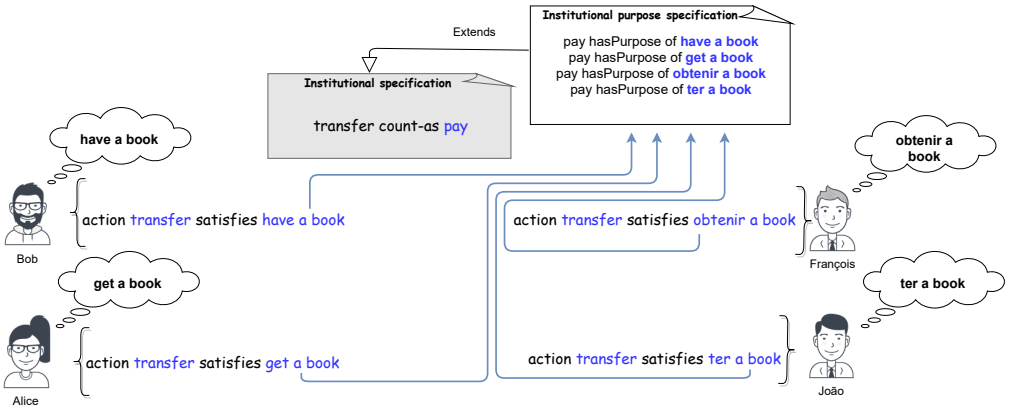


Figure 4. Use of the proposed model in an institutional specification.

specification in the Figure 4), we used an implementation of the Situated Artificial Institution (SAI) model [De Brito et al. 2018]. In SAI, the institutional reality is composed of *status-functions* attributed to concrete elements through the interpretation of *constitutive rules*. To implement the model proposed in this work (*Institutional purpose specification* in the Figure 4), we use ontologies (cf. Sec. 4.1). Finally, to make the model accessible to agents, we encapsulated it in a CartAgO artifact. The query and persistence of data in the ontology was possible because we used MasOntology[2], a set of tools developed in CArtAgO to interact with ontologies.

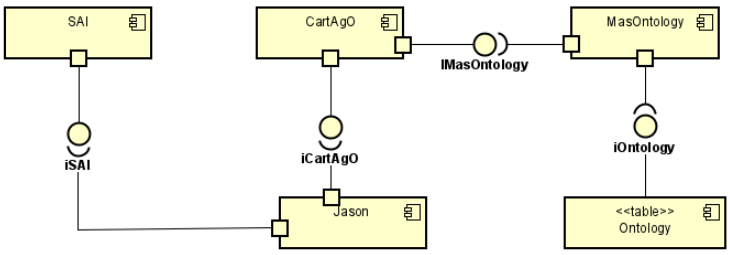


Figure 5. Component diagram with the systems used to compose the example.

Considering the focus of the article, which are (i) the specification through an ontology of a purpose model that explains the functions of status-functions (already specified in Sec. 4.1) and (ii) the relationship of these functions to the social goals of agents, the rest of this section shows how agents can benefit from the proposed model. The other components used in the system (SAI, CartAgO, etc.) are not detailed due to space.

Figure 6 depicts the agents program. The program of all agents are similar, varying only the term they use to refer to their social goals. The agents' social goals can be satisfied by the plans illustrated in lines 4 of each piece of code in Figure 6. In these plans, agents acquire a new goal called *commonPurpose*, that includes all the required actions to find the action that satisfies the purposed related to such social goal in the institution. These goals can be met through the plan detailed in Figure 6 (E). In this excerpt, first the agent consults the ontology (which is a representation of the model) using the *isPurposeOfStatusFunction* operation encoded in a CartAgO artifact (line 10). This

[2]https://github.com/smart-pucrs/MasOntology

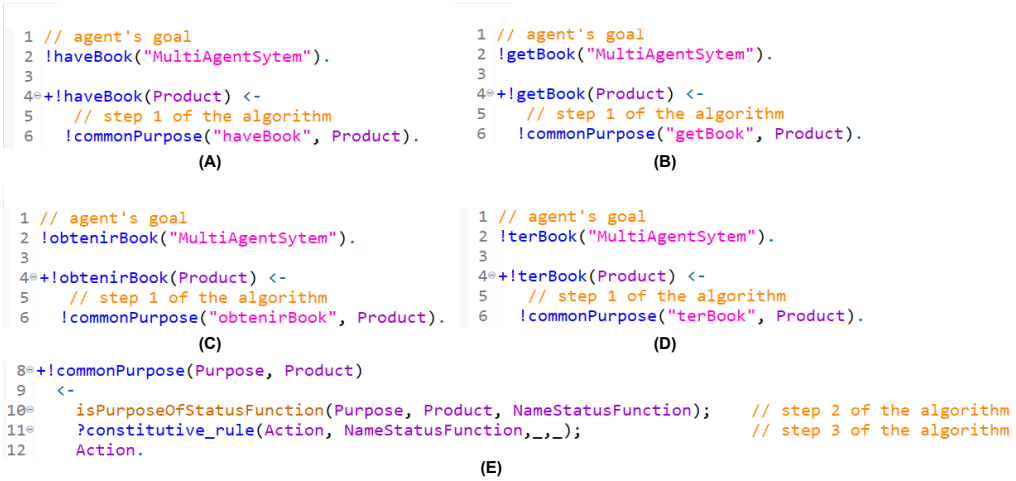


Figure 6. Plans of the agents Bob (A), Alice (B), François (C), João (D) to satisfy a social goal. These agents include the plan commonPurpose (E).

operation has as its first parameter the term used by the agent to refer to its social goal and the second parameter (if any) is the concrete elements that are related to the purpose (for example, the name of a book). When performing this operation, the agent obtains in the third parameter the status-function that satisfies the purpose. Second, the agent consults which concrete action may constitute the found status-function (line 11). Finally, the agent executes the action, eventually satisfying the social goal (line 12). Table 1 briefly shows the execution of agent *Bob* in the example. The other agents are executed similarly, varying only the time 0 and 1 according to their social goals. From these steps, all agents are able to achieve their different social goals in the same institution.

Table 1. Simulation of the execution of the example.

Time	Agents' actions
T=0	Goal: *!haveBook("MultiAgentSystem")*; Plan: *+!haveBook(Product)*.
T=1	Bob executes the *+!haveBook(Product)* plan and acquires the *commonPurpose("haveBook", Product)* goal.
T=2	Bob starts the execution of the *+!commonPurpose(Purpose, Product)* plan.
T=3	Bob executes *isPurposeOfStatusFunction* method (line 10). The result of this execution is the name of the status-function associated with its purpose (e.g., *pay*).
T=4	Bob queries which concrete action might constitute the status-function (line 11). The result is the name of the action (e.g., *transfer*).
T=5	Bob executes the *transfer* action (line 12), Tom receives the value, he achieves his social goal and hands the book to Bob.
T=6	Bob receives the book and achieved his social goal.

6. Results and future works

The problem motivating this paper is the difficulty of the agents to reason about the functions associated with status-functions representing the institutional interpretation of cer-

tain facts that occur in the environment. This problem is partially solved by computational models that implement artificial institutions. However, these models do not represent the purpose of this interpretation from an agent perspective. Agents have to be hard-coded to know which status-functions can be useful for them to achieve their social goals. Considering this problem, we propose the specification of a purpose model using an ontology to express the purposes of status-functions and allow agents to reason about them.

There are some advantages of such conception that we discuss below. The first one is related to the implementation of the model using OWL. Engineers can use the ontology to create representations of purposes based on the core of the proposed model. The ontology can then be extended with domain and application-specific modules. According to [Fornara 2011], Semantic Web technologies are increasingly becoming a standard for internet applications, and thus allow for a high degree of interoperability of data and applications, which is a crucial precondition for the development of open systems. The second is related to the *flexibility* for the agents, that can achieve their social goals even though it is specified with different nomenclatures. For example, the Sec. 5 illustrates a scenario where all agents are located in the same institution (e.g., book store). In this example, specified agents with vocabularies other than the institutional vocabulary can be specified considering the purposes related to the status-functions rather than the status nomenclature. The advantages are (a) the agents can reason about these purposes and adapt their behavior for different scenarios to satisfy their social goals and (b) by reasoning about the purposes of the status-functions, the agent can realise that these purposes are similar to their interests and therefore they can help the agents to reach their social goals. The agent's capability to reason about the purposes and adapt to different scenarios is an important advance in open systems' flexibility [Aldewereld et al. 2010]. The third is related to the *autonomy* of the agents. In this case, the agent can reason about the actions in the plans and the regulative rules that govern the system. In both cases, the agent has greater autonomy and flexibility in deciding whether a particular action will help him reach his social goal. There are also some advantages from an institutional perspective (see [Cunha et al. 2021] for more details).

As future work, we plan to explore additional theoretical aspects related to the model, such as (i) investigations about how other proposed institutional abstractions fit on the model, (ii) the verification of the consistency among status-functions' purposes and agents' social goals, and (iii) study the relationship between purposes and social functions in addition to artificial institutions as defined in other works (e.g., [da Rocha Costa and Dimuro 2012]). We plan to also address more practical points such as (i) the integration of this model in other models that implement artificial institutions, (ii) the implementation of a library that generalizes the use of the model and (iii) the use of the model in real scenarios.

7. Acknowledgements

The authors would like to thank the Federal Institute of Education, Science and Technology of Rio Grande do Sul (IFRS) for supporting this research.

References

Aldewereld, H., Álvarez-Napagao, S., Dignum, F., and Vázquez-Salceda, J. (2010). Making norms concrete. In *Proceedings of the 9th International Conference on Au-*

tonomous Agents and Multiagent Systems: volume 1-Volume 1, pages 807–814. International Foundation for Autonomous Agents and Multiagent Systems.

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

Cardoso, H. L. and Oliveira, E. (2007). Institutional Reality and Norms: Specifying and Monitoring Agent Organizations. *International Journal of Cooperative Information Systems*, 16(01):67–95.

Cliffe, O., De Vos, M., and Padget, J. (2006a). Answer set programming for representing and reasoning about virtual institutions. In *International Workshop on Computational Logic in Multi-Agent Systems*, pages 60–79. Springer.

Cliffe, O., De Vos, M., and Padget, J. (2006b). Specifying and reasoning about multiple institutions. In *International Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems*, pages 67–85. Springer.

Condello, A. (2018). Two questions on the ontology of money. *Ardeth*, (03):181–191.

Condello, A., Ferraris, M., and Searle, J. R. (2019). *Money, Social Ontology and Law*. Routledge.

Cunha, R. R., Hübner, J. F., and de Brito, M. (2021). Coupling purposes with status-functions in artificial institutions. *arXiv preprint arXiv:2105.00090*.

da Rocha Costa, A. C. and Dimuro, G. P. (2012). Elementary social functions: Concept and interrelation to social dependence relations. In *2012 Third Brazilian Workshop on Social Simulation*, pages 23–30. IEEE.

De Brito, M., Hübner, J. F., and Boissier, O. (2018). Situated artificial institutions: stability, consistency, and flexibility in the regulation of agent societies. *Autonomous Agents and Multi-Agent Systems*, 32(2):219–251.

Fornara, N. (2011). Specifying and monitoring obligations in open multiagent systems using semantic web technology. In *Semantic agent systems*, pages 25–45. Springer.

Fornara, N. and Colombetti, M. (2009). Ontology and time evolution of obligations and prohibitions using semantic web technology. In *International Workshop on Declarative Agent Languages and Technologies*, pages 101–118. Springer.

Rodriguez-Aguilar, J. A., Sierra, C., Arcos, J. L., Lopez-Sanchez, M., and Rodriguez, I. (2015). Towards next generation coordination infrastructures. *Knowledge Engineering Review*, 30(4):435–453.

Searle, J. (2010). *Making the social world: The structure of human civilization*. Oxford University Press.

Searle, J. R. (1995). *The construction of social reality*. Simon and Schuster.

Vázquez-Salceda, J., Aldewereld, H., Grossi, D., and Dignum, F. (2008). From human regulations to regulated software agents' behavior. *Artificial Intelligence and Law*, 16(1):73–87.

Viganò, F. and Colombetti, M. (2008). Model Checking Norms and Sanctions in Institutions. (ii):316–329.

The Impact of Norms Generality on MAS Goal

Jhonatan Alves , Jomi Fred Hübner , Jerusa Marchi

[1]Federal University of Santa Catarina (UFSC)
Florianópolis – Brazil

jhonatan.alves@posgrad.ufsc.br,{jomi.hubner,jerusa.marchi}@ufsc.br

Abstract. *In this paper we aim to investigate the impact of levels of generality on norms synthesized on the global goal satisfiability and conflicts avoidance of Normative Multiagent Systems. We found out that exists a level of generality in the scenarios we perform some experiments, which we call* apex level*, where norms tend to be efficient in avoiding conflicts while ensuring goal system reachability.*

1. Introduction

Norms are a topic of interest in many research areas. In Multiagent systems (MAS) they are generally understood as rules which, in certain contexts, impose restrictions on agent's behavior indicating which actions are allowed, prohibited or obligatory to achieve social order [Boella and van der Torre 2004]. Among the topics investigated on the norms subject in the MAS' field, we have the *synthesis* which refers to the process of creating a set of norms to regulate the agents. Basically, norms can be synthesized in two different ways: *manually* - where a designer produces the set of norms; or *automatically* - where a computational mechanism processes such synthesis [Frantz and Pigozz 2018].

Although the automated synthesis may provide certain advantages as time saving and standardization of results, it is a complex problem [Shoham and Tennenholtz 1995] and some important issues must be considered in this process to the resulting norms regulate the agents efficiently balancing control and autonomy. One of those problems is the synthesis of *mildly restrictive norms*, which barely regulate the agents giving them much freedom to act as they wish and, consequently, allowing conflicts during the system execution. Another problem is the synthesis of *strict norms* which, in turn, may excessively regulate the agents giving them low freedom to perform their actions preventing them from executing the necessary actions for the system fulfills its goal.

Given two norms n_1 and n_2, we say that norm n_1 is more generic than norm n_2 if it regulates the behaviors that norm n_2 does, however the opposite does not occur. For example, a norm which prohibits agents to move between two adjacent vertices in a grid is more generic than a norm which prohibits only the agent *B*ob to perform such action. The norm n_1 prohibits all agents to move, thus the behaviors regulated by norm n_2 are regulated by norm n_1 as well. Such norm is a *strict* one and turns the system goal unreachable since the agents get stuck in their depart vertices. On the other hand, norm n_2 is less restraining, since just Bob cannot move, thus the set of behaviors regulated by norm n_2 does not include those regulated by norm n_1. Norm n_2 is *mildly restrictive* and it is not capable to avoid conflicts since when the other agents move collisions may occur.

In this sense, exploring the *generality of norms* is a fundamental key to obtain a set of norms which constrains the agents avoiding conflicts and making the system

goal feasible to be reached. Although different mechanisms for the automated norm synthesis have been proposed [Onn and Tennenholtz 1997, Boella and van der Torre 2007, Savarimuthu et al. 2008, Christelis and Rovatsos 2009, Morales et al. 2011], just a few of them [Fitoussi and Tennenholtz 1998, Morales et al. 2013, Morales et al. 2014] consider such generality as part of the synthesis scope to measure how generic norms must be to obtain norms that are not mildly restrictive nor too restraining.

In this paper our aim is to investigate *how the generality of norms impacts on the system's goals reachability*. For this, we propose a model in which the norms are organized in levels of generality so that it is possible to correlate the generalities with the capability to avoid conflicts and keep the objectives of the system attainable. According to our experiments, there exists at least one intermediate level of generality, which we call *apex level*, where some norms are capable to maximize the goal reachability and minimize the conflicts' occurrence. We intend to use this information to posteriorly build a synthesis algorithm which use them as a heuristic to guide the process to obtain norms whose level of generality corresponds to the *apex level*.

The remainder of this paper is structured as follows. In Section 2 we introduce our model for norms generalization. In section 3 we present two algorithms to norm synthesis and norm evaluation. In section 4 we look for an answer to our research question, by presenting a set of tests which explore the generality of norms in order to figure out the existence of a level of generality where norms may regulate the agents efficiently. In Section 5 related works are presented and discussed. We also point out a possible and hypothetical direction in which we may overcome them. Finally, in Section 6 we present our conclusion and future works.

2. A Model for Norms Generalization

Before introducing our model consider the following toy scenario which will be used along the text to exemplify some of its concepts. The scenario is composed by a simple grid domain $m \times n$ where intersections represent vertices and edges represent paths which connect such vertices. There are two agents on the grid, $\mathtt{ag}_1$ and $\mathtt{ag}_2$ which must depart from an initial vertex to a goal one executing their traversal plans without getting in conflicts. A conflict is characterized as a collision occurring when an agent moves to a local occupied by another agent. Moreover, agents may just move between vertices which are adjacent.

Definition 1 (**Language**). *Let $\mathcal{L}$ be a restrict form of a first order language, defined as a 5-uple* $\langle \text{Pred}_{\mathcal{L}}, \text{Var}_{\mathcal{L}}, \text{Const}_{\mathcal{L}}, \text{Types}_{\mathcal{L}}, \text{Conec}_{\mathcal{L}} \rangle$, *where* $\text{Pred}_{\mathcal{L}} = \{pred_1, \ldots, pred_m\}$ *is a set of predicate symbols* $\cup \{False\}$, $\text{Var}_{\mathcal{L}} = \{v_1, \ldots, v_n\}$ *is a set of typed variable symbols,* $\text{Const}_{\mathcal{L}} = \{c_1, \ldots, c_k\}$ *is a set of typed constant symbols,* $\text{Types}_{\mathcal{L}} = \{t_1, \ldots, t_p\}$ *is a set of types and* $\text{Conec}_{\mathcal{L}} = \{\wedge\}$ *is a set of logical connectives.*

Each symbol of predicate is associated with a finite set of terms. A term is a variable or a constant symbol associated with a given type of $\text{Types}_{\mathcal{L}}$, such that, they form the set $\text{Terms} = \text{Var}_{\mathcal{L}} \cup \text{Const}_{\mathcal{L}}$. In special, there is one predicate with zero terms: *False* meaning false. The well-formed formulas are given by the formation rules of the first-order logic considering the connectives in $\text{Conec}_{\mathcal{L}}$. It is established that variables begin with capital letters while constants begin with lowercase ones and terms described with distinct symbols represent distinct objects.

Example 1. *Consider the grid domain previously introduced, the formula* $\texttt{next}(\texttt{L}_1,\texttt{L}_2)$ *represents that an arbitrary vertex* $\texttt{L}_1$ *is adjacent to another arbitrary vertex* $\texttt{L}_2$ *while the formula* $\texttt{at}(\texttt{ag}_1,\texttt{L}_1)$ *represents that the specific agent* $\texttt{ag}_1$ *is at the vertex* $\texttt{L}_1$.

Definition 2 (System State). *Let* $\mathcal{P} = \{p_1,\ldots,p_m\}$ *be a set of all ground atomic formulas with* $\mathcal{P} \subset \mathcal{L}$ *and* $\mathcal{S} = \{s_1,\ldots,s_n\}$ *be a set of system states, where* $\mathcal{S} \subseteq 2^{\mathcal{P}}$. *A state* $s_i = \{p_1,\ldots,p_k\}$ *is a set of ground atomic formulas which holds in a given time i.*

The states are built according to the hypothesis of closed world (predicates that do not make part of a given state are considered false). We may say that the special predicated False is prohibited to take part of any system state. As we will see, it is used as part of norms definition and has impact on the norm generality.

Definition 3 (Action). *Let* $\mathcal{A} = \{a_1,\ldots,a_m\}$ *be a set of unground actions. An action* $a_i = \langle \mathrm{Id}, \mathrm{Par}, \mathrm{Pre}, \mathrm{Add}, \mathrm{Del}\rangle$ *is a 5-uple, such that,* Id *is a identifier,* Par $= \{par_1,\ldots,par_n\}$ *is a set of parameters with* Par $\subset$ Var, Pre $= \{p_1,\ldots,p_k\}$ *is a set of preconditions which must hold to* a_i *be executed,* Add $= \{p_1,\ldots,p_j\}$ *is a set of effects which start to hold after the action execution and* Del $= \{p_1,\ldots,p_l\}$ *is a set of effects which does not hold anymore after the action execution, where* Pre, Add, Del $\subset \mathcal{L}$ *are unground atomic formulas. Let* $\mathcal{A}' = \{a_1,\ldots,a_z\}$ *be a set of ground actions, such that, an action* a'_i *is a copy of* a_i *where its parameters are constants of* $\mathrm{Const}_{\mathcal{L}}$ *and its sets are formed by ground atomic formulas.*

From the set $\mathcal{A}$ partial and grounded actions may be obtained which, in turn, are important to synthesise norms with different levels of generality (see algorithm 1 in section 3). However, to act in a environment the agents perform grounded actions of $\mathcal{A}'$.

Example 2. *To* move *between vertices, the agents may perform a grounded version of action* $\langle \texttt{move}, \{\texttt{Ag}_1,\texttt{L}_1,\texttt{L}_2\}, \{\texttt{at}(\texttt{Ag}_1,\texttt{L}_1), \texttt{next}(\texttt{L}_1,\texttt{L}_2)\}, \{\texttt{at}(\texttt{Ag}_1,\texttt{L}_2)\}, \{\texttt{at}(\texttt{Ag}_1,\texttt{L}_1)\}\rangle$.

According to the action move, to an agent $\texttt{Ag}_1$ move from a vertex $\texttt{L}_1$ to a vertex $\texttt{L}_2$, such agent must be at $\texttt{L}_1$ which, in turn, must be adjacent to $\texttt{L}_2$. After executing move, the agent will be at vertex $\texttt{L}_2$ and the agent won't be longer at the former vertex. We will use $Par(a)$, $Pre(a)$, $Add(a)$ and $Del(a)$ along the text to refer that such sets are components of a given action $a \in \mathcal{A} \cup \mathcal{A}'$. However, in the examples we will use the a's signature[1] to refer to it.

Definition 4 (MAS Domain). *Let* $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}', \gamma\rangle$ *be a domain of a MAS, such that,* $\mathcal{S}$ *denotes a set of system states,* $\mathcal{A}'$ *denotes a set of (grounded) agent's actions and* $\gamma : \mathcal{S} \times \mathcal{A}' \to \mathcal{S}$ *denotes a deterministic and discrete transition function between system states, such that,* $\gamma(s,a) = (s - Del(a)) \cup Add(a)$.

Other important concept of our model is conflict. The notion of conflict appears in a variety of areas, and it is defined by reflecting the own characteristics of each one. Conceptually, in this paper we consider a conflict as an event which may cause negative impacts on the system as avoiding agents to accomplish their tasks and compromising the system goal reachability. Formally, we define a conflict in terms of its effects.

Definition 5 (Conflict). *Let* $\mathcal{C} = \{c_1,\ldots,c_m\}$ *be a set of conflicts which may occur during the system execution, a conflict* c_i *is defined as a set of atomic formulas which denotes its effects, such that,* $c_i = \{p_1,\ldots,p_n\}$, *where* $c_i \subset \mathcal{L}$.

[1] A signature is a concatenation between the a action's identifier, an open parentheses, the action's parameters and a close parentheses with no curly braces as $\texttt{move}(\texttt{Ag}_1,\texttt{L}_1,\texttt{L}_2)$, for example.

Example 3. *A conflict which establishes a collision between agents on the grid may be described as* $c_1 = \{\texttt{at}(\texttt{Ag}_1,\texttt{L}_1),\texttt{at}(\texttt{Ag}_2,\texttt{L}_1)\}$. *Thus, according to* c_1*, a collision occurs when two distinct agents are on the same vertex* $\texttt{L}_1$.

Definition 6 (Conflict Instances). *Given a conflict* $c_i \in \mathcal{C}$, $I(c_i) = \{c_i^1, \ldots, c_i^n\}$ *is a set of instances of* c_i*, where an instance* c_i^j *is a copy of* c_i *with ground atomic formulas.*

Example 4. *An instance of conflict* c_1*, from example 3, could be* $c_1^j = \{\texttt{at}(\texttt{ag}_1,\texttt{a}),\texttt{at}(\texttt{ag}_2,\texttt{a})\}$ *denoting that both agents* $\texttt{ag}_1$ *and* $\texttt{ag}_2$ *are on the vertex* a.

Definition 7 (Conflict State). *A given state* $s \in \mathcal{S}$ *is said to be a conflict state if* $\exists c_i \in \mathcal{C}$: $\exists c_i^j \in I(c_i)$*, such that,* $c_i^j \subset s$.

A way to avoid conflict states and preserve the system goal reachability is regulating the agents' behaviors through norms.

Definition 8 (Norm). *Let* $\mathcal{N} = \{n_1,\ldots,n_m\}$ *be a set of prohibitive norms, where a norm* n_i *is a rule of the form* $\varphi \underset{p}{\rightarrow} a$*, where* $\varphi = \{p_1,p_2,\ldots,p_n\}$ *is a set of atomic formulas as being a context activation with* $\varphi \subset \mathcal{L}$ *and* $a \in \mathcal{A} \cup \mathcal{A}'$*. Given a state* $s \in S$*, if* φ *holds in s, then the action a is prohibited to be executed in s.*

From now onward, we will use $\varphi(n_i)$ and $a(n_i)$ to refer, respectively, to the context activation and prohibited action of given a norm $n_i \in \mathcal{N}$. In the examples we refer to $a(n_i)$ by its *signature*.

Example 5. *A norm which may regulate the agents' behavior in order to avoid collisions on the grid may be defined as* $n_1 = \{\texttt{at}(\texttt{Ag}_2,\texttt{L}_2)\} \underset{p}{\rightarrow} \texttt{move}(\texttt{Ag}_1,\texttt{L}_1,\texttt{L}_2)$ *which, in natural language, it may be read as "if an agent* $\texttt{Ag}_2$ *is on a vertex* $\texttt{L}_2$*, then an agent* $\texttt{Ag}_1$ *is prohibited to move from a vertex* $\texttt{L}_1$ *to vertex* $\texttt{L}_2$*".*

Definition 9 (Norm Instance). *Given a norm* $n_i \in \mathcal{N}$, $\mathfrak{I}(n_i) = \{n_i^1,\ldots,n_i^m\}$ *is a set of instances of* n_i*, where an instance* $n_i^j = \varphi' \underset{p}{\rightarrow} a'$ *is a copy of* n_i *with* $\varphi' \subset \mathcal{L}$ *being a set of ground atomic formulas and* $Par(a') \subset Const$.

Example 6. *An instance of norm* n_1*, from example 5, is* $n_1^j = \{\texttt{at}(\texttt{ag}_2,\texttt{b})\} \underset{p}{\rightarrow} \texttt{move}(\texttt{ag}_1,\texttt{a},\texttt{b})$ *which prohibits agent* $\texttt{ag}_1$ *to move from the vertex* a *to the vertex* b *if agent* $\texttt{ag}_2$ *is on* b.

Definition 10 (Norm Applicability). *Given a state* $s \in S$ *and a norm* $n_i \in \mathcal{N}$*,* n_i *is said to be applicable (or active) in s if* $\exists n_i^j \in \mathfrak{I}(n_i)$*, such that,* $\varphi'(n_i^j), Pre(a'(n_i^j)) \subset s$.

Definition 11 (Norm Generality). *Let* $Sa(n_i)$ *and* $Sa(n_j)$ *be, respectively, the sets of system states in which the norms* $n_i,n_j \in \mathcal{N}$ *are applicable, with* $n_i \neq n_j$*. We say that* n_i *is more generic than* n_j*, denoted by* $n_i \underset{g}{>} n_j$*, if* $Sa(n_j) \subset Sa(n_i)$ *and* $Sa(n_i) \not\subset Sa(n_j)$.

Whether $n_i \underset{g}{>} n_j$, then n_i regulates all the behaviour that norm n_j does, however the opposite does not occur. Syntactically, we may say that $\varphi(n_i)$ subsumes $\varphi(n_j)$ and $a(n_i)$ subsumes $a(n_j)$.

Example 7. *Consider the following norms:*

$n_2 = \{\texttt{False}\} \underset{p}{\rightarrow} \texttt{move}(\texttt{ag}_1,\texttt{a},\texttt{b})$ $\quad$ $n_3 = \{\texttt{at}(\texttt{ag}_2,\texttt{b}),\texttt{next}(\texttt{a},\texttt{b})\} \underset{p}{\rightarrow} \texttt{move}(\texttt{ag}_1,\texttt{a},\texttt{b})$

$n_4 = \{\texttt{at}(\texttt{ag}_2,\texttt{L}_2),\texttt{next}(\texttt{L}_1,\texttt{L}_2)\} \underset{p}{\rightarrow} \texttt{move}(\texttt{ag}_1,\texttt{L}_1,\texttt{L}_2)$ $\quad$ $n_5 = \{\texttt{at}(\texttt{Ag}_2,\texttt{b}),\texttt{next}(\texttt{a},\texttt{b})\} \underset{p}{\rightarrow} \texttt{move}(\texttt{Ag}_1,\texttt{a},\texttt{b})$

$n_6 = \{\texttt{at}(\texttt{Ag}_2,\texttt{L}_2),\texttt{next}(\texttt{L}_1,\texttt{L}_2)\} \underset{p}{\rightarrow} \texttt{move}(\texttt{Ag}_1,\texttt{L}_1,\texttt{L}_2)$ $\quad$ $n_7 = \{\,\} \underset{p}{\rightarrow} \texttt{move}(\texttt{Ag}_1,\texttt{L}_1,\texttt{L}_2)$

The norm n_2 is the less generic norm since it does not apply to any system state. Thus, it is generalized by any other norm. On the other hand, since $\varphi(n_7)$ is empty, norm n_7 is applicable to all system states. Moreover, $a(n_7)$ subsumes all the other norm actions (since any substitution of its parameters to the parameters of the other actions is possible), then norm n_7 is the most generic norm. Considering the norm n_1 from example 5, we have two distinct orders of norm generalities (note that norms n_5 and n_4 are not comparable): i) $n_7 \underset{g}{>} n_1 \underset{g}{>} n_6 \underset{g}{>} n_5 \underset{g}{>} n_3 \underset{g}{>} n_2$; and ii) $n_7 \underset{g}{>} n_1 \underset{g}{>} n_6 \underset{g}{>} n_4 \underset{g}{>} n_3 \underset{g}{>} n_2$.

To each norm we assign a level of generality which is a value indicating how generic a norm is in a given partial order. The more generic a norm is, the bigger is its level of generality. Norms with certain properties in common are assigned with the same level of generality k. Such norms are generalized by those with they are comparable with a level $k+n$, with $n > 0$.

Definition 12 (**Level of Generality**). *The level of generality of a given norm $n \in N$, NG(n) is given by Equation 1*

$$NG(n) = \begin{cases} 0, & \text{if } \varphi(n) = \{False\} \\ \dfrac{1}{pred(n) + \frac{1}{1+var(n)}}, & \text{if } \varphi(n) = \{p_1, \ldots, p_n\} \\ \dfrac{1}{1+const(n)} + 1, & \text{if } \varphi = \{\,\} \end{cases} \quad (1)$$

In Equation 1, pred(n), var(n) and const(n) refer, respectively, to the number of atomic formulas, variables and constants of a norm n. According to such equation, the levels of generality are distributed along the interval [0,2]. The levels increase from the norms whose activation context are False (NG = 0), going towards norms whose activation context are formed by a set of atomic formulas different from False (NG $\in$]0,1]) up to norms whose activation contexts are empty sets (NG $\in$]1,2] - norms that are always true). The more atomic formulas and constants a norm has, the more specific (and mildly restrictive) it is. However, as the number of atomic formulas decreases and the number of variables increases, the norms becomes more generic (and restraining). This way, mildly restrictive norms are assigned with levels of generality which put them more directed to the left of the interval [0,2], while the restraining ones are assigned with levels which put them more directed to the right of the given interval.

Example 8. *The Figure 1 illustrates a partial order formed by the norms of example 7. Such ordering is represented as a graph, where nodes correspond to norms and edges to relations of generalities. Whether $n_i \underset{g}{>} n_j$, then the edge depart from n_j to n_i.The values in red correspond to the levels of generality of each norm.*

Definition 13 (**System Goal**). *Let $\mathcal{G} = \{p_1, \ldots, p_n\}$ be a set of ground atomic formulas as a declarative system goal, where $\mathcal{G} \subset \mathcal{L}$.*

Lastly, one can define what a Normative Multiagent System is in our perspective.

Definition 14 (**Normative MAS**). *Let nMAS = $\{Ag, \mathcal{D}, s_0, \mathcal{G}, \mathcal{N}, \mathcal{C}\}$ be a normative Multiagent System, such that, $Ag = \{ag_1, \ldots, ag_m\}$ denotes a set of agents, $\mathcal{D}$ denotes a MAS domain, $s_0 \in S$ denotes a system initial state, $\mathcal{G}$ denotes a system goal, $\mathcal{N} = \{n_1, \ldots, n_k\}$ denotes a set of norms and $\mathcal{C} = \{c_1, \ldots, c_j\}$ denotes a set of conflicts.*

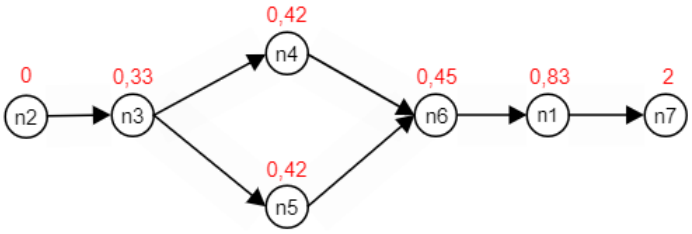


Figure 1. A partial order of norm generalities.

3. Norm Synthesis and System Execution

In this section we introduce the algorithms 1 and 2 which, respectively, synthesizes a set of norms and evaluates the norms' performance in the system execution. Basically, the algorithm 1 synthesises all ungrounded and grounded norms while the algorithm 2 groups the obtained results by level of generality. This way, we can explore the greatest amount of levels of generality and investigate how they impact on the system's goal reachability.

Algorithm 1 Norm Synthesis

Require: $\mathcal{M}$, $\mathcal{A}$, Pred $\subset \mathcal{L}$

1 $\mathcal{N}, \mathcal{N}' \leftarrow \emptyset$
2 $\mathcal{X} \leftarrow 2^{\text{Pred}} \times \mathcal{A}$
3 **foreach** $(\varphi, a) \in \mathcal{X}$ **do**
4 $\quad \mathcal{N} \leftarrow \varphi \underset{p}{\rightarrow} a$
5 **foreach** $n \in \mathcal{N}$ **do**
6 $\quad j \leftarrow$ getNumberOfTerms(n)
7 $\quad$ **foreach** $p \in [0, j]$ **do**
8 $\quad\quad \mathcal{N}' \leftarrow \mathcal{N}' \cup$
9 $\quad\quad$ getOtherNorms(n,$\mathcal{M}$,p)
10 **return** $\mathcal{N}'$

The algorithm 1 takes as input a set of atomic formulas Pred and actions $\mathcal{A}$, both ungrounded, and a mapper $\mathcal{M}$ which associates substitutions from variables to constants. Firstly, it synthesises all unground norms combining the possible activation contexts (with lengths from 0 to |Pred|) with the actions. For this, the algorithm calculates the power set of Pred multiplying it by $\mathcal{A}$ (line 2). To each pair (φ, a) of set $\mathcal{X}$, a norm is obtained and saved in the set $\mathcal{N}$ (line 4). Secondly, the algorithm obtains partial and grounded versions of norms of $\mathcal{N}$ through the method getOtherNorms (line 6). For each norm $n \in \mathcal{N}$, such method takes its variables p to p, with $0 \leq p \leq j$ (with j the number of terms of n obtained by the method *getNumberOfTerms*) to be substituted by all constants of same type (according to map $\mathcal{M}$). Each possible substitution generates a new norm which is assigned with a given level of generality (according to the definition 12) and is included in the set $\mathcal{N}'$ (consider the example 7, the norms n_1, n_6 and n_7 were obtained in the first step while n_2, n_3 and n_4 in the second one).

The algorithm 2 takes as input the previous set $\mathcal{N}'$ and map $\mathcal{M}$, the sets of ungrounded atomic formulas Pred$'$ and Pred$''$ (which are ungrounded versions of s_0 and $\mathcal{G}$ from definition 14, respectively) and a MAS. The algorithm works in three steps. Firstly, it obtains all possible initial states and system goals, S_0 and S_G, respectively, through the methods *getSMAInitialStates* (line 1) and *getSMAGoals* (line 2). Such methods instantiate the formulas of Pred$'$ and Pred$''$ by substituting their variables by constants (according to $\mathcal{M}$). In the second step, the MAS is executed multiple times through method *run* (line 6) varying its current *configuration* - a quadruple formed by a norm, an initial state, a goal system and a conflict. To certify the MAS' goal will not be satisfied before its launching, the MAS runs only if the intersection between its initial state and goal is empty (line 5).

As the configurations change, the agents will execute different sequences of actions to satisfy the system goal and, in this dynamic, different situations of conflicts may arise.

Algorithm 2 Norms Evaluation.

Require: $\mathcal{N}'$, $\mathcal{M}$, Pred$'$,Pred$''$ $\subset \mathcal{L}$, $c \in \mathcal{C}$, MAS

```
1  S_0 ← getSMAInitialStates(Pred', Map)
2  S_G ← getSMAGoals(Pred'', Map)
3  foreach norm n ∈ N' do
4      foreach state s_0 ∈ S_0 and G ∈ S_G do
5          if s_0 ∩ G = ∅ then
6              sat,tim,conf ← MAS.run(s_0, G, c, n)
7              storeResults(sat,tim,conf)
8  getLevelsPerformance()
```

Given a norm n, its performance in the system may: (i) avoid the conflict and make the system goals feasible to be reached; (ii) avoid the conflict to the detriment of system goals reachability (the system timeout since the effort to the goals to be satisfied becomes even greater or there are no alternative actions to be executed); (iii) not avoid the conflict which cause the system interruption (since we consider that conflicts are impediments to the system goals be satisfied). The method *run* returns a vector of length 3, indicating whether: the goal were satisfied (*sat* - according to case i), the execution finished by timeout (*tim* - according to case ii), or there was a conflict (*conf* - according to case iii). The results obtained at each execution are saved by method *storeResults* (line 7). At the end of the algorithm, the results are summarized by method *getLevelsPerformance* (line 12) which calculates, to each level of generality, the percentage of tests which ended up in goal satisfied, conflict occurrence and timeout. From the summarized results, we intend to identify how the norms' levels of generality influence on the system goals reachability.

4. Tests and Results

We have developed a simulator of MAS in Java, which implements the algorithms 1 and 2, to empirically evaluate the performance of norms in avoiding conflicts and keeping the system goals reachable. We perform experiments in two distinct scenarios: the traffic scenario domain, presented in section 2 and a scenario where agents (called as employees) try to access a set of files available by a given organization. In this scenario, agents may have individual goals which may be incoherent with the system goals (as open certain documents not specified by the system), where conflicts may arise since obtain certain information may compromise the organization's data and privacy. Thus, the norms must be able to regulate the agents in order to keep the data safe and the system goals reachable. We implemented a systematic sampling in methods *getInitialStates* and *getSMAGoals* of algorithm 2 with confidence level of 95% with a error margin of 5% for both scenarios. Moreover, in the graphics of the tests, the x-axis represents the levels of generality of norms while the y-axis represents rates of satisfiability, conflicts and timeout. Each scenario was tested ten times on a computer using a Intel Core i5 5200U processor running at 2.2GHz, with Ubuntu 16.10 64-bit as operating system and 8GB of memory.

4.1. Grid Scenario Evaluation

We tested the system varying the number of agents on the grid from 2 to 9. They used the *Manhattan distance* to estimate the best plan to reach the system goals. Let |agents| and |Vertices| be, respectively, the number of agents and vertices on the grid, then each agent

had $2^{|\text{Vertices}|}$ steps to conclude its plan and the system timeout in $|\text{agents}| * 2^{|\text{locals}|} + 1$ steps.

Figure 2 illustrates the curves of satisfiability rates. Generally speaking, the curves grow smoothly up to the level 0.8, reaching their highest value at the level 0.833 and, then, decreasing abruptly until the level 2. This happens because from the level 0 to 0.8 the norms are very specific which makes them inefficient in avoiding collisions as they hardly ever become active. However, as they become more generic (the number of atomic formulas decreases and the number of variables increases) the satisfiability rate grows somewhat since more agents' behaviors start to be regulated. Furthermore, as the number of agents increases, the rate of satisfiability decreases, since as more agents are on the grid, more collisions are likely to occur (for 2 agents, for example, the rate is initially about 66% while for 3 agents it is about 27%). At the level 0.833 there is uniquely the norm $n_1 = \{\texttt{at}(\texttt{Ag}_2,\texttt{L}_2)\} \underset{p}{\rightarrow} \texttt{move}(\texttt{Ag}_1,\texttt{L}_1,\texttt{L}_2)$ (from example 5 of section 2), thus no collision occurs when this norm is active. For 2, 3 and 4 agents on the grid, the satisfiability rate is *high* because it is easier to obtain alternative paths to achieve the system goal. However, for 5 or more agents, such rate is *low* since situations in which the agents are surrounded by other agents and can not move became common.

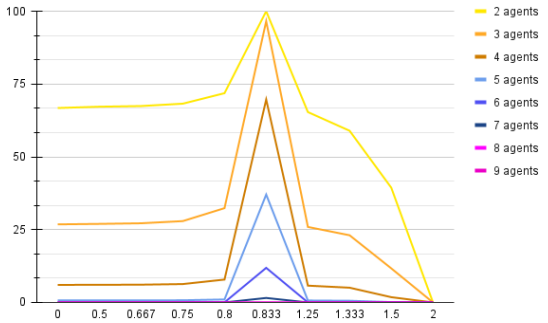


Figure 2. The rate of satisfiability for all executions.

After the level 0.833, the norms are of the form $\{\ \} \underset{p}{\rightarrow} a$. This type of norm are very generic and severely restrains the agents' autonomy since it applies to every system state. Such norms reduce the number of alternative actions to be taken in place of those that have been prohibited. Thus, achieving the system goal become a more complex or unfeasible task and, consequently, the satisfiability rate started to decrease. At level 2, there is uniquely the norm $n_7 = \{\ \} \underset{p}{\rightarrow} \texttt{move}(\texttt{Ag}_1,\texttt{L}_1,\texttt{L}_2)$ (from example 5 of section 2) which prevents the agents from moving, then no goal can be satisfied and the satisfiability rate is 0. Figure 4 illustrates the curves of conflict rates. As the number of agents increases the rate of conflicts increases as well. For all curves, such rate started in a given positive value and decreased smoothly until the level 0.8 becoming 0 at level 0.833 (due to the norm n_1 as explained previously). After that level, the rate of conflicts started to grow since the norms become more generic and strictly, thus the number of paths to be chosen became reduced and the agents were more susceptible to cross and collide. At level 2, the rate of conflicts became 0 since no agent can move (due to the norm n_7).

Lastly, figure 3 illustrates the curves of timeout rates. For all curves, as the level of generality gets bigger, the rate of timeout, which started as 0, keeps unchangeable until level 0.8. Such results correspond to executions with mildly restrictive norms. Since the agents had enough time to accomplish the system goals none of them got stuck in a given vertex and, consequently, there was no timeout for the interval [0,0.8] of levels of generality. Thus, conflicts occurred for some configurations and for others the system

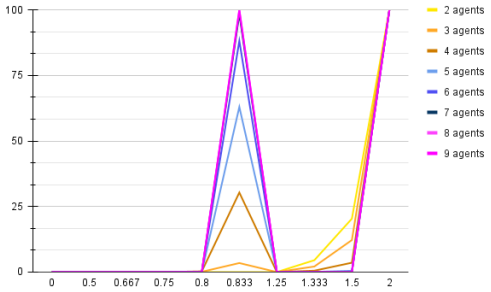


Figure 3. The rate of timeouts for all executions.

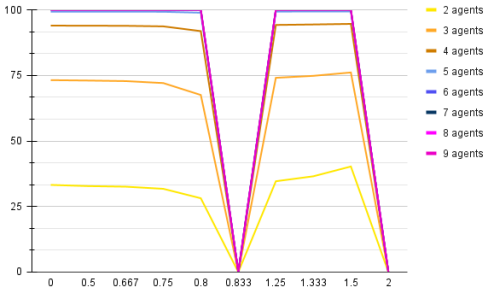


Figure 4. The rate of conflicts for all executions.

goals were satisfied. At level 0.833 the rate of timeout gets bigger as the number of agents increases on the grid. Since such results refers to executions with the norm n_1, the more agents are on the grid, the more vertices are occupied. Then, the agents got stuck in their depart vertices whereas was not possible to move to an adjacent vertex which was occupied. For all curves, the rate of timeout reached the value of 100% at level 2. This occurred because of norm n_7 which prevented the agents from moving to any vertex.

With these results we may think there exists a level of generality where norms tend to be capable of avoiding conflicts and keeping the goals reachable as much as possible. Let us call that level as *apex level*. According to the grid scenario, the *apex level* corresponds to the level 0.833 (around the center of the domain). In order to look further about the apex level, in the next section we provide some tests in another domain.

4.2. Security Domain Evaluation

In this scenario, an organization shares with its employees a set of documents. Such documents are classified as public or confidential while the employees, in turn, as low, medium or high according to the roles which they play in the company. The documents are available in a repository of a local server which is accessible from any computer of the organization's internal network. Then, to open a given document an employee must provide his credentials to access the repository. However, due to data protection and privacy issues the confidential documents must be open only by high employees. Thus, in this context, when a low or medium employee opens a document of that type a conflict situation occurs. Although opening a confidential document by those employees does not make part of the system goal, it can make part of the individual goal of some of them.

We have tested this scenario with 10 agents and 30 documents. Basically, the agents' behaviour consisted of entering the repository, trying to open the documents allocated to it by the system goal or those ones related to its individual goals and, finally, leaving the repository. Each initial state consisted in varying the classifications of employees and documents. Moreover, in all initial states the employees were logged out to the repository. On the other hand, each system goal consisted in varying the documents the employees should open. However, 30% of employees (low and medium) had individual goals which were not coherent with the system goal, that is, they intended to open confidential documents. Moreover, the agents decided with a 50% of chance whether they should accomplish the system goal or their individual ones.

Figure 5 illustrates the curves of satisfiability, conflict and timeout rates. The norm n_8 = $\{\texttt{roleLevel}(\texttt{Ag}_1,\texttt{low}), \texttt{info}(\texttt{Doc},\texttt{confidential})\} \underset{p}{\rightarrow} \texttt{open}(\texttt{Ag}_1,\texttt{Doc})$, which is at level 0.428, is the efficient norm for this scenario being capable of avoiding conflicts and making the system goal reachable. Such level corresponds to the apex level for this scenario and, differently from the grid scenario where there was only the norm n_1 at the corresponding apex level, it is composed by many norms. The most norms of the level 0.428 are inefficient in avoiding conflicts, then its rate of satisfiability is affected by the low system performance when they rule the agents' behaviours. This way, the shape of the current apex level is less evident than that of the grid domain. Nevertheless, the biggest rate of satisfiability is still found in the apex level (around 40%).

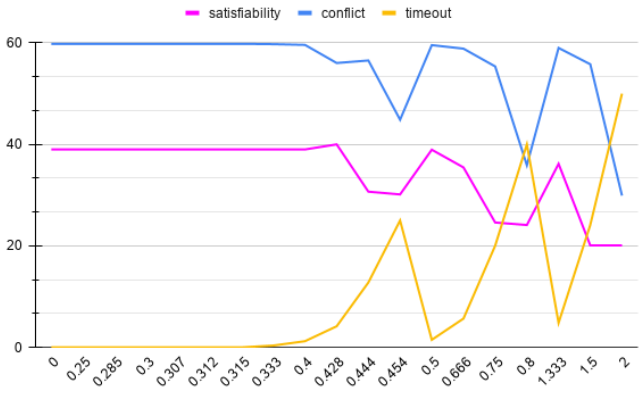


Figure 5. The results for Security Scenario.

As in the grid domain, before the apex level the norms are very specific and they hardly become active. Consequently, the agents have great autonomy to open the documents they wish which makes the rate of conflict for the interval [0,0.4] be high (around 60%) and the timeout be 0 (unless for levels 0.333 and 0.4). After the apex level, the norms start to become more strict having less atomic formulas. Whether a given norm prohibits an agent to open his target document and no alternative action in this scenario exist, the rate of timeout start to increase while the others rates decrease. However, the timeout declines from the level 0.454 to 0.5 and from 0.8 to 1.333 (which makes the others rates increase again). This happens because the levels 0.5 and 1.333 have one less formula, in relation to their previous levels, with all terms being constants. This way, the norms in levels 0.5 and 1.333 are not more genetic than those in the levels 0.454 and 0.8, respectively, thereby more agents became capable again to open their target documents which increases the conflicts and satisfiability rates. At last, the rate of timeout ends up 50% while the satisfiability and conflict ones ends up, respectively, 20% and 30%. Even though the current apex level has a satisfiability rate more discrete than that of the grid scenario, such apex level was found exactly in the center of the domain.

4.3. Discussion

According to the results, the apex level was found in the center of the domain or next to it. Before the apex level, the most norms showed to be mildly restrictive giving to the agents much freedom to act as they wished. Consequently, many conflicts occurred during the system execution. After the apex level, the most norms showed to be very strict giving to the agent low freedom to perform their actions. Consequently, the system did not satisfied its goal and timed out in many tests. On the basis of these data, we may say that norms which do not belong to the apex level impact negatively on the reachability of the system goal. On the other hand, the apex level showed to be an intermediate level of generality which have, at least, one norm capable to maximize the system goal

reachability and minimize the occurrence of conflicts balancing control and autonomy as much as possible.

The appearance of the apex level around the center of the domain is resulting from the way which our model organizes the norms (at least to problems which admit one norm as a solution to establish the social order). We suppose we can use this fact as a heuristic to guide a search to find an efficient norm for a given domain just synthesizing the norms from a selected set of levels of generality (those around the center of the domain). For domains where the apex level may not be found around their center, a mechanism to decide by which levels of generality the search must start still need be provided.

5. Related Works

A method to synthesize norms which gives the agents the maximum autonomy to accomplish their goals is proposed in [Fitoussi and Tennenholtz 1998]. The authors employed the concept of *minimality*. A norm n_1 is said to be minimal if it is useful and there is no other norm n_2 which is more specific than n_1. Although a minimal norm gives the agents great flexibly to perform their actions as they wish, a minimal norm is, in the context of our work, the most mildly restrictive norm. Depending on the application domain, the chances of conflicts occur may be high and, according to our tests, synthesizing norms with an intermediate level of generality can make the system performs better in terms of avoiding conflicts and satisfying goals.

In [Morales et al. 2013], a normative network in which norms are organized hierarchically via generalization relationships is proposed. During the system execution, as conflicts occur, norms are synthesized to avoid them in the future. Those norms are inserted in the network which has its structure updated in a way that just when all children of a potential generalization exist, a new norm is created to generalize them (the children become inactive).The network is built and restructured until no conflicts occur. However, until the system get in a stable situation, this is accomplished trough different conflicts occurrence. In many domains, such incidents are not admissible implying in high costs to implement recovery actions and making the system goals unfeasible to be reached.

Posteriorly, in [Morales et al. 2014] the same authors proposed a new method to obtain a normative network where the generalization relationships are inferred through an ontology which structures the domain knowledge as a tree representing a taxonomy of terms. In the same way as before, as conflicts occur, norms are synthesized to avoid them. Whether two or more norms are generalizable, a new norm is inferred to generalize them taking into account the terms in the ontology and selecting alternative more general terms. Such proposal also has the limitation to allow conflicts for a while. However, this new approach does not need that all children to its parent be inferred. Thus, the occurrence of conflicts may decrease along the system execution in relation to their former work.

6. Conclusion

In this paper we have presented a model for exploring the generality of norms. Our aim was to find out a level of generality where norms are capable to avoid conflict and keep the system goals reachable. We found, for two distinct domains, that there exists a level of generality, which we called as *apex level*, where exists a norm which tend to be efficient in avoiding conflicts and ensuring goals reachability as much as possible. As future works

we plan to evaluate empirically other scenarios where more than one type of conflict may occur and a set of distinct norms must be necessary to efficiently regulate the agents in order to further characterise the existence of the *apex level*.

References

Boella, G. and van der Torre, L. (2004). Regulative and constitutive norms in normative multiagent systems. In *Proc. of the Ninth Int. Conf. on Principles of Knowledge Representation and Reasoning*, KR'04, pages 255–265. AAAI Press.

Boella, G. and van der Torre, L. (2007). Norm negotiation in multiagent systems. *International Journal of Cooperative Information Systems (IJCIS)*, 16(2).

Christelis, G. and Rovatsos, M. (2009). Automated norm synthesis in an agent-based planning environment. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, volume 1 of *AAMAS '09*, pages 161–168, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Fitoussi, D. and Tennenholtz, M. (1998). Minimal social laws. In *Proc. of the Fifteenth National/Tenth Conf. on Artif. Intell./Innovative App. of Artif. Intell.*, page 26–31.

Frantz, C. K. and Pigozz, G. (2018). Modeling norm dynamics in multi-agent systems.

Morales, J., López-Sánchez, M., and Esteva, M. (2011). Using experience to generate new regulations. In *Proc. of IJCAI 2001*, page 307–312.

Morales, J., López-Sánchez, M., Rodríguez-Aguilar, J., Wooldridge, M., and Vasconcelos, W. (2014). Minimality and simplicity in the on-line automated synthesis of normative systems. In *AAMAS*.

Morales, J., Lopez-Sanchez, M., Rodriguez-Aguilar, J. A., Wooldridge, M., and Vasconcelos, W. (2013). Automated synthesis of normative systems. In *Proc. of AAMAS 2013*, pages 483–490.

Onn, S. and Tennenholtz, M. (1997). Determination of social laws for multi-agent mobilization. *Artificial Intelligence*, 95(1):155 – 167.

Savarimuthu, B. T. R., Cranefield, S., Purvis, M., and Purvis, M. (2008). Role model based mechanism for norm emergence in artificial agent societies. In *Proc. of COIN 2008*, pages 203–217.

Shoham, Y. and Tennenholtz, M. (1995). On social laws for artificial agent societies: off-line design. *Artificial Intelligence*, 73(1):231 – 252.

Melhorias na Sintaxe da Linguagem Jason

Jan Pierry Coelho dos Santos [1], **Jomi Fred Hübner**[2], **Jerusa Marchi**[1]

[1]Departamento de Informática e Estatística

[2]Departamento de Automação e Sistemas
Universidade Federal de Santa Catarina

janpierrycoelho@gmail.com, jomi.hubner@ufsc.br, jerusa.marchi@ufsc.br

Abstract. *Agent programming has been driven by several types of tools, such as the Jason programming language, which originally had academic purposes, but recently has also seen more use in real applications. However, throughout Jason's evolutionary process, several specific changes were made to it's grammar which, when reviewed in general, present some problems and potential for improvement. This article presents the problems identified and the improvements made in the grammar with a focus on simplicity, effectiveness and specificity without changing, as far as possible, the expressiveness of the language.*

Resumo. *A programação de agentes tem sido impulsionada por diversos tipos de ferramentas, como é o caso da linguagem de programação do Jason, que originalmente tinha fins acadêmicos, mas que vem sendo também utilizada em aplicações reais. No entanto, ao longo do processo de evolução do Jason, várias alterações pontuais foram feitas em sua gramática que, quando revistas de forma geral, apresentam alguns problemas e potencial de melhoria. Esse artigo apresenta alguns dos problemas identificados e as melhorias feitas na gramática com o foco em simplicidade, eficácia e especificidade sem alterar, na medida do possível, a expressividade da linguagem.*

1. Introdução

Assim como em outras áreas, a programação de agentes e sistemas multiagentes foi impulsionada pelo desenvolvimento de linguagens de programação e *frameworks* para fins específicos, como é o caso do Jason. O Jason é um *framework* para o desenvolvimento e simulação de sistemas multiagentes [Hübner et al. 2004, Bordini et al. 2007] que inicialmente tinha objetivos mais acadêmicos, mas que, com o passar do tempo, recebeu diversos usuários, tornando-se bastante utilizado em aplicações reais. É importante ressaltar que tanto o *framework*, quanto o interpretador e a linguagem que lhe são inerentes são chamados de Jason. A linguagem é uma extensão da linguagem de programação abstrata *AgentSpeak(L)* [Rao 1996] e é o foco principal deste trabalho.

O Jason teve sua primeira versão lançada no ano de 2004 [Hübner et al. 2004] e com o decorrer do tempo foi recebendo diversas atualizações[1]. Atualmente, ele se encontra na versão 2.6, sendo esta a versão utilizada como base deste trabalho. Contudo, para atender a crescente demanda dos sistemas multiagentes cada vez maiores e mais complexos, diversas mudanças foram feitas na linguagem Jason, muitas destas de forma

[1]http://jason.sourceforge.net

rápida visando atender às necessidades de usuários específicos. Ao longo do tempo, tais modificações resultaram em alguns problemas na estrutura da gramática da linguagem e na necessidade de um processo criterioso de revisão.

Desta forma, este trabalho tem como objetivos analisar a estrutura sintática da linguagem Jason de forma a identificar seus problemas, elaborar soluções para os problemas identificados, integrar tais soluções no código do interpretador Jason e por fim, realizar testes visando validar as mudanças e verificar se elas não impactam negativamente no correto funcionamento dos códigos já desenvolvidos na linguagem. Para tanto, o trabalho está estruturado da seguinte forma. Na Seção 2 é apresentada a estrutura sintática da linguagem Jason, apontando exemplos dos principais problemas encontrados. Na Seção 3 são apresentadas as soluções encontradas para os problemas relatados. As Seções 4 e 5 mostram como a nova estrutura sintática foi integrada ao nível semântico do Jason e a sua resposta com relação aos testes realizados. Por fim, a Seção 6 apresenta algumas considerações finais acerca do trabalho desenvolvido.

2. Estrutura sintática

A estrutura sintática de uma linguagem de programação é descrita por uma Gramática Livre de Contexto $G = (N, T, P, S)$ onde N é o conjunto de não-terminais ou categorias sintáticas, T é o conjunto de terminais, ou seja itens léxicos encontrados no texto fonte, como nomes dos agentes, nomes dos predicados e palavras reservadas. P é o conjunto de regras de produção que são estruturadas na forma $\alpha ::= \beta$ que significa que a forma sentencial $\alpha \in N$ será sobrescrita por β, onde $\beta \in \{N \cup T\}^*$[2]. Por fim, S se refere ao símbolo inicial da gramática, tal que $S \in N$.

A gramática de uma linguagem é normalmente descrita utilizando o formalismo Backus-Naur, do inglês *Backus-Naur formalism* (BNF), que padroniza e estende as regras de produção, permitindo uso de operadores como $*$ (fecho de Kleene) e $+$ (fecho positivo de Kleene) facilitando a definição das estruturas sintáticas.

A Figura 1 apresenta a gramática da linguagem Jason, onde é possível observar a constituição das regras de produção em BNF, como na produção `agent`. Já na produção `directive`, observam-se exemplos de categorias sintáticas, como `pred`, e terminais como ⟨`TK_BEGIN`⟩. É importante ressaltar que o item léxico `begin` encontrado no programa fonte será convertido no *token* `TK_BEGIN` quando analisado.

Após a análise e compreensão geral da estrutrura sintática da linguagem, tornou-se possível investigar as suas produções de forma a identificar os principais problemas.

2.1. Identificando Problemas

O interpretador Jason foi gerado utilizando o JavaCC[3], que é uma ferramenta *open source* para a geração de analisadores sintáticos ou *parsers* em código Java. *Parsers* gerados pelo JavaCC, não fazem uso de *backtracking* e, quando diante de um conflito de escolha entre duas ou mais opções de caminho de expansão, sempre escolherão a primeira. Para possibilitar a escolha diante de duas ou mais opções de caminhos de expansão possíveis,

[2]Onde $*$ é o fecho de Kleene e representa qualquer número de repetições de não-terminais e terminais concatenados, inclusive zero.

[3]https://javacc.github.io/javacc/

```
            agent ::= ( directive )* ( ( belief | initial_goal | plan )+ ( directive )* )* <EOF>
        directive ::= "{" ( <TK_BEGIN> pred "}" agent | pred "}" )
           belief ::= literal ( ":-" log_expr )? "."
     initial_goal ::= "!" literal "."
             plan ::= ( <TK_LABEL_AT> ( literal | list ) )? trigger ( ":" log_expr )? ( "<-" plan_body )? "."
          trigger ::= ( "+" | "-" | "^" ) ( ( "!" | "?" ) )? literal
        plan_body ::= plan_body_term ( ";" )? ( plan_body )?
   plan_body_term ::= plan_body_factor ( <TK_POR> plan_body_term )?
  plan_body_factor ::= ( stmtIF | stmtFOR | stmtWHILE | body_formula ) ( <TK_PAND> plan_body_factor )?
          stmtIF ::= <TK_IF> stmtIFCommon
     stmtIFCommon ::= "(" log_expr ")" rule_plan_term ( ( <TK_ELIF> stmtIFCommon | <TK_ELSE> rule_plan_term ) )?
         stmtFOR ::= <TK_FOR> "(" log_expr ")" rule_plan_term
        stmtWHILE ::= <TK_WHILE> "(" log_expr ")" rule_plan_term
     body_formula ::= ( "!" | "!!" | "?" | ( "+" ( ( "+" | "<" | ">" ) )? ) | ( "-" ( "+" | "-" )? ) )? ( log_expr )
   rule_plan_term ::= "{" ( ( <TK_LABEL_AT> ( pred | var ) )? trigger ( ":" log_expr )? ( ( "<-" | ";" ) )? )?
                          ( literal ":-" log_expr )? ( plan_body )? "}"
          literal ::= ( ( ( ( <ATOM> | var ) )? "::" )? ( <TK_NEG> )? ( pred | var ) ) | <TK_TRUE> | <TK_FALSE> )
             pred ::= ( <ATOM> | <TK_BEGIN> | <TK_END> ) ( "(" terms ")" )? ( list )?
            terms ::= term ( "," term )*
             term ::= log_expr
             list ::= "[" ( term_in_list ( "," term_in_list )* )? ( "|" ( <VAR> | <UNNAMEDVAR> | list ) )? "]"
     term_in_list ::= ( list | arithm_expr | string | rule_plan_term )
         log_expr ::= log_expr_trm ( "|" log_expr )?
     log_expr_trm ::= log_expr_factor ( "&" log_expr_trm )?
   log_expr_factor ::= ( <TK_NOT> log_expr_factor | rel_expr )
         rel_expr ::= ( arithm_expr | string | list | rule_plan_term )
                      ( ( "<" | "<=" | ">" | ">=" | "==" | "\\==" | "=" | "=.." )
                      ( arithm_expr | string | list | rule_plan_term ) )?
      arithm_expr ::= arithm_expr_trm ( ( "+" | "-" ) arithm_expr_trm )*
   arithm_expr_trm ::= arithm_expr_factor ( ( "*" | "/" | <TK_INTDIV> | <TK_INTMOD> ) arithm_expr_factor )*
arithm_expr_factor ::= arithm_expr_simple ( ( "**" ) arithm_expr_factor )?
arithm_expr_simple ::= ( <NUMBER> | "-" arithm_expr_simple | "+" arithm_expr_simple | "(" log_expr ")" | function )
          function ::= literal
               var ::= ( <VAR> | <UNNAMEDVARID> | <UNNAMEDVAR> ) ( list )?
            string ::= <STRING>
```

Figura 1. Gramática do Jason antes das alterações

o JavaCC permite que se defina um *lookahead* maior, ou seja, que o *parser* explore *tokens* mais à frente no fluxo de entrada, auxiliando a tomada de decisão [Aho et al. 2008].

Desta forma, para identificar problemas no interpretador Jason passou-se a observar os pontos de conflito existentes nas especificações léxica e sintática da linguagem Jason, que estão descritas no arquivo `AS2JavaParser.jj`. Este é o arquivo utilizado pelo JavaCC para produzir o *parser*. Foram identificados 12 pontos de conflito, relacionados a 4 *lookaheads* no código, cujos trechos de código são mostrados na Figura 2, 2 deles mais críticos, que verificam 47 e 150 *tokens* à frente do fluxo de entrada, e outros 2 mais simples que verificam apenas 4 *tokens* à frente.

O passo seguinte foi a análise de cada conflito. Dos 12 conflitos existentes, 8 estavam relacionados aos 2 *lookaheads* mais críticos. Os outros 4 conflitos eram relacionados a estrututura das produções da gramática, pois seus *lookaheads* não eram grandes.

Para auxiliar nesta análise, foram aplicados os conceitos de *First* e *Follow* [Aho et al. 2008]. O conjunto *First* identifica quais terminais podem iniciar uma forma sentencial a partir de um determinado não terminal que se encontra em análise, ou seja, qual é a produção que deve ser aplicada na derivação, logo é possível afirmar que existe um conflito de escolha se algum *token* do *First* de um dos caminhos de expansão também faz parte do *First* de outro caminho de expansão.

```
void directive() :
{}
{
  "{" ( LOOKAHEAD(4) <TK_BEGIN> pred() "}" agent() | pred() "}" )
}

void rule_plan_term():
{}
{
  "{"
    [ LOOKAHEAD(4) [ <TK_LABEL_AT> ( pred() | var() ) ] trigger() [ ":" log_expr() ] [ ( "<-" | ";" ) ] ]
    [ LOOKAHEAD(150) literal() ":-" log_expr() ]
    [ plan_body() ]
  "}"
}

void literal() :
{}
{
  ( [ LOOKAHEAD(47) [ ( <ATOM> | var() ) ] "::" ] [ <TK_NEG> ] ( pred() | var() ) ) | <TK_TRUE> | <TK_FALSE>
}
```

Figura 2. Lookaheads presentes na Gramática do Jason

Já o conjunto *Follow* permite saber, para cada não terminal, quais são os terminais que podem o suceder durante a avaliação, ou seja, permite identificar se um não terminal pode ser retirado da pilha de análise ou ainda quando a análise da forma sentencial derivada daquele não terminal acabou. Também é possível usar este conjunto para auxiliar na identificação do *First* em cenários específicos.

```
void agent() :
{}
{
  ( directive() )* ( ( belief() | initial_goal() | plan() )+ ( directive() )* )* <EOF>
}
```

Figura 3. Produção `agent` original.

Como exemplo do processo de análise dos conflitos, considere a produção `agent` apresentada na Figura 3, que é também o símbolo inicial da gramática da linguagem Jason e onde o ponto de conflito indicado pelo JavaCC se refere à parte onde ocorre a escolha entre os não-terminais `belief`, `initial_goal` e `plan`, cujos conjuntos *First* são os seguintes:

$$
\begin{aligned}
\mathit{First}(\texttt{belief}) &= \{\langle \texttt{ATOM}\rangle, \langle \texttt{VAR}\rangle, \langle \texttt{UNNAMEDVARID}\rangle, \langle \texttt{UNNAMEDVAR}\rangle, \\
&\quad \langle \texttt{TK\_NEG}\rangle, \langle \texttt{TK\_BEGIN}\rangle, \langle \texttt{TK\_END}\rangle, \langle \texttt{TK\_TRUE}\rangle, \\
&\quad \langle \texttt{TK\_FALSE}\rangle\} \\
\mathit{First}(\texttt{initial\_goal}) &= \{\text{“ ! ”}\} \\
\mathit{First}(\texttt{plan}) &= \{\langle \texttt{TK\_LABEL\_AT } \rangle, \text{“ + ”}, \text{“ - ”}, \text{“ ^ ”}\}
\end{aligned}
$$

Como é possível perceber na Figura 3 a escolha entre `belief`, `initial_goal` e `plan` possui o fechamento positivo de Kleene (+), indicando que o que está entre parênteses deve aparecer uma vez ou mais, por conta disso, após uma das opções aparecer, o *parser* tem como opção expandir novamente alguma delas ou seguir adiante e expandir `directive`. Levando isso em conta, também foi preciso considerar o *First* de `directive` para verificar o conflito.

$$\mathit{First}(\texttt{directive}) = \{\text{“ \{ ”}\}$$

Portanto, como não há *tokens* em comum entre os *Firsts* desses não-terminais, é possível afirmar que não existe conflito entre `belief`, `initial_goal`, `plan` e `directive`. Contudo, ainda assim é apresentado o aviso de conflito de escolha. A razão disso se dá pela forma como a produção `agent` foi estruturada, isso porque a parte `((belief | initial_goal | plan)+ (directive)*)*` possui o fechamento de Kleene (*) em volta, indicando que tudo entre parênteses pode aparecer quantas vezes forem necessárias, inclusive nenhuma, fazendo com que, caso apareça uma das produções das opções (`belief`, `initial_goal` e `plan`) no fluxo de entrada e na sequência apareça novamente mais uma delas, o *parser* não vai saber se deve expandir essa segunda a partir de `( belief | initial_goal | plan )+` ou se deve finalizar essa parte, desconsiderar `directive` e partir para mais uma iteração disso tudo. Por conta disso, é possível afirmar que colocar o fechamento positivo de Kleene em volta da escolha de `belief`, `initial_goal` e `plan` é redundante. Por fim, independentemente da escolha que o *parser* tome mediante este conflito, nenhuma delas resultará em um erro de interpretação, portanto o *lookahead* não é necessário, mas essa produção pode ser melhor estruturada, visando eliminar o conflito.

Além deste exemplo de conflito, um outro que vale destacar é o da produção `rule_plan_term`, apresentada na Figura 4, isso porque este conflito possui o maior *lookahead* encontrado na gramática do Jason, verificando 150 *tokens* à frente no fluxo de entrada. A causa do conflito se dá pela escolha que o *parser* tem entre expandir o não-terminal `literal` (na Figura 4, logo após o *LOOKAHEAD* (150), e expandir o não-terminal `plan_body` da linha seguinte, que também pode expandir `literal`. Dessa forma, o número de *tokens* a serem verificados à frente no fluxo de entrada deve levar em conta a quantidade de *tokens* que podem ser expandidos a partir de `literal`. No entanto, o que faz com que esse problema seja tão grave é que `literal` pode expandir o não-terminal `pred`, que pode expandir o não-terminal `list`, e que por fim pode expandir recursivamente mais não-terminais `list`. Por conta disso, tem-se que esse é um problema estrutural e que qualquer valor que venha a ser colocado no *lookahead* não resolverá este conflito de forma definitiva.

```
void rule_plan_term() :
{}
{
  "{"
    [ LOOKAHEAD(4)
      [ <TK_LABEL_AT> ( pred() | var() ) ]
      trigger()
      [ ":" log_expr() ] [ ( "<-" | ";" ) ]
    ]
    [ LOOKAHEAD(150) literal() ":-" log_expr() ]
    [ plan_body() ]
  "}"
}
```

Figura 4. Produção `rule_plan_term` original.

Para além da análise dos conflitos, para a identificação dos problemas considerou-se ainda a questão da especificidade das produções, ou seja, o quão bem uma produção representa o recurso da linguagem ao qual se propõe. Um bom exemplo disso se encontra na produção `directive`, cuja representação simplificada se encontra na Figura 5. De

forma a entender o problema em sua estrutura, é importante inicialmente ressaltar que as diretivas na linguagem Jason podem se dar de duas formas: com `begin` e `end`, e sem `begin` e `end`. Logo, a partir do ponto que uma diretiva inicia com *begin*, ela obrigatoriamente deve finalizar com *end*. Contudo, como é possível perceber na Figura 5, a produção `directive` possui um *token* ⟨`TK_BEGIN`⟩ mas não possui um *token* ⟨`TK_END`⟩. A razão para que mesmo assim ela ainda funcione é que, após o não-terminal `pred` e o *token* "}", há o não-terminal `agent`, que tem a opção de expandir novamente o não-terminal `directive`, que pode seguir pelo não-terminal `pred` ao final da produção, que pode então expandir o *token* ⟨`TK_END`⟩. No entanto, o *parser* pode expandir outros *tokens* no lugar de ⟨`TK_END`⟩ em `pred`, como ⟨`ATOM`⟩ ou até mesmo outro ⟨`TK_BEGIN`⟩. Contudo, como foi comentado, é imprescindível que uma diretiva que inicie com `begin` termine com `end`, por conta disso, já que o *parser* considera como válida uma diretiva que não atende a esse requisito é possível afirmar que essa produção não é específica.

```
void directive() :
{}
{
  "{" ( LOOKAHEAD(4) <TK_BEGIN> pred() "}" agent() | pred() "}" )
}
```

Figura 5. Produção `directive` original.

A existência desse tipo de inconformidade não implica necessariamente que códigos mais abrangentes, que não se adequam ao que o recurso da linguagem realmente representa, vão ser considerados válidos, pois ainda é possível que a análise semântica venha a detectar inconformidades como essa. No entanto, é preciso considerar que estes são problemas estruturais e portanto deveriam ser tratados pelo analisador sintático, pois cabe ao analisador semântico tratar apenas de questões relacionadas ao significado dos comandos, como por exemplo a incompatibilidade de tipos ou redeclarações de variáveis.

3. Soluções Elaboradas

A seção anterior ilustrou três exemplos de problemas encontrados após a análise da gramática do Jason. O passo seguinte foi elaborar soluções buscando eliminar os conflitos e melhorar a estrutura das produções. Para tanto, três características principais foram consideradas na construção das soluções: simplicidade, eficácia e especificidade.

3.1. Simplicidade

Para que a estrutura da gramática se tornasse mais organizada e legível, as melhorias foram pensadas de forma que as produções ficassem as mais simples possíveis. Como exemplo, reconsidere a produção `agent` apresentada na Figura 3. Como comentado anteriormente, a sua estrutura original é mais complexa do que o necessário, já que o fechamento positivo de Kleene englobando a escolha entre `befief`, `initial_goal` e `plan` é redundante. Todavia, por mais que a sua remoção resolva o conflito, também é possível reestruturar a produção de forma que se obtenha uma produção mais simples, como apresentado na Figura 6. Segundo esta nova estrutura, um agente pode ser definido como uma sequência de `directive`, `belief`, `initial_goal` e `plan` podendo aparecer quantas vezes forem necessárias e em qualquer ordem. A nova produção não é nem mais restritiva e nem abrangente que a especificação original e por conta disso é possível afirmar que ambas possuem o mesmo valor sintático.

```
void agent() :
{}
{
  ( directive() | belief() | initial_goal() | plan() )* <EOF>
}
```

Figura 6. Produção `agent` modificada.

3.2. Eficácia

Além da simplicidade, outro aspecto levado em consideração foi a eficácia das produções. Isso porque, como ilustrado no caso da produção `rule_plan_term` (apresentada na Figura 4), existiam conflitos na gramática que, mesmo com o aumento do *lookahead*, não seriam resolvidos. Para exemplificar como se deu a solução desse tipo de problema, considere a produção alterada apresentada na Figura 7. A nova estrutura possui outras alterações focadas em resolver outros problemas na produção, contudo, a mudança que foi feita para resolver o problema da eficácia se encontra na parte `plan_body` [":-" `log_expr`]. Esta parte foi modificada de forma a representar ambas as opções conflitantes deste contexto, que eram os recursos da linguagem: `rule_term`, representado por `literal` ":-" `log_expr`; e `plan_body_only` que, como o próprio nome diz, consiste apenas do não-terminal `plan_body`. O recurso `plan_body_only` pode ser representado expandindo `plan_body` e desconsiderando a parte opcional [":-" `log_expr`], e o recurso `rule_term` pode ser representado com o `literal` sendo expandido através de `plan_body`, que era a causa do conflito apresentado na Figura 4, e depois expandindo a parte opcional [":-" `log_expr`].

```
void rule_plan_term() :
{}
{
    "{"
      [
        [ plan_term_annotation() ]
            trigger() [ ":" log_expr() ] [ ( ";" [ plan_body() ] | "<-" plan_body() ) ]
        | plan_body() [ ":-" log_expr() ]
      ]
    "}"
}
```

Figura 7. Produção `rule_plan_term` modificada.

No entanto essa nova estrutura é uma representação mais abrangente do que a anterior, já que permite que outros *tokens* que possam ser expandidos através de `plan_body` sejam seguidos de ":-" `log_expr`, o que na prática não deve ser permitido. Esse problema foi solucionado com uma anotação de código que verifica se, quando a parte que é exclusiva de `rule_term` está presente, `plan_body` é composto de apenas um único `literal`, reportando um erro nesse caso. Por mais que a solução de problemas sintáticos através de anotações de código não seja ideal, o caso de `rule_plan_term` era bastante complexo e exigiu tal abordagem visando sua solução definitiva.

3.3. Especificidade

O último pilar teve como objetivo que as produções fossem representações mais fiéis dos seus respectivos recursos da linguagem. Exemplos de melhorias elaboradas com esse

foco podem ser observadas no caso da produção `directive`, apresentada na Figura 5. Como comentado anteriormente, esta produção possui problemas de especificidade por não restringir sintaticamente que diretivas que iniciam com *begin* terminem com *end*. A produção foi reestruturada conforme apresentado na Figura 8.

```
void directive() :
{}
{
    "{" (
          <TK_BEGIN> directive_argument() "}" ( agent_component() )* "{" <TK_END> "}"
          | directive_argument() "}"
        )
}
```

Figura 8. Produção `directive` modificada.

Assim como em `rule_plan_term`, essa produção possui mais alterações focadas em resolver outros problemas. Contudo, a solução para o problema de especificidade pode ser observada no primeiro caminho de expansão dessa produção, onde têm-se que ao iniciar a diretiva com o *token* ⟨`TK_BEGIN`⟩ é obrigatório que exista o fim dessa diretiva com o *token* ⟨`TK_END`⟩.

3.4. Resultado da nova estrutura

Ao juntar essas e as outras produções que tiveram correções com as produções que não foram alteradas, obteve-se a nova estrutura sintática da linguagem Jason. Ao gerar o *parser* foi constatada a existência de 6 novos conflitos de escolha para os quais, novamente, foi necessária a análise individual de forma a verificar se não representam problemas para o correto funcionamento do interpretador.

Destes novos conflitos, 2 puderam ser solucionados com o uso de lookaheads que verificam apenas 2 *tokens* à frente no fluxo de entrada, outros 3 já existiam na estrutura original do interpretador Jason e não necessitaram de tratamento pois o primeiro caminho de expansão possível sempre é o correto por uma questão de precedência de operadores, e por fim, o último conflito se referia a um problema de eficácia da estrutura original e foi solucionado através de um recurso do JavaCC que permite o uso de não-terminais no *lookahead* no lugar de um número de *tokens*, como mostra a Figura 9.

```
void literal() :
{}
{
    [ LOOKAHEAD(namespace()) namespace() ] [ <TK_NEG> ] ( pred() | var() ) | <TK_TRUE> | <TK_FALSE>
}
```

Figura 9. Produção `literal` modificada.

Essa modificação faz com que o *parser* só expanda o não-terminal `namespace` quando tiver certeza de que o código que está sendo analisado realmente representa este não-terminal. Desta forma, concluiram-se as alterações possíveis visando melhor estruturar a gramática da linguagem Jason. A nova gramática é apresentada na Figura 10.

4. Integração

A última etapa foi a integração na nova estrutura sintática ao interpretador Jason, gerando uma nova versão da linguagem. Para isto foi necessário analisar a ligação que existia

```
              agent ::= ( agent_component )* <EOF>
    agent_component ::= ( directive | belief | initial_goal | plan )
          directive ::= "{" ( <TK_BEGIN> directive_arguments "}" ( agent_component )* "{" <TK_END> "}" | directive_arguments "}" )
directive_arguments ::= <ATOM> ( "(" terms ")" )? ( list )?
             belief ::= literal ( ":-" log_expr )? "."
       initial_goal ::= "!" literal "."
               plan ::= ( plan_annotation )? trigger ( ":" log_expr )? ( "<-" plan_body )? "."
    plan_annotation ::= <TK_LABEL_AT> ( ( <ATOM> | <TK_BEGIN> | <TK_END> ) ( list )? | list )
            trigger ::= ( "+" | "-" | "^" ) ( ( "!" | "?" ) )? literal
          plan_body ::= ( plan_body_term ( ";" ( plan_body )? )? | statement ( ";" )? ( plan_body )?
     plan_body_term ::= plan_body_factor ( <TK_POR> plan_body_term )?
   plan_body_factor ::= body_formula ( <TK_PAND> plan_body_factor )?
          statement ::= ( stmtIF | stmtFOR | stmtWHILE )
             stmtIF ::= <TK_IF> stmtIFCommon
       stmtIFCommon ::= "(" log_expr ")" "{" ( stmt_body )? "}" ( <TK_ELIF> stmtIFCommon | <TK_ELSE> "{" ( stmt_body )? "}" )?
            stmtFOR ::= <TK_FOR> "(" log_expr ")" "{" ( stmt_body )? "}"
          stmtWHILE ::= <TK_WHILE> "(" log_expr ")" "{" ( stmt_body )? "}"
          stmt_body ::= plan_body
       body_formula ::= ( ( ( "!" | "!!" ) literal ) | ( ( "?" | "+" ( "+" | "<" | ">" )? | "-" ( "+" | "-" )? )? log_expr ) )
     rule_plan_term ::= "{" ( ( plan_term_annotation )? trigger ( ":" log_expr )? ( ( ";" ( plan_body )? | "<-" plan_body ) )?
                          | plan_body ( ":-" log_expr )? )? "}"
plan_term_annotation ::= <TK_LABEL_AT> ( ( <ATOM> | <TK_BEGIN> | <TK_END> ) ( list )? | var | list )
            literal ::= ( ( namespace )? ( <TK_NEG> )? ( pred | var ) | <TK_TRUE> | <TK_FALSE> )
          namespace ::= ( <ATOM> | var )? "::"
               pred ::= ( <ATOM> | <TK_BEGIN> | <TK_END> ) ( "(" terms ")" )? ( list )?
              terms ::= term ( "," term )*
               term ::= log_expr
               list ::= "[" ( term_in_list ( "," term_in_list )* )? ( "|" ( <VAR> | <UNNAMEDVAR> | list ) )? "]"
       term_in_list ::= ( list | arithm_expr | string | rule_plan_term )
           log_expr ::= log_expr_trm ( "|" log_expr )?
       log_expr_trm ::= log_expr_factor ( "&" log_expr_trm )?
    log_expr_factor ::= ( <TK_NOT> )? rel_expr
           rel_expr ::= ( arithm_expr | string | list | rule_plan_term ) ( ( "<" | "<=" | ">" | ">=" | "==" | "\\==" | "=" | "=.." )
                        ( arithm_expr | string | list | rule_plan_term ) )?
        arithm_expr ::= arithm_expr_trm ( ( "+" | "-" ) arithm_expr_trm )*
    arithm_expr_trm ::= arithm_expr_factor ( ( "*" | "/" | <TK_INTDIV> | <TK_INTMOD> ) arithm_expr_factor )*
 arithm_expr_factor ::= arithm_expr_simple ( ( "**" ) arithm_expr_factor )?
 arithm_expr_simple ::= ( "+" | "-" )? ( <NUMBER> | ( "(" log_expr ")" ) | function )
           function ::= literal
                var ::= ( <VAR> | <UNNAMEDVARID> | <UNNAMEDVAR> ) ( list )?
             string ::= <STRING>
```

Figura 10. Nova gramática do Jason.

entre a estrutura original da linguagem e as anotações de código que fazem a verificação semântica e a geração de código. A dificuldade de integração das produções varia dependendo da complexidade e também do tamanho das mudanças realizadas. Um exemplo simples que ilustra bem como se deu o processo de integração é o caso da produção `stmtFOR`, apresentada na Figura 11.

Como resultado das melhorias feitas na estrutura desta produção, ao invés de utilizar `rule_plan_term` como corpo do comando, é utilizado "{" `stmt_body` "}", como mostra a Figura 12. Essa mudança foi feita pois a linguagem Jason só permite que os comandos `if`, `for` e `while` tenham como corpo o não-terminal `plan_body` e `rule_plan_term` permitia que outras coisas fossem expandidas. Desta forma, o código que era necessário para verificar se `rule_plan_term` era composto apenas de `plan_body` pôde ser removido e em seu lugar foi adicionada uma verificação para o cenário do corpo estar vazio em `stmtFOR`.

De forma similar, procedeu-se às demais alterações visando a integração entre as partes sintática e semântica.

```
PlanBody stmtFOR() :   { Object B; Term T1; Literal stmtLiteral; }
{
  <TK_FOR>
  "("
  B = log_expr()
  ")"
  T1 = rule_plan_term()

            { try {
                if (T1.isRule()) {
                  throw new ParseException(getSourceRef(T1)+"for requires a plan body.");
                }
                stmtLiteral =
                  new InternalActionLiteral(ASSyntax.createStructure(".foreach", (Term)B, T1), curAg);
                stmtLiteral.setSrcInfo( ((Term)B).getSrcInfo() );
                return new PlanBodyImpl(BodyType.internalAction, stmtLiteral);
              } catch (Exception e) {
                e.printStackTrace();
              }
            }
}
```

Figura 11. Integração da produção `stmtFOR` original.

```
PlanBody stmtFOR() :
{
  Object B; PlanBody pb = null; Literal stmtLiteral;
}
{
  <TK_FOR> "(" B = log_expr() ")" "{" [ pb = stmt_body() ] "}"

            {
              try {
                if (pb == null) {
                  pb = new PlanBodyImpl();
                }
                stmtLiteral =
                  new InternalActionLiteral(ASSyntax.createStructure(".foreach", (Term)B, pb), curAg);
                stmtLiteral.setSrcInfo( ((Term)B).getSrcInfo() );
                return new PlanBodyImpl(BodyType.internalAction, stmtLiteral);
              } catch (Exception e) {
                e.printStackTrace();
              }
            }
}
```

Figura 12. Integração da produção `stmtFOR` modificada.

5. Testes

Foram diversas as mudanças implementadas, resultando em um novo arquivo do *parser* bastante distinto do original. Para assegurar que a nova versão do Jason suporte os códigos já desenvolvidos, não afetando os seus usuários, foram feitos testes de funcionamento. O Jason possui uma série de testes que foram implementados com o objetivo de validar seu funcionamento após mudanças em sua estrutura. Esses testes foram utilizados neste trabalho. O processo de integração e testes ocorreu de forma concomitante e, em caso de falha nos testes, ajustes na gramática e nas ações semânticas foram realizados. Eventualmente, alguma modificação foi necessária nos testes, pois ao tornar algumas produções mais específicas, algumas construções deixaram de ser válidas. Abaixo são listadas algumas construções que agora são inválidas na linguagem e a forma nova, que é aceita pelo *parser*.

5.1. Caso do *token* $\langle TK\_NOT \rangle$

Na definição original, era possível usar o `not` diversas vezes em sequência, como em `not not expressão`, devido a estrutura da produção `log_expr_factor` (veja Figura 1).

Com a alteração desta produção conforme apresentado na Figura 10, o comando passa a ser `not (not expressão)`.

5.2. Caso dos Operadores "+" e "-"

De forma similiar ao caso anterior, a definição original permite que se declare sequências de "+" e "-" antes de seguir para um número, função ou literal devido a estrutura da produção `arithm_expr_simple`, conforme apresentado na Figura 1. Um exemplo possível desta construção é `+ + - - - + number`. Com a nova estrutura desta produção, apresentada na Figura 10, tal construção não é mais possível.

5.3. Caso da construção `plan_body`

A terceira construção, ao contrário dos anteriores, tem um impacto significativo, pois modifica a forma como o corpo dos planos é declarado. Na gramática original, a produção `plan_body` (veja Figura 1) permite que os termos do corpo sejam separados ou não por "`;`" e também permitia o "`;`" depois do último comando, antes do "`.`", conforme ilustra a Figura 14.

```
+!plano
   <-
   !primeiroComando
   ?segundoComando;
   !terceiroComando;.
```

Figura 13. Exemplo de plano usando a produção `plan_body` original.

Contudo, para evitar problemas de conflitos optou-se por tornar o "`;`" obrigatório entre os termos do corpo, com exceção dos comandos `if | for | while`. Além disso, o "`;`" aparece apenas separando os termos, logo o último termo não possui o "`;`" apenas o "`.`". A nova versão da produção `plan_body` pode ser vista na Figura 10. Abaixo segue um exemplo utilizando a nova estrutura:

```
+!plano
   <-
   !primeiroComando;
   ?segundoComando;
   !terceiroComando.
```

Figura 14. Exemplo de plano usando a produção `plan_body` modificada.

6. Considerações finais

Este trabalho teve como objetivo inicial a análise e a refatoração do *parser* da linguagem Jason. Para realizar a análise, foi necessária uma ampla compreensão acerca das estruturas da linguagem e a identificação dos problemas teve como ponto de partida os conflitos existentes na gramática original. Dos 12 conflitos identificados e investigados, os 9 principais foram mitigados considerando três pilares: simplicidade, eficácia e especificidade. Outros 3 não foram tratados pois o primeiro caminho de expansão possível da produção é

sempre o correto. Numa segunda etapa, foram identificados 6 conflitos, 3 dos quais foram tratatos e os outros 3 são os mesmos citados anteriormente. Além dos conflitos, também foram reestruturadas diversas produções, visando atender aos critérios supracitados.

A nova gramática elaborada, foi validada através de testes disponíveis no próprio Jason de forma a garantir que o resultado final fosse uma versão do Jason que suportasse grande parte dos códigos já desenvolvidos pela comunidade de usuários.

As mudanças realizadas na gramática, em sua grande maioria, não implicam mudanças no funcionamento externo do Jason. Contudo, algumas delas realmente limitam o que pode ser escrito em códigos Jason na prática. No entanto, essas mudanças foram pensadas de forma a tornar a linguagem mais específica e organizada.

De forma geral, é possível afirmar que as mudanças realizadas impactaram positivamente a linguagem, pois sua estrutura ficou mais organizada, legível e específica e, em alguns casos, mais eficaz que a estrutura original, pois algumas mudanças resolveram problemas de corretude existentes na estrutura original. Assim a nova gramática facilitará futuras manutenções e expansões, assegurando que o Jason possa continuar sendo uma importante linguagem no desenvolvimento de sistemas multiagentes.

A nova versão do Jason, contendo as alterações realizadas está disponível no repositório `http://jason.sourceforge.net`.

Referências

Aho, A., Lam, M., Sethi, R., and Ullman, J. (2008). *Compiladores - Princípios, técnicas e ferramentas*. Pearson Addison-Wesley, 2nd edition.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons.

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.

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.

Integrating neural networks into the agent's decision-making: A Systematic Literature Mapping

Rodrigo Rodrigues, Ricardo Azambuja Silveira, Rafael de Santiago

[1]Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brazil

rodrigo.mello@posgrad.ufsc.br, ricardo.silveira@ufsc.br, r.santiago@ufsc.br

Abstract. *AI systems have been playing a crucial role in many different fields of study. Even though connectionist methods, more precisely deep neural networks, are more prevalent nowadays, many of their limitations have delayed the deployment of AI systems in relevant areas, such as healthcare, financial, and legal. One of its main criticisms relies on the fact that deep neural networks require large data sets, poor generalization, and lack of interpretability. Researchers believe that the next level of AI will require integrating these connectionist methods with different AI's fields. Although many different studies explore this research topic, many of them are surveys or do not cover AI's new advances. A Systematic Literature Mapping is performed to fill this gap, which aims to explore the integration of neural networks into the intelligent agent's decision making. In this study, we analyzed over 1000 papers, and the main findings are: (i) 64% of studies use neural networks to define the learning agent's reward policies; (ii) 5% of studies explore the integration of neural networks as part of the agent's reasoning cycle; and (iii) although 55% of studies main contributions are related to neural networks and agents design, we find that the remaining 45% of the studies use both agents and neural networks to solve or contribute to a particular field of study or application.*

1. Introduction

When decisions derived from intelligent systems ultimately affect humans lives (e.g. medicine, law or legal), there is an emerging need for understanding how AI methods execute these decisions [Goodman and Flaxman 2017, Arrieta et al. 2019]. Even though connectionist techniques are more precise, these methods result in opaque and hard to interpret systems. Since Deep Neural Networks (DNN) now represents the flagship in AI, it is crucial to establish its main limitations. Most of the criticism revolves around data inefficiency, poor generalization, and lack of interpretability [Garnelo and Shanahan 2019a]. In a symbolic approach, we have an easily understandable and transparent system. However, they are known as less efficient [Arrieta et al. 2019, Anjomshoae et al. 2019].

Considering the benefits that both methods bring to AI, many studies have been focusing on combining connectionist and symbolic approaches. The main goal is to increase intelligent systems expressiveness, trust, and efficiency [Arrieta et al. 2019, Bennetot et al. 2019, Garnelo et al. 2016, Marra et al. 2019, Garcez et al. 2019]. The literature presents many works that review the usage of both techniques. Before our study, we find that different parts of [Jedrzejowicz 2011, Garnelo and Shanahan 2019b, Rizk et al. 2018] works are similar to ours, although most of these works are

surveys and do not present a systematic review with a well-defined protocol. [Garnelo and Shanahan 2019a] present compelling arguments about the necessity to integrate symbolic and DNN. However, [Garnelo and Shanahan 2019a] do not present a systematic literature review, and its work focus on object representation and compositionality and how they can be accommodated in a deep learning framework. [Rizk et al. 2018] present a survey about how reinforcement learning, dynamic programming, evolutionary computing, and neural networks can be used to design algorithms for MAS decision-making. [Jedrzejowicz 2011] also explores the integration of machine learning and agents. However, we believe that it is required to revisit the last five years of advances in AI.

This step of the research followed the guideline presented in [Kitchenham and Charters 2007] to execute a Systematic Literature Mapping (SLM). We analyzed 1019 papers from Scopus and ACM, and 110 papers remained after applying the inclusion and exclusion criteria. We compiled them to answer the following research questions: (i) which class of agents and neural networks architecture are being employed; (ii) how these studies combine neural network and agents; and (iii) which scenarios are these intelligent systems being deployed.

This paper is organized as follows. Section 2 presents a short review about intelligent agents and Artificial Neural Networks (ANN). Section 3 presents the protocol used to execute this systematic literature mapping. In section 4, we present the main findings of this systematic literature mapping. In section 5 we present the conclusion and future works.

2. Background

In this section will be briefly presented a short review of intelligent agents and ANN.

2.1. Intelligent agents

Despite the existence of different definitions about intelligent agents, in this study, we assume that an agent has specific properties, such as autonomy, social skills, reactive, and proactive [Wooldridge et al. 1995]. Based on these properties, we use [Russell and Norvig 2002] agents classification, which consists of the following types:

- simple reflex agent: performs actions based on the current state of the world, which can map to conditions-actions rules;
- model-based reflex agent: models internal states that can be used during decision-making;
- goal-based agent: defines goals based on a desirable state that it wants to achieve;
- utility-based agent: uses a function that maps a state or a sequence of states to a real number, which defines preferences between different states;
- learning agent: can improve its decision-making by using learning capabilities, which can be improved based on past experiences.

A multi-agent system consists of agents that interact by using a protocol to communicate with each other. Usually, agents represent different people or entities, which each of them could have different goals and motivations [Wooldridge 2009].

2.2. Artificial Neural Network

Neural networks are models inspired by the structure of the brain [Ozaki 2020, McCulloch and Pitts 1990], which provides a mechanism for learning, memorization and generalization. These models can differ not only by their weights and activation function but also in their structures, such as the feed-forward NN that are known for being acyclic, while recurrent NN has cycles [Ozaki 2020]. An ANN consists of different neuron layers, where input layers form the NN, one or more hidden layers, and an output layer [Wang 2003]. Definition 1 is presented in [Kriesel 2007] and models a simple neural network.

Definition 1 *An NN is a sorted triple* (N, V, w) *with two sets N, V and a function* w*, where N is the set of neurons and V a set* $\{(i,j)|i,j \in \mathbb{N}\}$ *whose elements are called connections between neuron* i *and neuron* j*. The function* $w : V \rightarrow \mathbb{R}$ *defines the weights, where* $w((i,j))$*, the weight of the connection between neuron* i *and neuron* j*, is shortened to* w_{ij}*.*

3. Systematic Literature Mapping protocol

We follow [Kitchenham and Charters 2007] work as a guideline to perform this Systematic Literature Mapping (SLM). An SLM differs from a Systematic Literature Review (SLR) mainly because it presents a broader overview about a field of study, establishes the existence of research evidence, and provides an indication of the number of evidence [Kitchenham and Charters 2007].

According to [Kitchenham and Charters 2007], a systematic literature review or mapping involves several discrete activities. Three main phases with different tasks can divide this process. The phases and tasks that we executed are the following:

- Planning: identification of the need for a review, specifying the research question(s), developing a review protocol, evaluating the review protocol;
- Conducting: identification of research, selection of primary studies, study quality assessment, data extraction, and data synthesis;
- Reporting: formatting the main report and evaluating the report.

3.1. Research questions

In some studies, defining research questions can involve different components and properties. To assist us during this step, we employed the five criteria Population, Intervention, Comparison, Outcomes, and Context (PICOC) presented in [Petticrew and Roberts 2008]. Since our research questions explore the combination of two different approaches, It is worth mentioning that we did not use the comparison criteria of the PICOC method in our study. The main reason for this decision is that our initial research focused on investigating how the integration between agents and neural networks occurs. The PICOC criteria, its definitions, and how it relates with our research are the following:

- P (population or problem): intelligent agents and their different classes;
- I (intervention or interest): which neural networks architecture are employed;
- O (Outcome/results): main contributions achieved by the system originated by combining neural networks and intelligent agents;

- C (Context): scenarios in which the proposed approach was used.

The research questions are defined as follows:

- RQ1: Which class of agents are being used?
- RQ2: Which architectures of neural networks are being used?
- RQ3: How do these works combine neural networks and agents?
- RQ4: Which scenarios are these intelligent systems being deployed?
- RQ5: Do these works contributions focused on improving neural networks, intelligent agents, or both fields?

3.2. Search string

In this work, we decided to use SCOPUS and ACM databases. Since the main goal of this work is to study and analyze the integration between connectionist methods and intelligent agents, the search strings executed in SCOPUS and ACM are the following:

- Scopus: (”deep learning” OR ”neural network”) AND (”intelligent agent” OR ”autonomous agent”);
- ACM: Title:(((”deep learning” OR ”neural network”) AND (”intelligent agent” OR ”autonomous agent”))) OR Abstract:(((”deep learning” OR ”neural network”) AND (”intelligent agent” OR ”autonomous agent”))) OR Keyword:(((”deep learning” OR ”neural network”) AND (”intelligent agent” OR ”autonomous agent”)));

Since it is common to perform an initial search with different strings, we noticed that some works use the term ‘deep learning’ to refer to neural networks during one of these searches. Taking that into consideration, we added this term in our final search string.

Table 1 presents the inclusion and exclusion criteria used to filter the relevant studies in our SLM. As previously mentioned, [Jedrzejowicz 2011] also explores the integration of machine learning and agents development. However, this work did not explore the last five years of advances in AI. Considering that, we believe that it is required to revisit the last five years of AI contributions.

Table 1. Inclusion and exclusion criteria

Inclusion (I)	Exclusion (E)
published between 2015 to 2021	published before 2015
written in English	not written in English
available to download	unavailable to be read
combines neural network and intelligent agent to build an intelligent system	does not use intelligent agents
present a qualitative or quantitative evaluation	does not use the neural network
published in conference or journal	do not present quantitative or qualitative evaluation
primary studies	secondary or tertiary studies

3.3. Selection process

Figure 1 presents the steps performed during the selection, data extraction, and data analyses. Each step contains the number of papers that were selected for the next step. It is important to remark that even after step 4 when inclusion and exclusion criteria were applied, some of the papers did not fit those criteria; therefore, they were removed before the data extraction step. We noticed that some papers were not available to download, which caused a reduction in the number of papers used in the data extraction step. We considered the unavailability of these papers as one of the validity threats. Taking that into consideration, the final number of analyzed papers was 110.

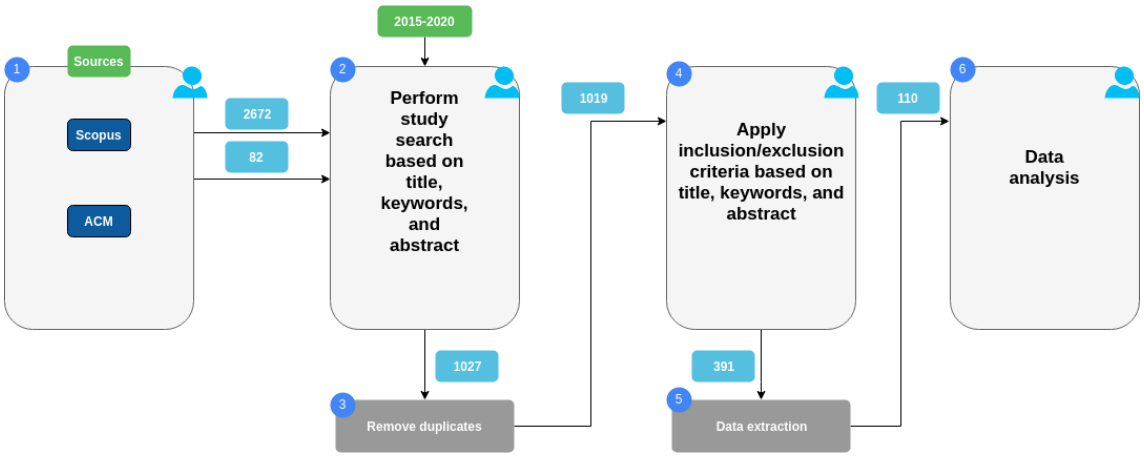


Figure 1. Steps executed during systematic literature mapping.

3.4. Data extraction

The fields and their definitions used during data extraction are the following:

1. RQ1 - Agents class: since an agent terminology and its architectures vary across different works and fields, we chose to use the classification presented in [Russell and Norvig 2002], which defines the following agent's types: (i) simple reflex agents; (ii) model-based reflex agents; (iii) goal-based agents; (iv) utility-based agents; and (v) learning agents.
2. RQ2 - Neural networks types: this field provides information about the type of neural network used;
3. RQ3 - Integration: the primary goal of this field is to study how different works combine neural networks and agents during AI systems development;
4. RQ4 - Contributions: this field identifies what is the main contribution resulted from the combination of neural network and agents;
5. RQ5 - Scenario: this field intends to report where the proposed agent was or intended to be deployed and whether it exists a concern about using these approaches to assist in real-world problems resolution.

   To access the Data extraction form, the reader could click here.

3.5. Validity threats

According to Figure 1, the selection and data extraction were executed by one researcher. This decision is the one that represents more risks to our study and originates the following threats:

1. Researcher expertise: since the steps of studies selection and data extraction were executed only by one researcher that has a background in intelligent agent, some of the relevant features of the neural network could be ignored or wrongfully reported;
2. Data aggregation: based on what is presented in section 4, to answer some research questions were necessary to define groups of agents and the employed neural networks. In this sense, the interpretation of the main findings could present imprecision and limitations;
3. Unavailable papers: we noticed that some papers from relevant conferences and journals were not available to download in our institution, which limited our SLM results.

To mitigate the problems previously presented, we intend to involve two other researchers to revise the data extraction fields. Both researchers are neural network specialists, which enables us to improve the data analyses quality.

4. Results from the data analyses

In this section, we report the most important findings we gathered during the data analyses step. The analyses method and the results employed in our work is called thematic. This method goal is to describe and present an overview of existing works [Dixon-Woods et al. 2006]. The decision to use this approach is supported by the fact that this work is a systematic literature mapping and does not require a qualitative analysis.

We start by showing in subsection 4.1 the retrieved studies distribution. In subsections 4.2, 4.3, and 4.4 we discuss the first three research questions, which are related to intelligent agent's different groups, neural networks architectures, and the combination of neural networks and intelligent agents. Subsection 4.5 presents the main findings of the two remaining research questions.

4.1. Studies distribution between 2015 and 2020

Figure 2 shows the interest in intelligent agents and neural networks in the last five years. The interest in this field starts increasing in 2017, in which the amount of works between 2018 and 2020 represents 59%. It is also worth mentioning that this search occurred on 05/04/2020; therefore, it does not include 2020 in its totality.

4.2. RQ1 - Intelligent agents groups

Figure 3 shows the distribution of different agents returned after the data extraction step. Analyzing Figure 3 it is possible to observe that:

- Reinforcement learning agents usage. Following the results obtained by employing DNN, it is noticeable the usage of reinforcement learning agents. This result could be explained by employing DNN in the definition of reward policies, which represented the main limitation of reinforcement learning. For instance, in [Mnih et al. 2013, Mnih et al. 2015] were achieved relevant results using DNN and reinforcement learning agents.

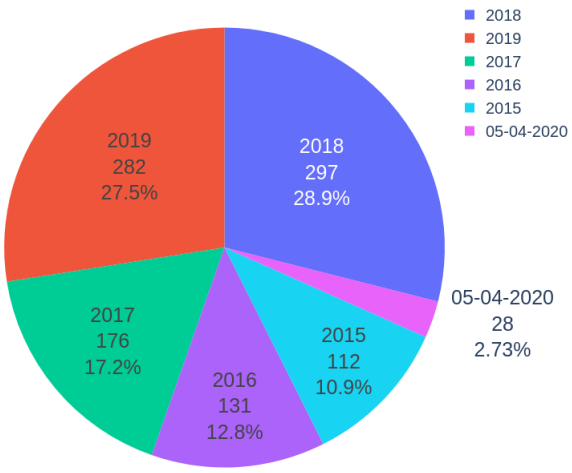


Figure 2. Studies distribution returned from 2015 to 2020.

- Simple-reflex agents. Being one of the most explored types of agents, it is still relevant to point its usage. One of the main reasons is that it is straightforward to combine this type of agent with other techniques since most of the time, the chosen technique acts as decision-making, and the agent only possesses sensors and actuators.

Even though we did not fully explore the Multi-Agent System (MAS) usage, we noticed a relevant increase in combining MAS with reinforcement learning and DNN. This approach points towards a direction where agents could use different policies to coordinate their actions.

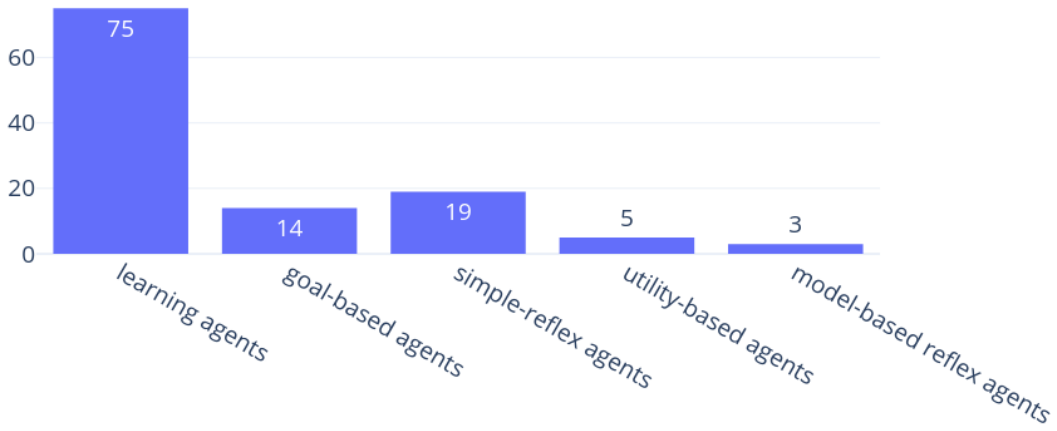


Figure 3. Agents class distribution during the period of 2015 to 2020.

4.3. RQ2 - Neural networks

Following the same approach explored in Figure 3, in Figure 4 we present the most used neural networks architectures during data extraction. Another relevant piece of information is that some works did not accurately report the neural network used. Considering that, we removed the works that did not present information. Since some of the works use more than one type of neural network, the total amount of neural network differs from the number of analyzed papers.

Different from Figure 3, in Figure 4 there is a wider usage of different neural networks. However, convolution, feed-forward, and recurrent neural networks were more frequent. These numbers also agree with the approach used in [Mnih et al. 2013, Mnih et al. 2015], in which these works outperformed all previous approaches on different games and surpass a human expert.

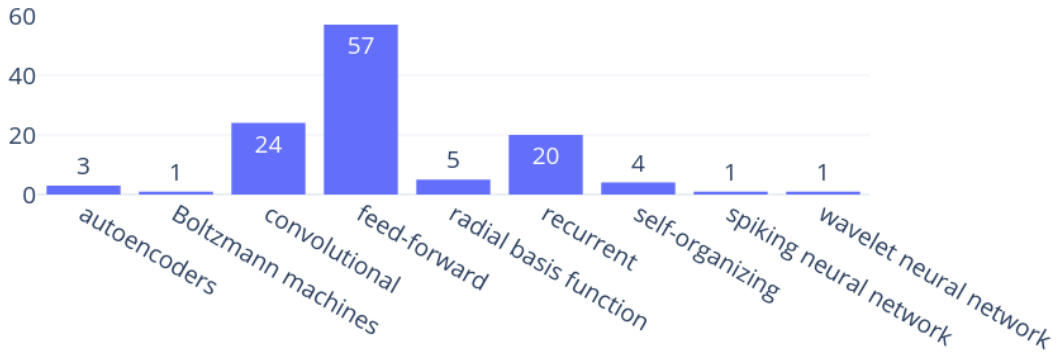


Figure 4. Neural networks architecture distribution during the period of 2015 to 2020.

To answer the remaining research questions, we use a scope analyses approach. A scoping analysis represents a flexible way of providing a broader view of the selected researches, which fits the primary goal of a Systematic Literature Mapping [Dixon-Woods et al. 2006].

4.4. RQ3 - Combination of neural networks and agents

Figure 5 summarizes how studies combine different neural networks architectures and intelligent agents. Two of our study's most relevant findings could be explained through Figure 5. The first one is using feed-forward, convolution, and recurrent neural networks as a mechanism to define reward policies for learning agents, representing 64% of analyzed studies. The second one is that only 5% of studies use the neural networks as an input or combines with the agent's reasoning. This decision could be linked with the implication of combining neural networks in these steps, which requires dealing with many different fields, such as information fusion, knowledge consistency, and planning, for instance.

One of the main reasons for the numbers presented in Figure 5 could be explained by using neural networks to define actions or policies previously explored in the literature

and do not require to change the agent's reasoning cycle during decision-making. In this sense, using the neural network as input or part of the decision-making process could require a more robust implementation of these agents.

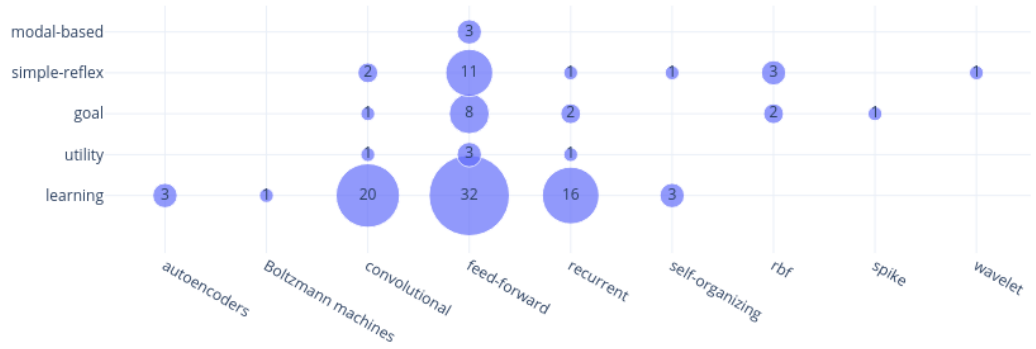


Figure 5. Studies distribution on the usage of different neural networks and intelligent agents.

4.5. RQ4 - Scenarios and RQ5 - Contributions

Even though the spam of contribution from the different works varies, it is possible to define in which group these works focus their contributions. In our study, we define the following group:

- intelligent agents contribution;
- neural networks contribution;
- intelligent agents and neural network contributions;
- application area contribution, in which it was achieved by designing an intelligent system to solve or assist a task resolution in a field of study.

Figure 6 presents the findings related to these contributions groups. Even though 55% of the analyzed studied are contributing to intelligent agents and neural networks design, the remaining 45% represents that combining the existent approaches of NN and agents enables solving problems of different fields of study.

Based on what was obtained during the data extraction phase, the range of scenarios used in the different works varied during this study execution. However, it is possible to define which scenario is more frequent. Many studies contributed by applying agents and neural networks to assist during problem resolution or simulations that model real-world scenarios. It was simulated some behaviour or situation. As presented in [Lamouik et al. 2017, Chen et al. 2019, Loumiotis et al. 2018, Garg et al. 2019, Amrani et al. 2019, Klose and Mester 2019, Kotyan et al. 2019] different studies built systems able to assist humans during task resolution. Some studies showed an agent responsible for driving a car or controlling a traffic light signal autonomously.

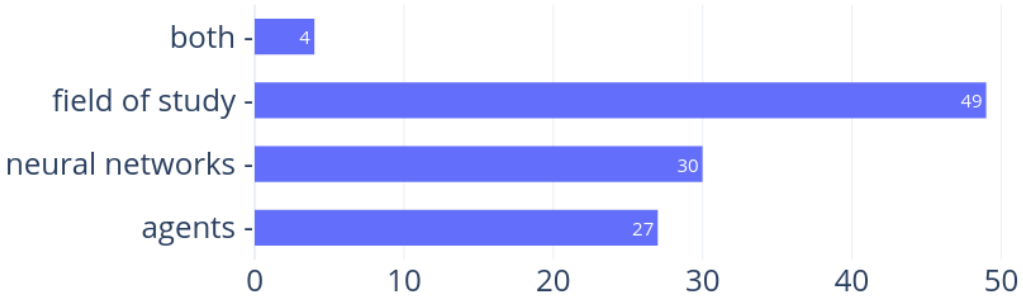


Figure 6. Distribution of contribution between the period of 2015 to 2020.

5. Conclusion and future works

In this paper, we presented a systematic literature mapping, where the main goal was to report an overview of the integration of neural networks into the agent's decision-making. To achieve our goal, we define several research questions related to the type of agents and neural networks employed, which step of the decision-making a neural network was used, the main contributions of these studies, and the scenario in which these systems were deployed. The amount of 1019 papers from 2015 to 2020 shows the relevance of the field explored. The studies from 2018, 2019, and 2020 were responsible for 73,76% of the works used in our systematic literature mapping, showing the field's growth after 2017.

One of the most important findings of our SLM shows that few studies explore the integration of neural networks as part of the agent's decision-making. Most of the studies use neural networks to define learning agents reward policies. Even though these approaches provide significant results, these systems have been suffering from a lack of transparency and require a considerable amount of data [Adadi and Berrada 2018, Arrieta et al. 2019]. This criticism also limits the field of study that an AI system can be deployed, such as in health care, finance, and legal [Garnelo and Shanahan 2019a]. Although many studies contributed to neural networks and agents design, several studies use both agents and neural networks to solve or contribute to a particular study field.

A promising path towards integrating neural networks into the agent's reasoning cycle can be achieved by considering the neural-symbolic field. The neural-symbolic field provides the effective integration of connectionist and symbolic methods, more precisely learning and reasoning [Parisotto et al. 2016]. Neural-symbolic can be employed where large amounts of heterogeneous data exist, and knowledge descriptions are required [Garcez et al. 2015].

As future work, we can explore two different paths: (i) - increase this systematic literature mapping confiability by applying our search string in different digital libraries; (ii) - perform a systematic literature review, using more specific search strings, including multiagent systems and neural-symbolic, and employ quality assessments techniques.

References

Adadi, A. and Berrada, M. (2018). Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, 6:52138–52160.

Amrani, N., Abra, O., Youssfi, M., and Bouattane, O. (2019). A new interpretation technique of traffic signs, based on deep learning and semantic web. cited By 0.

Anjomshoae, S., Najjar, A., Calvaresi, D., and Främling, K. (2019). Explainable agents and robots: Results from a systematic literature review. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1078–1088. International Foundation for Autonomous Agents and Multiagent Systems.

Arrieta, A. B., Díaz-Rodríguez, N., Ser, J. D., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., Chatila, R., and Herrera, F. (2019). Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai.

Bennetot, A., Laurent, J.-L., Chatila, R., and Díaz-Rodríguez, N. (2019). Towards explainable neural-symbolic visual reasoning.

Chen, I.-M., Zhao, C., and Chan, C.-Y. (2019). A deep reinforcement learning-based approach to intelligent powertrain control for automated vehicles. cited By 0.

Dixon-Woods, M., Cavers, D., Agarwal, S., Annandale, E., Arthur, A., Harvey, J., Hsu, R., Katbamna, S., Olsen, R., Smith, L., et al. (2006). Conducting a critical interpretive synthesis of the literature on access to healthcare by vulnerable groups. *BMC medical research methodology*, 6(1):1–13.

Garcez, A., Besold, T. R., Raedt, L., Foldiak, P., Hitzler, P., Icard, T., Kuhnberger, K.-U., Lamb, L. C., Miikkulainen, R., and Silver, D. L. (2015). Neural-symbolic learning and reasoning: contributions and challenges.

Garcez, A. d., Gori, M., Lamb, L. C., Serafini, L., Spranger, M., and Tran, S. N. (2019). Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. *arXiv preprint arXiv:1905.06088*.

Garg, D., Chli, M., and Vogiatzis, G. (2019). A deep reinforcement learning agent for traffic intersection control optimization. cited By 0.

Garnelo, M., Arulkumaran, K., and Shanahan, M. (2016). Towards deep symbolic reinforcement learning. *arXiv preprint arXiv:1609.05518*.

Garnelo, M. and Shanahan, M. (2019a). Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. *Current Opinion in Behavioral Sciences*, 29:17–23.

Garnelo, M. and Shanahan, M. (2019b). Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. *Current Opinion in Behavioral Sciences*, 29:17 – 23. SI: 29: Artificial Intelligence (2019).

Goodman, B. and Flaxman, S. (2017). European union regulations on algorithmic decision-making and a "right to explanation". *AI Magazine*, 38(3):50–57.

Jedrzejowicz, P. (2011). Machine learning and agents. In *KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*, pages 2–15. Springer.

Kitchenham, B. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering.

Klose, P. and Mester, R. (2019). Simulated autonomous driving in a realistic driving environment using deep reinforcement learning and a deterministic finite state machine. In *Proceedings of the 2nd International Conference on Applications of Intelligent Systems*, pages 1–6.

Kotyan, S., Vargas, D., and Venkanna, U. (2019). Self training autonomous driving agent. cited By 0.

Kriesel, D. (2007). A brief introduction on neural networks.

Lamouik, I., Yahyaouy, A., and Sabri, M. (2017). Smart multi-agent traffic coordinator for autonomous vehicles at intersections. cited By 6.

Loumiotis, I., Demestichas, K., Adamopoulou, E., Kosmides, P., Asthenopoulos, V., and Sykas, E. (2018). Road traffic prediction using artificial neural networks. cited By 3.

Marra, G., Giannini, F., Diligenti, M., and Gori, M. (2019). Integrating learning and reasoning with deep logic models. *arXiv preprint arXiv:1901.04195*.

McCulloch, W. S. and Pitts, W. (1990). A logical calculus of the ideas immanent in nervous activity. *Bulletin of mathematical biology*, 52(1-2):99–115.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Ozaki, A. (2020). Learning description logic ontologies: Five approaches. where do they stand? *KI-Künstliche Intelligenz*, 34(3):317–327.

Parisotto, E., Mohamed, A.-r., Singh, R., Li, L., Zhou, D., and Kohli, P. (2016). Neuro-symbolic program synthesis. *arXiv preprint arXiv:1611.01855*.

Petticrew, M. and Roberts, H. (2008). *Systematic reviews in the social sciences: A practical guide*. John Wiley & Sons.

Rizk, Y., Awad, M., and Tunstel, E. (2018). Decision making in multiagent systems: A survey. *IEEE Transactions on Cognitive and Developmental Systems*, 10(3):514–529. cited By 10.

Russell, S. and Norvig, P. (2002). Artificial intelligence: a modern approach.

Wang, S.-C. (2003). *Artificial Neural Network*, pages 81–100. Springer US, Boston, MA.

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

Wooldridge, M., Jennings, N. R., et al. (1995). Intelligent agents: Theory and practice. *Knowledge engineering review*, 10(2):115–152.

Proposta de arquitetura que integre SMA, Memória e Emoções

Thiago Dantas[1], **Patrícia Padula**[1], **Diana F. Adamatti**[1], **Cleo Z. Billa**[1]

[1]Programa de Pós-Graducação em Computação
Universidade Federal do Rio Grande (FURG)

{thiagodantas923,padulalopes,dianaada,cleobilaa}@gmail.com

Resumo. *Este artigo apresenta uma arquitetura para sistemas multiagentes (SMA) que integra o uso de emoções e memória para o estudo das interações entre agentes. No sistema desenvolvido, foi utilizado um mapeamento das 22 emoções do modelo OCC, utilizando o Watson e o Jason através de um sistema de transferência de mensagens que utiliza uma automação desenvolvida com Selenium. O caso de testes utilizado para avaliar este trabalho baseia-se em um problema relacionado a teoria dos jogos e é conhecido como o "Dilema do Prisioneiro". Como resultados, observa-se que as possibilidade de interações entre os agentes podem ser infinitas, fazendo com que seja possível realizar* n *simulações entre SMAs que envolvam memória e emoções.*

1. Introdução

Este trabalho está focado no desenvolvimento de um sistema para identificar emoções definidas pelo modelo OCC (sigla de seus criadores Ortony, Clore e Collins) [Ortony et al. 1990] através do processamento de linguagem natural. Nosso objetivo é tornar os agentes cognitivos capazes de identificar emoções expressas em mensagens para auxiliar na sua tomada de decisão. O projeto desenvolvido visa avaliar as emoções expressas entre os agentes, bem como responder às suas mensagens utilizando as emoções captadas, dando o resultado desejado, através da geração da emoção presente nas interações agente-agente.

O objetivo maior é auxiliar na compreensão das interações que ocorrem entre os agentes, uma vez que os agentes ao reconhecerem uma determinada emoção, realizam suas ações influenciados por essa emoção.

A emoção pode ser definida como um estado complexo de sentimento que resulta em mudanças físicas e psicológicas que influenciam o pensamento e o comportamento. Em outras palavras, emoções são sentimentos que alteram diretamente o processo de tomada de decisão do agente. Um dos modelos mais usados para representar emoções é o modelo OCC [Ortony et al. 1990]. O Modelo OCC fornece informações que permitem uma interpretação de uma situação para um agente e a que emoção essa interpretação nos leva.

Para poder interpretar as interações que ocorrem entre os agentes, é utilizada uma ferramenta que pode realizar o processamento de linguagem natural (PNL) e a utiliza para identificar emoções, utilizando os parâmetros apresentados no Modelo OCC. A ferramenta que usamos foi o sistema de computação cognitiva da IBM, denominado Watson[1].

O funcionamento do modelo desenvolvido ocorre basicamente da seguinte maneira: uma mensagem é enviada de um agente para outro, a qual contém alguns *tokens*. Estes *tokens* são então

[1]Disponível em https://www.ibm.com/Watson/br-pt/

capturados e analisados na base de perguntas e respostas do Watson. Essa base determina qual a emoção identificada, tendo como base as 22 emoções presentes no Modelo OCC.

Na arquitetura proposta, o Watson é responsável apenas pela identificação dos *tokens* presentes nas mensagens trocadas. Desta forma, pode ser substituído por uma ferramenta semelhante caso seja de interesse do desenvolvedor.

Existem dois tipos possíveis de mensagens a serem interpretadas pelo Watson:

- Declaração com emoção incluída: mensagens que contém pelo menos um símbolo em sua estrutura. Por exemplo:
  - Estou ressentido com ele.
  - Estamos eufóricos com a viagem.
  - Você é uma pessoa amarga.

  Nestas frases, `ressentido`, `eufórico` e `amarga` São identificadas como tokens.
- Declaração sem emoções incluídas: estas são geralmente respostas diretas monossilábicas. Por exemplo:
  - Sim.
  - Não.

  No entanto, também podem ser sentenças mais longas, sem expressão emocional. Por exemplo:
  - Eu vou ao mercado.
  - O filho dos meus vizinhos nasceu esta noite.
  - Ele está construindo sua casa.

Nenhuma dessas declarações contém *tokens* que expressam emoção. Portanto, o Watson não transmitirá ao agente receptor uma emoção expressa pelo agente emissor.

E cada agente conta com uma memória para lembrar se determinado tópico é útil ou não ao se comunicar com o outro agente.

Para um melhor entendimento do funcionamento do sistema é detalhado por meio da Figura 1 a ordem de processamento em relação a comunicação entre os agentes A e B no sistema:

1. O agente A envia uma mensagem para o Agente B.
2. Esta mensagem é processada pelo Watson, que busca *tokens* e ao encontrar “apavorado” identifica a emoção expressa como medo.
3. O Watson passa a informação de que o agente A está expressando medo ao Agente B.
4. O agente B responde ao agente A com objetivo de acalmá-lo.
5. O Watson processa a mensagem do agente B (*”Você estudou e precisava de pouca nota, tenho certeza que vai ser aprovado.”*) e não encontra *token* o que indica que nenhuma emoção foi expressa.

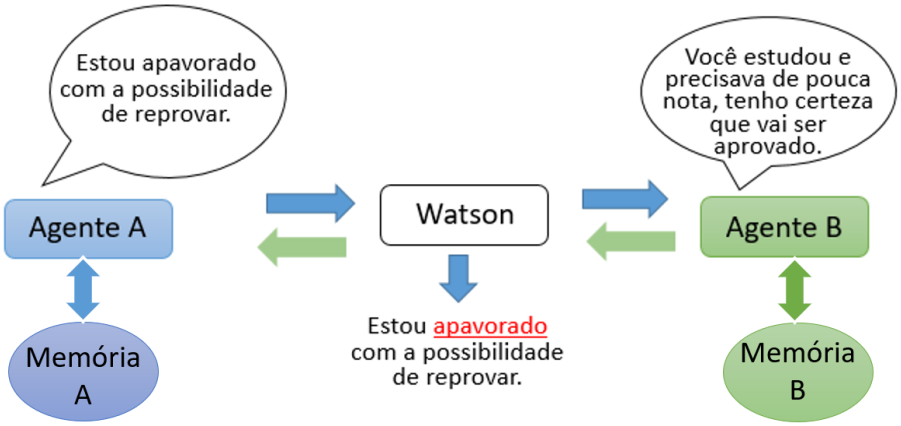


Figura 1. Troca de mensagens entre agentes A e B e identificação de *tokens* pelo Watson.

Após a modelagem do sistema e a definição dos *tokens* referentes a cada uma das 22 emoções, foi realizada a implementação no Watson. Esta implementação foi realizada utilizando o módulo do Watson *Assistant*, que normalmente é utilizado em *chat bots*. Porém, neste projeto ele foi adaptado para identificar as emoções presentes no modelo OCC, sendo necessário usar os conceitos de intenção *intent* e Entidade *entity*:

- *Intent*: é a representação do propósito que o usuário inseriu na mensagem, uma intenção é definida para cada tipo de chamada que a aplicação deseja atender. Neste projeto existe uma *intent* para cada emoção.
- *Entity*: representa o termo ou o objeto que é relevante para a intenção e provêm o contexto específico para a intenção. Neste projeto, as entidades são os *tokens* de cada emoção.

A implementação do sistema de emoções e posteriormente, a memória vão afetar de forma expressiva o *reasonyng cycle* do *AgentSpeak-L* que consiste em uma série de etapas de execução que começam a ser executadas até que o agente interrompa seu raciocínio, pois ao utilizar as emoções e lembrar de eventos passados, a base de crenças do agente será alterada, as mensagens que serão selecionadas para comunicação dependerão da emoção, da memória ou de ambos e o plano a ser aplicado em determinada situação será afetado por esses dois fatores.

Foram implementados, primeiramente três agentes que interagiam entre si a partir de perguntas e respostas, porém a conclusão dessa interação era gerada randomicamente pelo Jason para servir como base de comparação para os outros sistemas, pois não haviam emoções e memória implementadas para este sistema. Posteriormente, o sistema já contava com a implementação de emoções e continha três elementos principais: a IDE de automação em Java Selenium, o Watson e a plataforma de multiagente Jason, onde a partir do Selenium era realizada a transferência das mensagens entre o Jason e o Watson através de uma automação implementada.

2. Emoções e Modelo OCC

Existem várias propostas para simular e identificar emoções para tornar possível a simulação computacional das mesmas [Marsella et al. 2010]. No contexto da Inteligência Artificial (IA), esta área de pesquisa contém vários tipos de abordagens que vão desde o Social, passando pelos processos

Cognitivos, até os Biológicos [Aboulafia and Bannon 2004]. É um tema muito complexo e objeto de pesquisas constantes, que tem servido de fonte de inspiração para novas abordagens no desenvolvimento de Agentes Inteligentes [Arraes 2012].

Um dos modelos mais utilizados é o modelo OCC (sigla dos seus criadores Ortony, Clore e Collins) [Ortony et al. 1990] devido à sua simplicidade e capacidade de identificar diferentes emoções, baseado no contraste entre emoções positivas e negativas. Esta abordagem é útil em simulações onde as emoções são de grande importância, como a tomada de decisão [Marsella et al. 2010].

Ao classificar as emoções em 22 tipos diferentes, os obstáculos de complexidade inerentes à simulação de emoções são simplificados pelo modelo OCC, conforme apresentado na Tabela 1.

Além dessas classificações, cada emoção possui uma especificação dividida em 3 partes (mostradas na Figura 2).

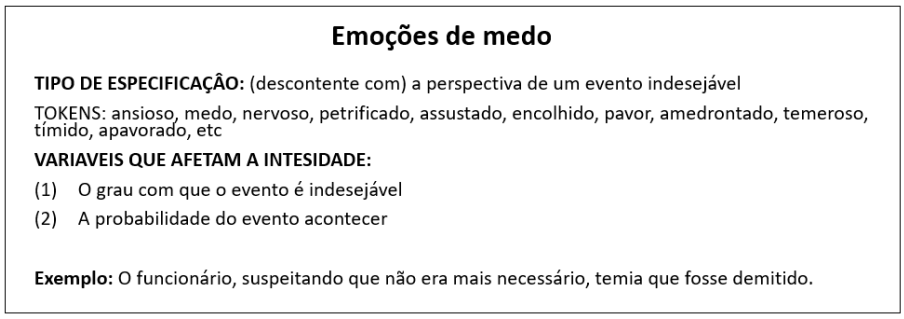


Figura 2. Especificação da Emoção MEDO em seus três elementos, adaptada de [Ortony et al. 1990]

- TIPO DE ESPECIFICAÇÃO: Descreve a condição que causa uma determinada emoção;
- *TOKENS*: Uma lista de tokens é apresentada especificando quais palavras indicam uma determinada emoção. Tão ansiosos, apavorados ou medrosos, são tipos de medo;
- VARIÁVEIS QUE AFETAM A INTENSIDADE: Cada emoção contém uma lista de variáveis que afetam sua intensidade. Essas variáveis afetam uma única emoção, variáveis que afetam várias não são consideradas. Quanto maior o número de variáveis, mais forte é a emoção.

Além disto pela semelhança com a proposta de arquitetura a ser desenvolvida neste trabalho, analisou-se o trabalho de [Sen et al. 2018], o qual, através do desenvolvimento de um jogo com agentes, denominado *Game of Trust* (GoT), investiga como a experiência passada afeta a confiança humana e os níveis de esforço baseado em interações entre agentes. O conceito de confiança utilizado em Got foi utilizado na implementação do sistema.

3. Processamento de Linguagem Natural e WATSON

Atualmente, é notória a utilização de IA nos mais diversos segmentos da área de Tecnologia da Informação (TI). A IA visa tornar as máquinas e computadores capazes de imitar as ações humanas para descobrir soluções para problemas comuns [Makridakis 2017].

A IA usa métodos de aprendizagem para resolver problemas por meio do conhecimento adquirido. Para isso, o computador precisa ter algumas capacidades, como entender linguagem natural [Norvig and Russell 2014].

Tabela 1. As 22 emoções do modelo OCC, adaptada de [Ortony et al. 1990]

Emoções	Condições
Joy	(contente com) um evento desejável
Distress	(descontente com) um evento indesejável
Happy-for	(contente com) um evento que se presume desejável para outro indivíduo
pity	(descontente com) um evento que se presume indesejável para outro indivíduo
Gloating	(contente com) um evento que se presume indesejável para outro indivíduo
Resentment	(descontente com) um evento que se presume indesejável para outro indivíduo
Hope	(contente com) a perspectiva de um evento desejável
Fear	(descontente com) a perspectiva de um evento indesejável
Satisfaction	(contente com) a confirmação da perspectiva de um evento desejável
Fear-confirmed	(descontente com) a confirmação da perspectiva de um evento indesejável
Relief	(contente com) a não confirmação da perspectiva de um evento Indesejável
Disappointment	(desaprovação de) uma ação condenável do próprio indivíduo
Pride	(aprovação de) uma ação louvável de outro indivíduo
Shame	(desaprovação de) uma ação condenável do próprio indivíduo
Admiration	(aprovação de) uma ação louvável de outro indivíduo
Reproach	(desaprovação de) uma ação condenável de outro indivíduo
Gratification	(aprovação de) uma ação louvável do próprio indivíduo e (estar contente com) um evento desejável relacionado
Remorse	(desaprovação de) uma ação condenável do próprio indivíduo e (estar descontente com) um evento indesejável relacionado
Gratitude	(aprovação de) uma ação louvável de outro indivíduo e (estar contente com) um evento desejável relacionado
Anger	(desaprovação de) uma ação condenável de outro indivíduo e (estar descontente com) um evento indesejável relacionado
Love	(gostar de) um objeto atraente
Hate	(não gostar de) um objeto desagradável

A comunicação existente entre máquinas e humanos ocorre através da linguagem natural, o que permite aos computadores, com base em modelos estatísticos e na análise de padrões de comportamento linguístico, traduzir para a linguagem de máquina as frases que são informadas pelos seus utilizadores [da Silva 2006] .

O Watson é um dos sistemas de aprendizado de máquina que utiliza linguagem natural para manter a comunicação com os usuários, por meio da simulação de processamento linguístico realizado por humanos. Utiliza tecnologia de cognição para simular os processos realizados pela mente humana e assim aprender de acordo com a informação que é transmitida pelo usuário [High 2012].

Esse sistema se tornou tão avançado e famoso desde que foi introduzido em 2011, durante o programa americano de perguntas e respostas, chamado Jeopardy, que é uma competição de reconhecimento de palavras na televisão americana, onde o sistema se tornou campeão por ser capaz de processar a linguagem natural de forma muito eficiente e rápida, devido ao grande investimento em hardware poderoso [Ferrucci 2012].

O Watson é composto de serviços que possuem funções diferentes, incluindo conversão de voz em texto, conversão de documentos, conversão de texto em fala, pesquisa otimizada de documentos e reconhecimento de imagens. O sistema, disponível através da plataforma online da IBM, consiste em arquitetura, a qual, de acordo com [Glehn 2018], pode ser visto em camadas, conforme mostrado na Figura 3.

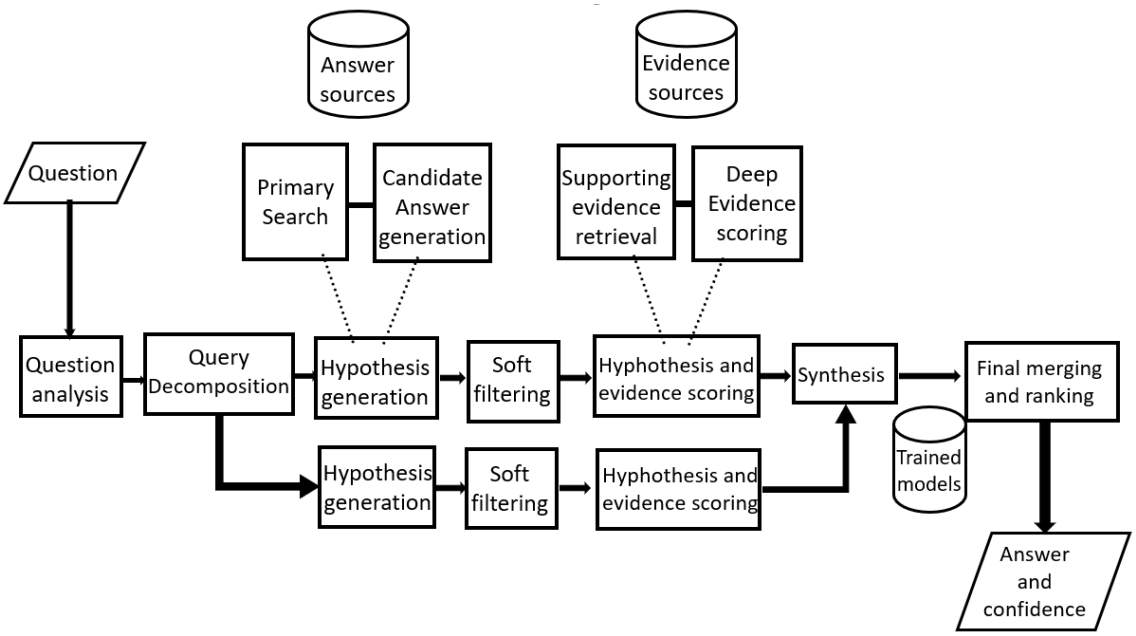


Figura 3. Arquitetura Watson, adaptada de [Ferrucci 2012].

Todas as etapas que ocorrem na Figura 3 exigem um processo de duas etapas:

1. Análise de sentenças: cujo conteúdo de cada pergunta é analisado.
2. Análise semântica e estatística: etapa de busca de sentenças e evidências para selecionar as respostas e consolidar um banco de dados.

Na etapa 1 da Arquitetura do Watson, quando um banco de dados é obtido, a questão é analisada e é possível identificar a qual tópico ele pode estar relacionado. Nesta primeira etapa,

são identificados alguns elementos, como: o assunto principal da pergunta, o tipo de pergunta solicitada, sua classificação e os elementos que necessitam de algum tratamento.

O próximo passo é decompor a pergunta, onde fatos independentes estão relacionados a outras perguntas, a fim de responder à pergunta principal feita. Na etapa 2, é realizada uma busca no banco de dados para gerar as hipóteses que atendem às respostas da pergunta, para isso, diversas técnicas de busca são utilizadas nesse processo. Para selecionar as melhores respostas, ocorre o que é chamado de pontuação de evidência. Essas respostas são refinadas no estágio `Synthesis` e as 100 principais respostas devem ser classificadas para o top 5. No final do processo, modelos predefinidos gerados durante o desenvolvimento da arquitetura são usados para obter o melhor resposta [Varga 2014].

4. Memória

Ao observar os seres vivos, principalmente os animais, percebe-se que eles geralmente interagem com o ambiente ao seu redor, com base em experiências passadas, aprendidas durante a vida. Esta observação conduz a acreditar que uma linha de pesquisa promissora consiste em analisar a natureza [Roisenberg et al. 1996]. Por exemplo, um animal, ao provar uma planta com gosto ruim poderá não mais comê-la novamente, pois adquiriu-se um determinado conhecimento sobre essa planta em específico, que será lembrado e utilizado de forma consciente, fazendo com que o animal venha lembrar-se do gosto da planta.

Uma outra situação pode ser observada analisando-se o comportamento dos ursos. Esses animais, comem mel devido ao seu gosto doce e valor nutritivo, mesmo que para isso recebam picadas de abelhas, pois o benefício compensa a dor [Spear 1973]. Na natureza, para comer, é preciso saber onde encontrar comida. E os mecanismos de memória são bons em lembrar a localização das refeições anteriores: quando um animal encontra alimento em um local específico, faz uma conexão mental entre o local e a comida.

No caso de alguns animais da savana africana, por exemplo, durante uma temporada de seca, os estudo científicos apontam para a chamada memória da espécie, ou seja, seguem uma determinada rota orientando-se por acidentes geográficos (rios, lagos, montanhas) perfeitamente conhecidos para regressarem a lugares que sejam seguros [Clayton et al. 2001].

É possível perceber com estes exemplos, que tanto as memórias boas como as memórias ruins afetam o comportamento dos animais e que o mesmo ocorre com os humanos [Smith et al. 1998]. Memórias positivas reforçam comportamento e memórias negativas enfraquecem comportamentos [Griffiths et al. 1999].

No entanto, o ser humano pode usar a memória de diferentes maneiras, podendo criar planos a longo prazo, aprender habilidades novas com base nas memórias adquiridas, utilizar algo aprendido em uma conversa ao falar com uma mesma pessoa no futuro, como chamá-la pelo nome ou falar de algo que ela gosta, evitando assuntos que ela não gosta [Mourão Júnior and Faria 2015].

Todas essas características fazem com que a memória seja essencial ao ser humano e, sua implementação em uma simulação que a represente de forma mais fidedigna à memória dos seres humanos ainda é alvo de várias pesquisas na área de inteligência artificial [Kintsch 2014]. Seguindo a proposta desse trabalho, ao utilizar-se da tecnologia de agentes, que pretendem recriar o comportamento humano, é necessário a presença do conceito de memória.

Neste sentido, cabe salientar que de acordo com [Cosenza and Guerra 2009] existem dois tipos principais de memória:

- Memória explícita: memória de curta duração ou curto prazo, conhecimento adquirido, lembrados e utilizados conscientemente. Um exemplo é a lembrança sobre o que jantamos ou o que vestimos de manhã.
- Memória implícita: memória de longa duração ou memória de longo prazo, que guarda as lembranças permanentes se manifesta sem interação consciente ou esforço. São exemplos de memória implícita a habilidade de escovar os dentes ou andar de bicicleta.

Os conceitos de memória explícita e implícita foram utilizados no desenvolvimento da arquitetura proposta, para simular memória de curto e longo prazo.

5. Implementação

Para a arquitetura proposta, foi necessário utilizar um conjunto de ferramentas para sua implementação.

Na primeira etapa, utilizou-se o Jason, e foi desenvolvido um SMA que não utilizava memória e nem emoções. Neste sistema, implementou-se três agentes que interagiam entre si a partir de perguntas e respostas, porém a conclusão dessa interação era gerada randomicamente pelo Jason, pois não haviam emoções e memória implementadas para este sistema. A segunda etapa, porém já utilizava emoções e continha três elementos principais: a IDE de automação em Java Selenium, o Watson e a plataforma de multiagente Jason, conforme são apresentados, porém o resultado final era dado com base na emoção que apareceu mais vezes independente se é uma emoção recente ou antiga.

Já na terceira etapa, foi desenvolvida uma automação utilizando Selenium, a qual atuava apenas com a comunicação entre o Jason e o Watson, transferindo as mensagens de um sistema para o outro como mostrado na figura 4. As emoções identificadas pelo Watson eram transferidas para o agente, no Jason que finalmente, conseguia escolher a resposta adequada entre seus planos. O último estágio da implementação conta com memória, onde ao longo de cada interação que era realizada pelos agentes, armazenava a intensidade de cada uma das emoções.

Figura 4. Integração dos sistemas através do Selenium .

6. Resultados

A análise dos resultados baseia-se na aplicação de um problema relacionado a teoria dos jogos e é conhecido como o "Dilema do Prisioneiro". É uma técnica de análise de jogos padrão, criada em 1950 por Merrill Flood e Melvin Dresher. Para realização dos testes, as regras originais foram levemente alteradas, utilizando o conceito de que um suspeito tem maior chance de confessar caso o policial consiga gerar uma ligação e, ou conforto na conversa através de emoções [Gudjonsson 2003]. Originalmente, na análise padrão de jogos criada por Merrill Flood e Melvin

[preso1] Olá, policial
[policial] Olá Kleberton
[preso1] Por que eu estou aqui?
[preso1] Não fiz nada isto obviamente é um engano
[policial] Estou esperançoso que você tome a decisão correta.
[preso1] você tem esperança que vou fazer a escolha correta?
[policial] Tenho esperança que você coloque a mão na conciencia e confesse o que fizeram.
[preso1] Tenho esperança que percebam que sou inocente logo.
[policial] Consigo perceber o medo crescendo em você.
[preso1] Esou começando a ficar com medo e se eu for condenado?
[policial] Tenho esperança de que você me conte o que aconteceu aquele dia
[preso1] Tenho esperança de que vocês descubram o que aconteceu aquele dia pois eu não sei.
[policial] Espero que você me conte quem era a outra pessoa que estava lá aquele dia.
[preso1] Espero que vocês acreditem no meu alibi.
[policial] Seu alibi não foi confirmado mas você pode ter esperança de ficar pouco tempo na cadeia caso confesse.
[preso1] Tenho esperança de sair livre afinal você parece um policial sério.
[policial] Voce não tem medo de não confessa e acabar pegando pena maxima?
[preso1] temo que o juiz não acredite em mim.
[policial] Tenho esperança que você confesse pois as pistas te ligam ao crime.
[preso1] Tenho esperança de sair livre.
[policial] Então você se declara inocente ou culpado?
[preso1] Hmm.
[preso1] Me declaro culpado

Figura 5. Primeiro diálogo do agente policial com o preso 1

Dresher, há dois prisioneiros, que podem ou não dedurar o seu companheiro de crime. No caso deste trabalho, o suspeito pode confessar o crime ou não, sem dedurar ninguém.

No sistema, são implementados três agentes. Dois deles são interpretados como infratores, que são interrogados por um policial. O policial oferece uma forma de justiça negociada entre os suspeitos para que eles confessem o crime. A conversa sempre inicia pelo policial. Sua comunicação é enviada para um sistema, que é o Watson, ao qual possui um mecanismo de processamento que simula a inteligência humana e consegue identificar as emoções que estão presentes nestas frases com base no modelo OCC, já treinado na etapa anterior deste trabalho.

Após, começam a ocorrer as interações entre estes agentes, entre o policial e o primeiro prisioneiro. Com base nas respostas que são obtidas através dos questionamentos do policial ao criminoso, o agente espera por uma confissão. Durante a conversa, as emoções que se repetem são reforçadas e as que não se repetem são enfraquecidas, caso uma emoção positiva seja predominante as emoções negativas, o prisioneiro confessa o crime. O mesmo ocorre com o segundo prisioneiro, ao qual é interrogado pelo policial. Ao final, o veredito é emitido. Se ambos confessarem, adquirem uma pena de cinco anos de prisão. Se um confessar e outro não, o que não confessou saíra livre e o outro adquire uma pena de dez anos. E se os dois não confessarem, cada um adquirirá um ano de prisão.

Os resultados de uma das simulações utilizando emoções e memória é apresentada a seguir. Durante a interação que ocorre entre o policial e um dos presos, o preso se declarou culpado, pois o agente policial conseguiu gerar a intensidade de uma emoção positiva maior do que a de emoções negativas. Isso pode ser observado no gráfico da Figura 5.

No gráfico da Figura 6 fica claro que a emoção mais intensa foi esperança, pois apesar de algumas quedas, se manteve acima da emoção medo e sua intensidade ao final do diálogo foi maior que a emoção medo.

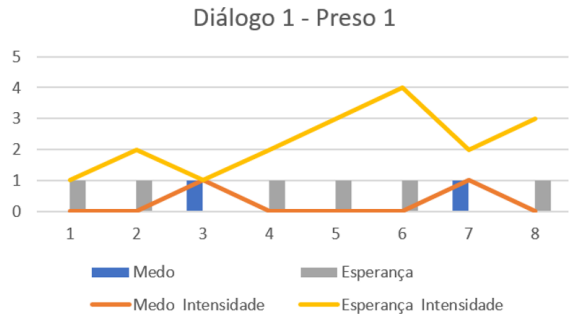


Figura 6. Representação da interação do agente policial com o preso 1

Tabela 2. Tabela representando o gráfico da Figura 6

Emoção	**Diálogo 1 - Preso 1**								
Medo	Sim/Não	0	0	1	0	0	0	1	0
	Intensidade	0	0	1	0	0	0	1	0
Esperança	Sim/Não	1	1	0	1	1	1	0	1
	Intensidade	1	2	1	2	3	4	2	3

Na Tabela 2 são apresentados os valores da intensidade das emoções para cada interação do diálogo apresentado no gráfico da Figura 6.

Apresentou-se o dilema padrão, com emoção e memória. Porém, as possibilidade de interações entre os agentes podem ser infinitas, fazendo com que seja possível realizar n simulações entre SMAs que envolvam memória e emoções.

7. Conclusão

Com os resultados obtidos, observou-se que o trabalho atendeu ao objetivo geral que propunha definir uma arquitetura que utilizasse os conceitos de memória e emoções relacionados a sistemas multiagente e realizar vários experimentos, dentre estes: sem emoções, com emoções e com emoções e memória.

Pode-se concluir que é possível simular memória e emoções, utilizando SMAs de modo que se assemelhe a memória humana. Isto fica claro pela diferença das respostas adquiridas entre os experimentos sem memória e sem emoções, sem memória e com emoções, com memória e com emoções.

Contudo, o diferencial da arquitetura apresentada é que através dos conceitos de memória humana e emoções integradas a um SMA, possibilitou-se simular "problemas"ou "situações"que necessitam destes conceitos, como o problema do Dilema do Prisioneiro adaptado neste trabalho para simular a teoria de que ao gerar *rapport* através de emoções positivas a chance de um suspeito culpado confessar torna-se maior.

Referências

Aboulafia, A. and Bannon, L. J. (2004). Understanding affect in design: an outline conceptual framework. *Theoretical Issues in Ergonomics Science*, 5(1):4–15.

Arraes, H. G. R. (2012). *Arquitetura de um Agente Emocional baseado em Modelos Psicológicos para uso em Jogos Eletrônicos*. PhD thesis, Universidade de Brasília.

Clayton, N. S., Griffiths, D., Emery, N., and Dickinson, A. (2001). Elements of episodic–like memory in animals. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 356(1413):1483–1491.

Cosenza, R. and Guerra, L. (2009). *Neurociência e Educação*. Artmed Editora.

da Silva, B. C. D. (2006). O estudo lingüístico-computacional da linguagem. *Letras de Hoje*, 41(2):103–138.

Ferrucci, D. A. (2012). Introduction to "this is watson". *IBM Journal of Research and Development*, 56(3.4):1–1.

Glehn, L. V. (2018). Construindo uma solução com watson.

Griffiths, D., Dickinson, A., and Clayton, N. (1999). Episodic memory: what can animals remember about their past? *Trends in cognitive sciences*, 3(2):74–80.

Gudjonsson, G. H. (2003). *The psychology of interrogations and confessions: A handbook*. John Wiley & Sons.

High, R. (2012). The era of cognitive systems: An inside look at ibm watson and how it works. *IBM Corporation, Redbooks*, pages 1–16.

Kintsch, W. (2014). *The representation of meaning in memory (PLE: Memory)*. Psychology Press.

Makridakis, S. (2017). The forthcoming artificial intelligence (ai) revolution: Its impact on society and firms. *Futures*, 90:46–60.

Marsella, S., Gratch, J., Petta, P., et al. (2010). Computational models of emotion. *A Blueprint for Affective Computing-A sourcebook and manual*, 11(1):21–46.

Mourão Júnior, C. A. and Faria, N. C. (2015). Memory. *Psicologia: Reflexão e Crítica*, 28(4):780–788.

Norvig, P. and Russell, S. (2014). Inteligência artificial: Tradução da 3a edição (vol. 1).

Ortony, A., Clore, G. L., and Collins, A. (1990). *The cognitive structure of emotions*. Cambridge university press.

Roisenberg, M., Barreto, J. M., and Azevedo, F. M. (1996). Biological inspirations in neural network implementations of autonomous agents. In *Brazilian Symposium on Artificial Intelligence*, pages 211–220. Springer.

Sen, S. et al. (2018). The effects of past experience on trust in repeated human-agent teamwork. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 514–522. International Foundation for Autonomous Agents and Multiagent Systems.

Smith, J. D., Shields, W. E., Allendoerfer, K. R., and Washburn, D. A. (1998). Memory monitoring by animals and humans. *Journal of Experimental Psychology: General*, 127(3):227.

Spear, N. E. (1973). Retrieval of memory in animals. *Psychological Review*, 80(3):163.

Varga, S. (2014). Uma visão geral sobre o uso de sistemas de perguntas e respostas na ciência cognitiva. *Campinas: Universidade Estadual de Campinas*.

Automated Planning and BDI Agents: a Case Study

Rafael C. Cardoso[1], **Angelo Ferrando**[2], **Fabio Papacchini**[3]

[1]Department of Computer Science – The University of Manchester
Manchester M13 9PL, UK

[2]Department of Computer Science – University of Genova
Genova 16145, Italy

[3]Department of Computer Science – University of Liverpool
Liverpool L69 3BX, United Kingdom

rafael.cardoso@manchester.ac.uk

angelo.ferrando@unige.it

fabio.papacchini@liverpool.ac.uk

Abstract. *There have been many attempts to integrate automated planning and rational agents. Most of the research focuses on adding support directly within agent programming languages, such as those based on the Belief-Desire-Intention model, rather than using off-the-shelf planners. This approach is often believed to improve the computation time, which is a common requirement in real world applications. This paper shows that even in complex scenarios, such as in the Multi-Agent Programming Contest with 50 agents and a 4 second deadline for the agents to send actions to the server, it is possible to efficiently integrate agent languages with off-the-shelf automated planners. Based on the experience with this case study, the paper discusses advantages and disadvantages of decoupling the agents from the planners.*

1. Introduction

Automated (or also referred to as Artificial Intelligence) planning consists of using a search algorithm to find a solution to a problem [Nau et al. 2004]. The planner receives as input information about the domain (predicates and actions that can be applied) and the problem (initial state of the environment and goals) and will apply a search algorithm to transition between states until the goals have been achieved. The output of a planner is a plan containing a sequence of actions that achieves the goals of the problem. The action theory in STRIPS (STanford Research Institute Problem Solver) [Fikes and Nilsson 1971] is the backbone of later formalisms such as the widely used PDDL (Planning Domain Definition Language) [Mcdermott et al. 1998]. PDDL has been the de-facto standard formalism for representing classical planning problems, but it has also been extended to support temporal, probabilistic, and other types of planning. In this paper, we focus on classical PDDL/STRIPS task planning.

Autonomous agents and Multi-Agent Systems (MAS) have vast and diverse subareas of research. In particular, we focus on Agent-based programming [Cardoso and Ferrando 2021], which uses agent-oriented languages to implement some of the concepts found in MAS. The Belief-Desire-Intention

model [Bratman 1987, Rao and Georgeff 1995] is used in most agent-based programming languages [Bordini et al. 2020], as well as in research in the area of agent programming [Logan 2018]. The BDI model consists of a reasoning cycle based on three main concepts: *beliefs* – represent knowledge about the world, *desires* – goals to achieve, and *intentions*, steps that can be made towards achieving something. A simple reasoning trace starts with changes in the belief base which can trigger plans to achieve a goal, which if selected for execution will go through the stack of intentions of the plan and execute them one by one.

Most agent-based programming languages have some form of inherent task planning. Agents have access to a plan library and based on certain triggering events a search is made to find applicable plans from their library. The difference here is that interpreting actions (i.e., their pre-conditions and effects) is delegated to the environment and is not a concern of the agent. This kind of planning is more similar to the non-primitive tasks that can be found in Hierarchical Task Planning (HTN) [Nau et al. 2003]. Even though action descriptions are still needed for HTN planning, the non-primitive tasks (also called methods) are very similar to what plans represent in BDI programming in terms of search, that is, they allow the search space to be effectively pruned. Note that this is different from adding explicit support for automated planning in agent-based programming languages. We discuss approaches that do that later on in the Related Work section.

However, plans in the agents' libraries are usually pre-designed by a developer. Complex case studies and unpredictable environments may result in a plan library that does not contain the necessary plans to solve some of the problems that may appear. This is where using automated planners to generate those missing plans (either at runtime as we will demonstrate in this paper, or offline at design time) may remove the burden from the developer of trying to encode every possible solution.

In this paper we use an off-the-shelf planner in combination with our agents that were programmed using a BDI agent-based language to solve some complex problems in the 14th and 15th Multi-Agent Programming Contest[1] scenarios [Ahlbrecht et al. 2020]. The main problem we wanted to solve with this integration is to plan optimal movement of the agents in a grid environment with optional actions for clearing obstacles. Agents plan individually because they have partial view of the environment (i.e., they can only see a few squares around them), thus, a centralised approach would not be very effective. It is also important to note that since we have a grid environment with extra actions (i.e., state-based), task planners are much more suited to tackle this problem rather than traditional path planning algorithms, since there are no real actuators, effectors, or sensors in play.

We explain how we have made this integration work and what had to change between the 14th and 15th editions to still make this strategy effective. In particular, we have used the Fast Downward[2] [Helmert 2006, Helmert 2009] planner for the automated planning, and the JaCaMo[3] [Boissier et al. 2013, Boissier et al. 2020] multi-agent oriented programming framework.

The rest of this paper is organised as follows. In the next section we discuss some

[1] `https://multiagentcontest.org/`
[2] `http://www.fast-downward.org/`
[3] `http://jacamo.sourceforge.net/`

approaches that have tried to explicitly incorporate planning in BDI languages, as well as BDI applications that have used off-the-shelf planners in the past. Section 3 briefly explains our case study, with a particular focus on the elements that were advantageous to apply automated planning. In Section 4, we describe in detail how we have used an off-the-shelf planner to solve the problems described in the previous section. A discussion about our experience with the combination of automated planning and BDI agents is presented in Section 5. We conclude the paper and list some future directions in Section 6.

2. Related Work

Automated planning and BDI programming have been used individually to solve an assortment of different tasks in the past. An example of the former is [Borgo et al. 2016], where an off-the-shelf planner is used in a manufacturing plant for logical reconfiguration of the control nodes after changes in the environment (production change, fault in physical components, or changes in the production goals). An example of the latter is [Cardoso et al. 2021], where agents are implemented in a BDI-based programming language to support ethical reasoning by adding agents that can recommend ethical actions to be executed.

Some approaches have been developed which aim to integrate BDI programming languages with off-the-shelf automated planners. An example of such approach is the work in [Cardoso and Bordini 2019], which uses the SHOP2 HTN planner [Nau et al. 2003] without requiring any modifications to the planner itself. Instead, the agents in the JaCaMo framework call an individual instance of the planner when required to perform multi-agent planning, with planning being coordinated and tasks allocated at runtime using agent communication protocols and techniques. There are several differences between our work and theirs: (a) we do not perform coordinated multi-agent planning, our agents plan individually to each other; (b) we use a PDDL planner while they used HTN; and (c) we focus on a more practical and complex case study.

Conversely, there have been many approaches that tried to integrate automated planning directly into BDI programming languages. In [de Silva et al. 2009] first principles classical planning is introduced to a theoretical (never implemented) BDI language through the derivation of abstract planning operators from BDI programs. A further extension of this work is reported in [Sardiña and Padgham 2011], which adds failure handling and declarative goals on top of the planning, however even though the theoretical contributions were important at the time, the main issue still remains that the language described was never implemented. A more practical approach with a mapping between classical planning formalisms and traditional BDI agent languages is presented in [Meneguzzi and Luck 2013], describing a formal translation process from BDI plans to classical planning operators. We believe there are a number of fundamental differences in using off-the-shelf planners and integrating automated planning directly into BDI programming languages. We discuss what these differences are in Section 5.

3. Multi-Agent Programming Contest: Agents Assemble

The Multi-Agent Programming Contest (MAPC) is an annual competition with complex multi-agent oriented challenges (such as communication, coordination, and interaction between agents) that aims to promote and strengthen the use of multi-agent programming

frameworks, tools, and methodologies. By applying these techniques to difficult scenarios it is possible to identify which features are missing, can be improved, or have to be fixed in agent-based technologies. The scenarios change every few years, with extensions being done to the previous scenarios on off years. The core structure of the simulations remain mostly the same: it is a synchronous step-based simulation wherein clients (the teams) have to send actions to the server under a certain deadline (usually 4 seconds per step), and a match is composed of 3 rounds (each with a different map).

The 14th edition of the MAPC [Ahlbrecht et al. 2020] introduced the "Agents Assemble" scenario. In this scenario, two teams of 10 agents each compete to accumulate the most amount of currency by assembling block structures to match tasks announced by the server in a grid environment. We participated as team LFC (Liverpool Formidable Constructors) [Cardoso et al. 2020] and achieved first place. We believe one of the main factors in our exemplary performance during that edition of contest was due to our strategy in using an off-the-shelf planner to generate plans for movement at runtime. The scenario has many other details which required an assortment of different strategies that the interested reader can find more information about in [Ahlbrecht et al. 2020, Cardoso et al. 2020]. For the remaining of this paper, we focus only on the elements of the contest that were relevant for the use of the automated planner and its integration with the agents.

The 15th edition of the MAPC [4] used the same scenario but extended it in many interesting ways. Of relevance to our planning strategy was the change from the static 10 agents to 15 agents in round 1, 30 agents in round 2, and 50 agents in round 3. This required us to adapt our strategy, as calling 30 and 50 instances of the planner would slow down the reasoning of our agents and make them unable to send an action before the deadline. Our solution to this was developing a plan cache, which we go into detail later on in Section 4.2. We achieved second place in the 15th edition of the MAPC, however we believe our planning strategy performed very well and we only lost due to the lack of optimisation in some of our other strategies (such as task assembly).

In this paper, we focus on the challenge of performing efficient movement in the grid. Map grids during the 14th and 15th MAPC ranged from 50x50 (2500 cells) to 100x100 (10000 cells). Each agent has a local view of 61 cells around them. An example of the local view of an agent is shown in Figure 1. Initially, we tried to encode the agents' movements directly into the agent program, however, we soon realised that there were too many edge cases to consider and that our solution resulted in longer routes and even a few deadlocks. Using an automated off-the-shelf planner allowed us to focus on the other strategies while knowing that the movement of our agents were very efficient.

The main challenge for planning the agents' movement (either directly in the agents' program or in an automated planner) is the dynamic nature of the environment and the lack of knowledge of the agent about cells outside its local vision. For example, the following dynamic events can happen during a step: (a) an agent from the other team (or from our own team, if we do not disclose movement information to other agents) may move into one of the cells that are a part of our agent's route, generating a conflict; (b) a special event called a *clear* event has a random chance of occurring each step which may

[4] https://multiagentcontest.org/2020/

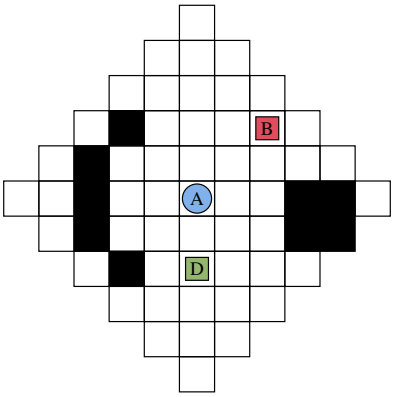


Figure 1. Example of local view of an agent (A). Black squares are obstacle cells, green squares with a (D) are dispensers that can spawn blocks when requested by an agent, and red squares with a (B) are blocks.

remove obstacle cells, create obstacle cells, disable agents, and remove blocks; and (c) agents also have access to a *clear* action, however compared to the environment event this action has a reduced radius and can not create obstacles.

4. Automated Planning and BDI Agents

Our team[5] was programmed in the JaCaMo multi-agent oriented development framework using its Eclipse plugin. JaCaMo has three main programming dimensions (agent, environment, and organisation), each handled by a different tool that has been developed over many years and then integrated to work well with each other. Jason [Bordini et al. 2007] is a popular BDI-based agent programming language that has been shown to be one of the most efficient agent-based languages in several benchmarks [Cardoso et al. 2013, Mohajeri Parizi et al. 2020]. We used Jason to define our agent programs. CArtAgO [Ricci et al. 2009] is based on the notion of artifacts that are used to represent and interact with the environment, and in our case this meant interfacing with the server of the MAPC as well as interfacing with the off-the-shelf planner. Finally, Moise [Hübner et al. 2007] provides organisation constructs to determine roles and norms in groups of agents, which our team used to aid in the coordination of certain tasks.

Fast Downward (FD) is a well-established planner which has been used several times in the International Planning Competition (IPC). For example, different variations of the planner were used in the IPC in 2018[6] and performed well in the classical, satisficing, and bounded-cost tracks[7]. Clear actions take three steps to be successfully used by an agent, and we needed to encode it as a planning operator because clearing obstacles can make much more optimal routes. However, FD does not support numeric planning. To workaround this problem we made use of action costs where all movement actions have a cost of 1 and the clear action has a cost of 3, and then the planner is asked to find plans that can minimise the cost.

[5] Our team code for the 14th MAPC:
`https://github.com/autonomy-and-verification-uol/mapc2019-liv`
Our team code for the 15th MAPC:
`https://github.com/autonomy-and-verification-uol/mapc2020-lfc`

[6] `https://ipc2018.bitbucket.io/`

[7] `https://ipc2018-classical.bitbucket.io/#results`

Next, we describe our planning strategies for the 14th MAPC (Local Vision Planning) as well as the necessary extensions for it to work well in the 15th MAPC (Plan Cache).

4.1. Local Vision Planning

We limited the use of the planner only for performing the movement of the agents in the grid (which includes the clear action for more efficient routes). If the grid cells were static and the map was completely observable then it would have made more sense to try to plan the entire route of the agent at once. However, apart from the fact that this solution would likely result in performance problems, the cells in the grid were dynamic and could change because of clear actions or agents' actions. The local vision and the lack of knowledge of the remaining cells in the map also made it pointless to plan too far in advance. Therefore, we call the planner with a target cell that is inside the local vision of the agent. An example of how this works is given later on.

Planning is performed at runtime and an instance of the FD planner is invoked by the agents when they required planning. The workflow is defined as follows:

1. The agent wants to move to a target position in the grid
2. Is the target position inside the agent's vision? If yes jump to step 4
3. Select the closest cell to the final target position that is inside the agent's vision
4. Determine what type of domain will be used, planning operators change depending on specific circumstances:
    - if the agent has enough energy to perform clear actions enable clear as a planning operator
    - if the agent has a block attached (maximum one block attached for planning movement) enable operators to move with one block
5. Call a CArtAgO artifact which will generate the problem file based on the local vision of the agent (obtained through the perceptions observed in that step from the server), with the target cell as a goal, and the appropriate domain (based on the flags determined in previous steps)
6. Call an instance of the FD planner with the generated problem file and the appropriate pre-generated domain file
7. The solution plan is saved onto a file which the CArtAgO artifact reads and then translates it back to the agent
8. If a plan was found, the agent simply executes each action of the plan at each step (if yes jump to step 10)
9. If a plan is not found, then the agent tries to move (without the planner) in a direction that would bring it closer to the final target, and then calls the planner again (back to step 2)
10. If the agent is not in the final target cell after executing all actions in the plan, the process goes back to step 2, otherwise planning has finished

The 14th edition of the MAPC had 10 agents per team. Initially, we thought that if all of our 10 agents started an instance of the planner at the same step, then we may have encountered some performance problems. To circumvent this, we introduced a planning counter which would limit the number of agents allowed to start an instance of the planner at any given step when the counter hits the maximum number allowed. Setting this number varied depending on the processing power of the computer that was

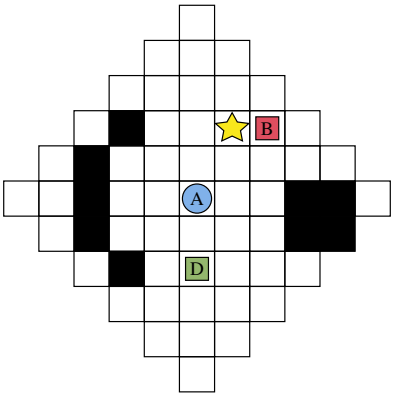
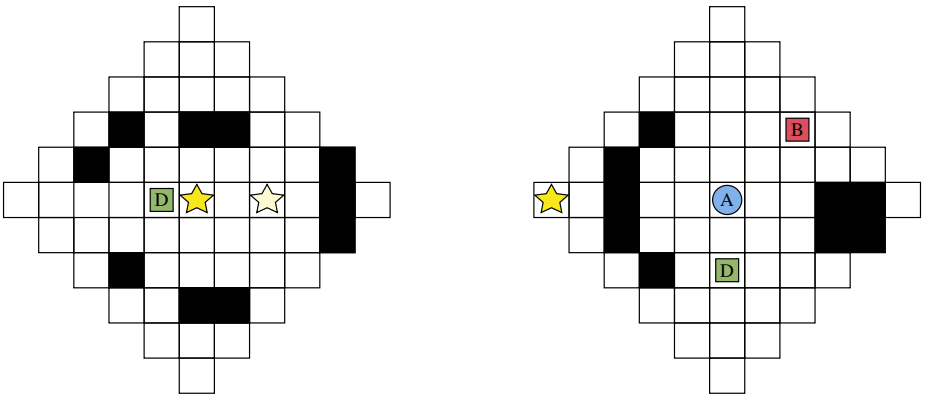


Figure 2. Example of a final target position (star) inside the vision of the agent.



(a) Example of where the final target position would be with respect to Figure 3b. The faded star represents the expected position of the agent after Figure 3b.

(b) Example of a partial target position (star) with a final target outside the vision of the agent.

Figure 3. Examples of planning targets (destination cells) for when the target is outside the vision of the agent.

being used. During the contest, we disabled this feature, as the computer we used to run was powerful enough to comfortably handle the maximum number of agents.

Example.

To better explain the difference between local and final target selection we exemplify some scenarios in Figure 2 and Figure 3.

Figure 2 shows an example of when the final target position is inside the vision of the agent. Since we are dealing with local vision the coordinates are all local with respect to the agent, that is, the agent is in cell (0,0). Therefore, the target position in this example is set to cell (1,-2), one cell to the right and two cells above. This is the simplest scenario, and when the agent arrives at the destination planning is concluded.

Figure 3a has a target cell that is outside the local vision of the agent. It would be inefficient to try to plan the whole route since things could change and then it would require replanning the route multiple times. Therefore, first we must find a target cell inside the agent's vision that would bring it closer to the final target, which in this case is shown in Figure 3b to be cell (-5,0). Upon arriving at the local target the agent recalculates

the difference to its final target, which in this example would result in cell (-2,0), and then calls the planner once again.

4.2. Plan Cache

Using automated planning at runtime is far from being an easy task. The entire process, from encoding the state of the agent as a problem file, to solving it with a planner, is computationally demanding. This can be reasonable for small and simple applications, but it becomes an issue when applied to large and complex systems, such as a MAS. Indeed, even though the planning problem for a single agent is feasible, it might not be for a coalition of agents as well. Let us assume we have N agents, we would need to call the planner N times (one per agent). Considering the MAPC scenario, this could happen in each step of the simulation.

Since physical resources (CPU, memory) and time (how long an agent can wait) are finite, it is always possible to pick a number N of agents for which it is not possible to solve the planning problem in less than a certain amount of time (4 seconds for each simulation step in the MAPC). In particular, as we mentioned previously the 15th edition of the contest extended the scenario to have 15 agents in round 1, 30 agents in round 2, and 50 agents in round 3. Through testing, we noticed that our previous strategy managed to hold up for 15 agents, but it did not work for 30 and 50 agents. Because of this, alternatives to speed the planning process up must be considered.

We investigated alternatives to speed up the process, and found that one possible way to make the planning process faster is by *caching* the plans. By caching, we refer to the act of storing previously generated plans, instead of forgetting about them after execution. When an agent asks the planner to solve a problem, if such a problem has been already solved in the past, there is not really the necessity for the planner to waste time in solving it again. This can be achieved by keeping a mapping between $Problem \rightarrow Plan$, which given a problem file, returns its corresponding plan (if present in the cache). When a problem file does find a match in the cache, it means that it has never been solved before (cache miss), in which case the execution continues by calling the planner and then updating the cache. Note that this cache is saved independent of the size of the grid, number of agents, or any other parameter. This means that the cache can be used in any configuration to speed up the planning process.

The first aspect we have to consider about caching is how to encode a problem file, so it can be straightforwardly retrieved from the cache. Such an encoding must uniquely identify a problem file. Thus, all information which describe the agent's local view needs to be considered. In Figure 4, we report an example of how we generate such encoding.

Starting from the agent's local view (left side of Figure 4), a possible encoding can be obtained by unrolling the grid as a one-dimensional array (top right of Figure 4). The unrolling starts from the upper most cell, and then all rows are appended one by one from left to right. Finally, in the bottom right of Figure 4, we can see a possible encoding of the unrolling; where empty spaces are mapped[8] to 0 (dispensers and the agent current position are considered empty), obstacles to 1, blocks to 2[9], and the goal to 3.

[8]The mapped values are not semantically relevant, as long as is preserved for all mappings.

[9]After the contest we realised that since blocks are considered as obstacles for the planner, we could have set their value to 1, which would have decreased the number of cached plans at least by a small margin.

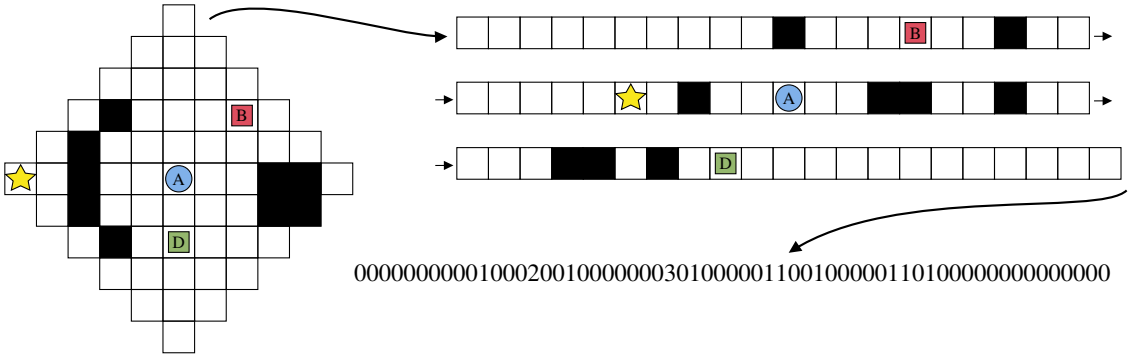


Figure 4. Example of the step by step process of converting from a problem description to an encoding that can be easily read.

Once the encoding is created, it can be used to query the cache. Since we want to use the cached plans amongst different executions, the cache is stored in the secondary memory. More in detail, for each cached plan, a file is stored. The encoding of the agent's local view is used to name such a file. Consequently, to check if a certain planning problem has already been solved, it is enough to look if there exists a file named as the encoding of the problem. If such a file exists, there is no need to call the planner since the corresponding plan is already available inside the file, otherwise, the planner is called and a new file is stored (named after the encoding of the problem).

For instance, in the example of Figure 4, the plan returned and stored as a file (named using the encoding) will contain the sequence of actions reported in Listing 1.

```
1 clear(-3,0)
2 move(w)
3 move(w)
4 move(w)
5 move(w)
6 move(w)
```

Listing 1. Plan returned by the planner for Figure 4, and stored in the cache.

As we have seen previously, it is possible to call the planner in two different settings. The first one is calling for a plan to the agent by itself, which we have already covered how to encode it. The second is that we may call for a plan to the agent with a block currently attached. The encoding for this is almost identical, except that in this case we append the local view coordinates of the block that is attached (note that our planning domain only supports movement with up to one attached block). For example, if there is a block attached to the agent and the block is located one cell below the agent, then we would append 01 to the beginning of the encoding.

5. Discussion

The most glaring benefit of using an off-the-shelf planner is that we are able to take advantage of the many sophisticated and efficient tools that have been developed over the years in the community of automated planning. Re-implementing the features provided by these tools in an agent programming language can take a lot of effort. A more sub-

tle advantage is that if the application domain requires distinct planning techniques for different types of problems in the system then we can use multiple off-the-shelf planners that are most appropriate for each problem. For example, we may need at one point to perform probabilistic planning, and at another stage we require temporal planning.

A limiting factor in using off-the-shelf planners with autonomous agents at runtime in time critical applications is that the solution may not be produced inside acceptable time bounds. Initially, we did not consider using automated planners to be feasible because we thought we would run into this issue. However, after testing we noticed that for 10 agents we could comfortably perform planning under a four seconds restriction. When the number of agents increased to 30 and 50, then we had to modify our solution by performing plan caching which provided excellent results. Our initial analysis indicates that using off-the-shelf planners in domains with loosely coupled agents (low or no interactions between agents) and/or low number of agents should generate results very quickly. Note that determining if the result can be produced inside the time constraint depends specifically on the application domain. Alternative solutions such as plan caching can also make previously unfeasible planning scenarios work well, but the size of the cache must be controlled since it can eventually increase up to a point where it is no longer worth using.

The seamless integration of a planner in an agent programming language has the advantage that the computing performance can be vastly superior. This is not only because the agents themselves can perform planning directly without having to call a separate process, but also because *lookahead* online planning can be used, wherein only a subset of the problem is planned for and then executed before proceeding to plan for the rest.

The direct integration of planning in agents can restrict the type of planning supported, which can limit its applicability to certain domains. Furthermore, it can make the code difficult to port to other agent languages. For example, in the MAPC it is required for all teams to make their source code available after the contest. This can help new teams (or under performing teams) to look for inspiration in strategies that have worked well in the past. Because we used an off-the-shelf planner, other teams can easily make similar use of it without being tied to the agent development framework we used (JaCaMo).

6. Conclusion

In this paper, we have described our experience in applying an off-the-shelf automated planner at runtime for a very complex problem requiring a solution under a strict deadline. We have integrated this solution with agents that were developed in a BDI agent programming language. Our case study was based on the scenario first introduced in the 14th edition of the MAPC (and then later extended on the 15th edition). There are advantages and disadvantages to either using off-the-shelf planners or integrating automated planning directly in an agent programming language. Ultimately, the best choice is going to depend on the complexity and limitations imposed by the application/case study as well as the tools and languages that are available to solve the problem.

Future work involves trying to categorise different classes of applications and case studies to understand at a more general level when can an off-the-shelf planner be used alongside an agent programming language with minimal effort, as well as listing possible combinations of planners and agent-based languages that could work well together.

Acknowledgements

Work supported by UK Research and Innovation, and EPSRC Hubs for "Robotics and AI in Hazardous Environments": EP/R026092 (FAIR-SPACE) and EP/R026084 (RAIN). Cardoso's work is also supported by Royal Academy of Engineering.

References

Ahlbrecht, T., Dix, J., Fiekas, N., and Krausburg, T. (2020). The multi-agent programming contest: A résumé. In Ahlbrecht, T., Dix, J., Fiekas, N., and Krausburg, T., editors, *The Multi-Agent Programming Contest 2019*, pages 3–27, Cham. Springer International Publishing.

Boissier, O., Bordini, R., Hubner, J., and Ricci, A. (2020). *Multi-Agent Oriented Programming: Programming Multi-Agent Systems Using JaCaMo*. Intelligent Robotics and Autonomous Agents series. MIT Press.

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

Bordini, R. H., Seghrouchni, A. E. F., Hindriks, K. V., Logan, B., and Ricci, A. (2020). Agent programming in the cognitive era. *Auton. Agents Multi Agent Syst.*, 34(2):37.

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

Borgo, S., Cesta, A., Orlandini, A., and Umbrico, A. (2016). A planning-based architecture for a reconfigurable manufacturing system. In *Proceedings of the 26th International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'16, pages 358–366, London, UK.

Bratman, M. E. (1987). *Intentions, Plans, and Practical Reason*. Harvard University Press.

Cardoso, R. C. and Bordini, R. H. (2019). Decentralised planning for multi-agent programming platforms. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, pages 799–807, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

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

Cardoso, R. C., Ferrando, A., Dennis, L. A., and Fisher, M. (2021). Implementing ethical governors in bdi. In *9th International Workshop on Engineering Multi-Agent Systems*.

Cardoso, R. C., Ferrando, A., and Papacchini, F. (2020). Lfc: Combining autonomous agents and automated planning in the multi-agent programming contest. In *The Multi-Agent Programming Contest 2019*, pages 31–58, Cham. Springer International Publishing.

Cardoso, R. C., Zatelli, M. R., HÃ¼bner, J. F., and Bordini, R. H. (2013). Towards benchmarking actor- and agent-based programming languages. In *Workshop on Programming based on actors, agents, and decentralized control*, pages 115–126, Indianapolis, Indiana, USA.

de Silva, L., Sardiña, S., and Padgham, L. (2009). First principles planning in bdi systems. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pages 1105–1112, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Fikes, R. E. and Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3):189 – 208.

Helmert, M. (2006). The fast downward planning system. *J. Artif. Intell. Res.*, 26:191–246.

Helmert, M. (2009). Concise finite-domain representations for PDDL planning tasks. *Artif. Intell.*, 173(5-6):503–535.

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. *Int. J. Agent-Oriented Software Engineering*, 1(3/4):370–395.

Logan, B. (2018). An agent programming manifesto. *International Journal of Agent-Oriented Software Engineering*, 6(2):187–210.

Mcdermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). PDDL - The Planning Domain Definition Language. Technical Report TR-98-003, Yale Center for Computational Vision and Control.

Meneguzzi, F. and Luck, M. (2013). Declarative Planning in Procedural Agent Architectures. *Expert Systems with Applications*, 40(16):6508 – 6520.

Mohajeri Parizi, M., Sileno, G., van Engers, T., and Klous, S. (2020). Run, agent, run! architecture and benchmarking of actor-based agents. In *Proceedings of the 10th ACM SIGPLAN International Workshop on Programming Based on Actors, Agents, and Decentralized Control*, AGERE 2020, pages 11–20, New York, NY, USA. Association for Computing Machinery.

Nau, D., Ghallab, M., and Traverso, P. (2004). *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Nau, D. S., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., and Yaman, F. (2003). SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20(1):379–404.

Rao, A. S. and Georgeff, M. (1995). BDI Agents: From Theory to Practice. In *Proc. 1st Int. Conf. Multi-Agent Systems (ICMAS)*, pages 312–319, San Francisco, USA.

Ricci, A., Piunti, M., Viroli, M., and Omicini, A. (2009). Environment programming in CArtAgO. In *Multi-Agent Programming: Languages, Tools and Applications*, Multiagent Systems, Artificial Societies, and Simulated Organizations, chapter 8, pages 259–288. Springer.

Sardiña, S. and Padgham, L. (2011). A BDI Agent Programming Language with Failure Handling, Declarative Goals, and Planning. *Autonomous Agents and Multi-Agent Systems*, 23(1):18–70.

Um estudo sobre sistemas multiagentes e a categorização de diferentes níveis para a transferência de conhecimento

Paulo Felipe G. L. Rodrigues[1], **Diana F. Adamatti**[1], **Eder Mateus N. Gonçalves**[1]

[1]Centro de Ciências Computacionais (C3)
Universidade Federal do Rio Grande (FURG) – Rio Grande – RS – Brasil

paulorodrigues@furg.br, {dianaada,eder.m.golcalves}@gmail.com

Abstract. *This article presents an analysis of agents, multi-agent systems, knowledge transfer and techniques for transfer. It is proposed to classify different types of transfer, separated into layers for the level of agents, organizations and societies, investigating how policies for carrying out tasks by agents are understood in the literature.*

Resumo. *Este artigo apresenta uma análise sobre agentes, sistemas multiagentes, transferência de conhecimento e técnicas para a transferência. Propõem-se a classificação de diferentes tipos de transferência, separadas em camadas para o nível de agentes, organizações e sociedades, investigando como políticas para a realização de tarefas por agentes são compreendidas na literatura.*

1. Introdução

Um sistema multiagente é composto por um ou mais ambientes, e os agentes dispostos nestes mundos. Um agente deve ser projetado com um objetivo e precisa realizar tarefas, individualmente, em conjunto ou antagonicamente, para alcançá-lo. As ações tomadas por agentes alteram o ambiente em cena, algo que podemos relacionar a um cenário de um jogo digital por exemplo. As ações de um agente visam a adaptação ao ambiente e suas mudanças [Grover et al. 2018, Lesser 1999, McArthur et al. 2007].

Agentes inteligentes são autônomos e possuem a capacidade de percepção, de raciocinar sobre ações, de interação com outros agentes e decisão. Essas interações são definidas com base em uma linguagem comum estabelecida e nos objetivos de cada agente. O comportamento de um agente é regido pelo ambiente em sua volta e as atividades que podem ser realizadas para a conclusão de objetivos [Demazeau 1995, McArthur et al. 2007, Tuyls and Stone 2017].

Tanto as atividades quanto as interações de agentes são restringidas por normas e políticas de convenção social. Estas medidas definirão os níveis para a troca de conhecimento entre agentes e a formação de organizações e sociedades em um ambiente [Demazeau 1995, Huhns and Stephens 1999, Shoham and Tennenholtz 1995, Takadama et al. 2000].

Para a realização da transferência de conhecimento em sistemas multiagentes, algumas técnicas como *Transfer learning - TL* (Aprendizagem por Transferência) e *Reinforcement learning - RL* (Aprendizado por Reforço) são empregadas e podem ser encontradas com frequência na literatura. A transferência de conhecimento mostra-se importante,

pois impacta no quão bem um agente consegue concluir suas atividades e no grau de complexidade atribuído a este ator. Um agente em um ambiente desconhecido, aprende com outro agente ao trocar conhecimentos/políticas [Liang and Li 2020].

Este artigo tem o objetivo de instanciar o problema de transferência de conhecimento sobre os diferentes níveis de um sistema multiagente, a saber, agente, organização e sociedade. Além disso, visa discutir sobre as principais técnicas existentes para a transferência de conhecimento. Desta forma, torna-se possível obter uma base teórica e prática para um melhor entendimento de como aplicar os conceitos estudados em modelos de organizações e sociedades de agentes. O trabalho está estruturado em uma seção dedicada a uma revisão dos conceitos acerca dos temas dispostos, uma visão sobre os diferentes níveis para a transferência de conhecimento, separados em camadas de agentes, sociedades e organizações, e sua conclusão.

2. Referencial Teórico

Nesta seção serão apresentadas contribuições bibliográficas acerca dos temas propostos para o artigo.

2.1. Agentes

Objetos tornam-se agentes quando estão associados a capacidade de percepção, comunicação, raciocínio e decisão. Tais capacidades permitem aos agentes processarem autonomamente a informação para resolver um problema e o sistema a ser simulado, além da informação relativa a outros agentes, ao domínio e ao idealizador/utilizador [Demazeau 1995].

2.2. Sistemas Multiagentes

[Lesser 1999] define sistemas multiagentes como ambientes computacionais em que agentes interagem ou trabalham de forma cooperativa para satisfazer seus objetivos. Estes ambientes podem ser heterogêneos ou homogêneos, considerando um agente disposto em um sistema como um objeto resolvedor de problemas, com certo grau de autonomia.

2.3. Conhecimento

Um sistema de inteligência artificial que possua e utilize estruturas de dados que contenham objetos e processos, por exemplo, pode ser classificado como um sistema representativo de conhecimento, sendo esta representação um dos principais problemas nesta área de estudo [Newell et al. 1982].

De acordo com [Newell et al. 1982], o conhecimento é intimamente ligado a racionalidade. Segundo o autor, sistemas com agentes capazes de emular certo grau de racionalidade podem ser classificados como possuidores de conhecimento. Este conhecimento é capaz de realizar ações para alcançar objetivos.

2.3.1. Conhecimento dos Agentes

[Newell et al. 1982] elabora o conceito de nível de conhecimento (*knowledge level*), para a representação do mesmo. O nível de conhecimento é composto por agentes, que representam um sistema, onde seus componentes são os objetivos, ações e os respectivos

corpos. Um agente em si é composto por estas partes. Os processos dos agentes devem determinar quais decisões serão tomadas pelos atores. A divisão é realizada da seguinte forma:

- Em um primeiro momento, o agente;
- Em seguida, o conhecimento. Através dele, os agentes podem determinar quais ações são apropriadas em situações específicas;
- No último caso, em relação as leis de comportamento, está o princípio da racionalidade, onde são selecionadas estas ações para atingir os objetivos gerados pelo comportamentos dos agentes.

2.3.2. Conhecimento das Organizações

Conforme [Demazeau 1995], as organizações de agentes podem ser dinâmicas, inspiradas em estudos biológicos, ou mais rígidas, inspiradas em leis sociais e baseadas em estudos sociológicos. As trocas de informação e conhecimento entre os agentes, afetarão o domínio da aplicação como um todo e a organização em questão.

O conhecimento em nível organizacional pode ser qualificado como aquele referente à realização de tarefas, atribuição de papéis e ao desenvolvimento de relacionamentos em uma determinada organização, como a relação entre uma autoridade e seus subordinados [Takadama et al. 2000].

2.3.3. Conhecimento em Sociedades de Agentes

Quando trabalhamos com inúmeros agentes em um único ambiente, a complexidade do sistema aumenta consideravelmente, inviabilizando manipular estes atores de maneira individual. Para a solução deste problema, deve-se observar os agentes do ponto de vista de uma sociedade multiagente, mais facilmente controlável e operacional [Huhns and Stephens 1999].

Em uma sociedade de agentes, os atores ocuparam os papéis de administradores, tomadores de decisão, associações, cidadãos, entre outros, com problemas a serem resolvidos do tipo ecológico ou econômico, por exemplo. Estas sociedades, por sua vez, serão baseadas por políticas e normas. Os protocolos de interação estabelecidos são fundamentais para uma transmissão correta e compreensão de informações e conhecimentos, que impactarão a sociedade multiagente [Demazeau 1995].

2.4. Transferência de Conhecimento

A transferência de conhecimento é um método para o aprendizado de uma nova tarefa em um novo ambiente. Em sistemas multiagentes quando um agente não conhece o ambiente em que está disposto e busca aprender para aumentar suas recompensas, ele se beneficia ao trocar conhecimento com outro agente mais experiente [Liang and Li 2020].

2.4.1. Técnicas

Reinforcement Learning - RL, ou Aprendizado por Reforço, é um das técnicas mais populares e flexíveis para o treinamento de agentes autônomos e transferência de co-

nhecimento, tratando-se de um *framework* bastante empregado em problemas referentes a tomada de decisão. Consiste basicamente de ações aplicadas em um ambiente comum, observando-se o quão boa uma ação se prova em diferentes situações. Todavia, as abordagens clássicas de RL necessitam de muitas interações dos agentes com o ambiente, para o aprendizado de como resolver tarefas, o que pode resultar em um grande custo computacional caso estejam dispostos muitos agentes com muitas tarefas no mesmo mundo, com os agentes errando várias vezes até acertarem em como resolver suas atividades. Esses problemas de escalabilidade intensificam-se em sistemas multiagentes, pois o tomador de decisões precisará raciocinar sobre os outros agentes no mundo [Liang and Li 2020, Silva 2019].

Já a *Transfer Learning* - TL, ou Aprendizagem por Transferência, incrementa o aprendizado através da experiência obtida em tarefas relacionadas, em algo chamado reutilização de conhecimento. O objetivo principal ao se reutilizar o conhecimento é acelerar o aprendizado. Um exemplo de Aprendizagem por Transferência é aprender espanhol antes de estudar português, o que torna a compreensão da nova linguagem mais rápida e fácil do que um outro idioma qualquer. Sobre *Reinforcement Learning*, a *Transfer Learning* ao longo dos anos conseguiu escalonar o processo de aprendizado tanto em ambientes de domínio único quanto em multiagentes [Liang and Li 2020, Taylor et al. 2008, Silva 2019].

3. Os níveis para a Transferência de Conhecimento

O nível mais básico é aquele em que ocorre a troca de informações diretas entre os agentes, levando em consideração graus de credibilidade de um agente ao passar informações para outro, a capacidade de raciocínio de um agente e os objetivos de cada um. Em um nível superior, estão as chamadas organizações de agentes. Essas organizações serão as responsáveis por formular estratégias e planos para a resolução de problemas. Por fim, no topo desta hierarquia, as sociedades de agentes. As sociedades de agentes ocorrem a partir do conjunto de agentes e organizações, compostas por políticas e normas sociais. Os diferentes níveis hierárquicos para a troca de conhecimento podem ser exemplificados da seguinte forma:

- 3. As Sociedades, que transferem políticas e normas. As políticas são criadas a partir das organizações e as normas aplicam restrições ao ambiente;
- 2. As Organizações, que transferem estratégias e planos. Os planos executam estas estratégias e são dependentes dos recursos e ações disponíveis para o agente no modelo organizacional;
- 1. Os Agentes, o nível base, que transferem racionalidades e crenças através de seus protocolos de comunicação;

4. Conclusões e Trabalhos Futuros

Foram apresentados temas e considerações sobre agentes, sistemas multiagentes e transferência de conhecimento, propondo-se uma investigação bibliográfica sobre os diferentes tipos de transferência classificados a nível de agentes, organizações e sociedades. Concluiu-se a partir dos tópicos abordados, que técnicas como Aprendizagem por Transferência combinada ao Aprendizado por Reforço mostram-se mais frequentes na literatura desta área em relação ao tratamento do conhecimento por agentes inteligentes, que podem agir de forma cooperativa, antagônica ou individual.

Para trabalhos futuros, têm-se por objetivos o aprofundamento da pesquisa realizada e o possível desenvolvimento de uma vindoura aplicação, como um jogo digital, própria para a realização da transferência de conhecimento nas camadas de transferência aqui expostas, visando o seu correto funcionamento em um ambiente multiagente comum e a decorrente investigação das relações entre agentes. Em paralelo a isto, a utilização de modelos para a representação e estruturação dos processos de troca de conhecimento entre agentes.

Referências

Demazeau, Y. (1995). From interactions to collective behaviour in agent-based systems. In *In: Proceedings of the 1st. European Conference on Cognitive Science. Saint-Malo*. Citeseer.

Grover, A., Al-Shedivat, M., Gupta, J., Burda, Y., and Edwards, H. (2018). Learning policy representations in multiagent systems. In *International conference on machine learning*, pages 1802–1811. PMLR.

Huhns, M. N. and Stephens, L. M. (1999). Multiagent systems and societies of agents. *Multiagent systems: a modern approach to distributed artificial intelligence*, 1:79–114.

Lesser, V. R. (1999). Cooperative multiagent systems: A personal view of the state of the art. *IEEE Transactions on knowledge and data engineering*, 11(1):133–142.

Liang, Y. and Li, B. (2020). Parallel knowledge transfer in multi-agent reinforcement learning. *arXiv preprint arXiv:2003.13085*.

McArthur, S. D., Davidson, E. M., Catterson, V. M., Dimeas, A. L., Hatziargyriou, N. D., Ponci, F., and Funabashi, T. (2007). Multi-agent systems for power engineering applications—part i: Concepts, approaches, and technical challenges. *IEEE Transactions on Power systems*, 22(4):1743–1752.

Newell, A. et al. (1982). The knowledge level. *Artificial intelligence*, 18(1):87–127.

Shoham, Y. and Tennenholtz, M. (1995). On social laws for artificial agent societies: Off-line design. *Artificial intelligence*, 73(1-2):231–252.

Silva, F. (2019). *Methods and Algorithms for Knowledge Reuse in Multiagent Reinforcement Learning*. PhD thesis.

Takadama, K., Terano, T., and Shimohara, K. (2000). Which organizational knowledge is useful: rare, medium, or well-done?-comparison of different levels of organizational knowledge in multiagent environments. In *2000 26th Annual Conference of the IEEE Industrial Electronics Society. IECON 2000. 2000 IEEE International Conference on Industrial Electronics, Control and Instrumentation. 21st Century Technologies*, volume 4, pages 2891–2896. IEEE.

Taylor, M. E., Kuhlmann, G., and Stone, P. (2008). Autonomous transfer for reinforcement learning. In *AAMAS (1)*, pages 283–290.

Tuyls, K. and Stone, P. (2017). Multiagent learning paradigms. In *Multi-Agent Systems and Agreement Technologies*, pages 3–21. Springer.

Directions for implementing BDI agents in embedded systems with limited hardware resources

Matuzalem Muller dos Santos[1], **Jomi Fred Hübner**[1], **Maiquel de Brito**[2]

[1]Dep. de Automação e Sistemas
Universidade Federal de Santa Catarina (UFSC) – Florianópolis – SC – Brazil

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

matuzalem.m@posgrad.ufsc.br, jomi.hubner@ufsc.br, maiquel.b@ufsc.br

Abstract. *The characteristics of autonomy, reactivity, and proactivity, commonly present in embedded systems used in cyber-physical systems, are often similar to the ones of agents. Although the usage of agents in these scenarios would be beneficial, the lack of tools to implement agents in hardware commonly used in low-cost embedded systems is one of the reasons that prevent embedded agents from becoming a reality. This paper discusses some implementation challenges and design considerations required to develop a framework that allows implementing BDI agents in embedded systems.*

1. Introduction

Over the past decade, research in the field of Cyber-Physical Systems (CPS) has made significant advancements in designing and implementing complex systems using low-cost hardware. However, while the cost of hardware commonly used in embedded systems has decreased and its processing capabilities have increased, it is still a challenge to implement complex systems that require features that are typical of agents, such as autonomy, proactivity, reactivity, and social ability [Aliyuda 2016].

Common development tools used for programming agents, such as JADE and Jason, are Java-based and require more resources than is available in the hardware commonly used for embedded systems. Given this scenario, we are motivated to explore the implementation of tools that promote agents in embedded systems. Due to its popularity, the BDI architecture is an interesting agent paradigm to be considered when conceiving this tool, as both reactive and proactive agents can be implemented using this architecture.

This work provides directions for implementing BDI agents in embedded systems with limited hardware resources. The following section provides background information on CPS, embedded systems, agents, and the BDI architecture, followed by a review of related work and our considerations on implementing BDI agents in embedded systems.

2. Background

This section presents the keys concepts introduced in this article. First, cyber-physical and embedded systems are introduced, which is later be associated with BDI agents, AgentSpeak and Jason.

2.1. Cyber-Physical and Embedded Systems

Cyber-physical systems refer to the confluence of cyber and physical systems, such as embedded systems, distributed sensor systems, and control systems [Rajkumar et al. 2010]. CPS focuses on the *interaction* between its composing systems, allowing the creation of adaptive and predictive systems for enhanced performance [Park et al. 2012]. The computational core of CPS is embedded systems, often distributed [Horvath and Gerritsen 2012].

The field of embedded systems is wide and varied, and it is thus difficult to provide a precise definition for the term. According to [Berger 2002], embedded systems are applied computer systems that run on custom hardware, unlike personal computers that can be used for general computing. Examples of applications that contain embedded systems are calculators, digital watches, heating systems, and televisions.

Since embedded systems are often designed for scenarios with resource-constrained characteristics, computational resources and energy are often limited. According to [Marwedel 2010], one can consider the following metrics to evaluate resource efficiency: energy consumed, usage of execution time, code size, weight, and cost. To build an efficient embedded system, it is expected that the energy consumed, cost, code size, and weight are minimized, while usage of execution time is maximized. Some techniques for efficient resource usage are static memory allocation, out-of-memory running avoidance, and real-time operating system (RTOS) usage to follow real-time restrictions.

2.2. BDI Agents

Due to the multiple applications and scenarios in which agents are used, multiple definitions of the term are available in the literature. [Wooldridge 2009] states that "an agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives". Although there are multiple definitions of the term agent, it is agreed that agents usually present the following common characteristics [Wooldridge and Jennings 1995]:

- Autonomy: agents control their actions and goals, without human intervention;
- Social Ability: agents can interact between themselves over an agent-communication language;
- Reactivity: agents are sensible to the environment and can change their actions based on these perceptions;
- Proactivity: agents exhibit goal-directed behavior, which means that they do not solely respond to changes in the environment.

The Belief-Desire-Intention architecture, based on Bratman's behavioral model [Bratman 1987], is a popular agent architecture. In the BDI architecture, actions taken by an agent result from the agent's beliefs, desires, and intentions [Bordini et al. 2009].

***B**eliefs* are information the agent has about the world. Ambient temperature and time are examples of information that can be represented as beliefs. Note that this information can be out of date or inaccurate [Bordini et al. 2007a]. ***D**esires* represent the state of affairs that the agent might like to accomplish. For example, an agent embedded in an air conditioning may desire the ambient temperature to be under a specific

value, but that does not necessarily mean that the agent is taking actions to fulfill that desire. Lastly, ***I**ntentions* are the state of affairs that the agent has decided to work towards [Bordini et al. 2007a]. In practical terms, these represent the course of actions under execution to achieve the agent's desires [Mascardi et al. 2005].

Beliefs, *Desires*, and *Intentions* are constructs used in the decision model known as *practical reasoning*. Practical reasoning consists of two activities: *deliberation* and *means-end reasoning*. During deliberation, an agent decides what states of affairs it commits to achieve (i.e., which desires are "promoted" to intentions). On means-end reasoning, the agent decides how to achieve these states of affairs [Bordini et al. 2007b]. Practical reasoning is represented in Figure 1.

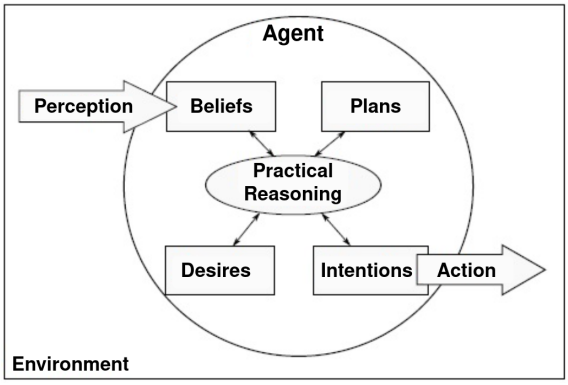


Figure 1. Representation of Practical Reasoning
[Santos 2015]

To summarize Figure 1: the agent perceives the environment and updates its beliefs. After updating its beliefs, the agent starts the deliberation process to determine which objectives it wishes to achieve, considering its current beliefs, desires, plans, and intentions. Once an objective is chosen, a plan of action is selected to fulfill that objective via the means-ends reasoning process. Finally, the agent acts in the environment and later repeats the cycle to re-evaluate its objectives and the state of the world.

2.3. AgentSpeak & Jason

Agent-Oriented Programming (AOP) refers to a paradigm of programming computer systems using the abstraction of agents. In particular, several languages for AOP are based on agent with "mental states", which in the BDI architecture are represented by beliefs, desires, and intentions. AgentSpeak is a popular programming language for AOP, but other languages can also be used, such as the Procedural Reasoning System (PRS), dMARS, and Jadex.

AgentSpeak has been conceived by [Rao 1996] and can be used to implement deliberative agents. The main constructors of the AgentSpeak language are beliefs, goals, and plans. Beliefs represent information the agent has about the world. Goals express the properties of the states of the world that the agent wishes to bring about. Lastly, plans define how an agent can act to reach goals [Bordini et al. 2007b].

Along with the language constructors, AgentSpeak also relies on other components and data structures for agent reasoning. These components are the belief base, plan library, event and intention queues [Santos 2015].

The belief base stores the agent beliefs, which are updated by perceiving the environment. By perceiving the environment and updating its beliefs, events are generated, which are queued for processing and can lead to the execution of a plan available in the plan library. An intention is represented by the execution of applicable plans for desires. Intentions are stored in a data structure: the intention queue.

One of the main differences between BDI programming and traditional programming methodologies is that BDI agents do not end their execution after completing a plan or an action. Instead, BDI agents are always running, perceiving the environment, and (re)evaluating their goals.

To run AgentSpeak, it is necessary an interpreter (or compiler) that supports the language syntax and reasoning cycle. Jason is a popular AgentSpeak interpreter that provides a platform for developing multi-agent systems with additional programming capabilities, extending the AgentSpeak syntax with annotations, internal actions, and others [Bordini et al. 2007a].

3. Related Work

Research in the field of embedded agents is rich and diverse. When searching for the term "embedded agents" in Google Scholar, and filtering the results for the past 5 years (from 2016 to 2021), about 314,000 results are found.

In [Barros et al. 2014] and [Lazarin and Pantoja 2015], the authors use Raspberry Pi and Arduino boards for agent reasoning and acting/perceiving the environment, respectively. Although the projects succeed in proposing a feasible way to embed agents, it falls short in offering a homogeneous system where the agent can reason, perceive, and act in the environment in a single computational platform.

Another attempt to propose using agents in embedded systems is the work of [Aliyuda 2016]. Despite the thorough analysis of how multi-agent architectures can tackle CPS design challenges, the author does not implement embedded agents to address the scenarios described. The work proposes to use the JADE framework to develop agents to address the design issues of CPS. However, the JADE framework is not compatible with most hardware used for embedded systems and would hardly suffice the requirements of running agents in common low-cost hardware.

Lastly, [Bucheli et al. 2015] implements an AgentSpeak translator to embed agents in Unmanned Aerial Vehicles (UAV). The AgentSpeak translator implemented aims to allow programming agents for UAVs with specific characteristics. Therefore, its usage was not tested and validated for common embedded hardware and other applications, making multi-platform and multi-application compatibility harder to achieve.

Due to the hardware constraints presented by common embedded systems, using the tool set usually applied to implement BDI agents becomes a challenge. Frameworks such as JADE and Jason are Java-based, which requires large memory and processing capabilities to run the Java Virtual Machine (JVM) and the agent code on top of it.

To address this challenge, some propose to run heterogeneous systems where agent reasoning and acting in the environment are split between two computational systems. However, this increases the cost of the system, which should be minimized.

4. Agent Design for Embedded Systems

Given the existing solutions, we propose directions for developing a tool that allows the implementation and execution of BDI agents in hardware commonly used in embedded systems, such as Arduino UNO and ESP32 boards. Based on our research, it is desirable that the tool is implemented in a programming language commonly used in embedded hardware, allowing efficient resource management and code optimization. In addition, it is desired that the programmer has control over the size of the internal data structures of the agent, allowing relative control over memory usage and agent characteristics.

This section describes the initial steps towards developing a framework to implement and run BDI agents in the constrained hardware used by embedded systems. These steps include the development methodology, the requirements of the framework, and the requirements of the implemented agents.

4.1. Development Methodology

It is desired that the reasoning cycle and basic features of the framework are based on existing programming languages and interpreters, as the abstractions provided by those can give a sound basis for designing the framework and deciding how features can be implemented. Due to its popularity, we indicate the AgentSpeak programming language and the Jason interpreter as the basis to design the software architecture and define the features of the framework.

In addition, we also recommended the development methodology of the framework to be incremental, where less-complex features are implemented first to validate the framework usage with common embedded hardware. Although the hardware resources required by the framework can only be validated during its implementation, we indicate that the range of hardware between an Arduino UNO and an ESP32 are good target platforms to embed agents due to their processing power, low cost, and popularity. Once confirmed that the target platforms support specific features, these can be implemented and improved, allowing more control over the development process, features implemented, and overall compatibility.

4.2. Requirements of the Framework

The framework should not target a specific hardware platform. Instead, it should be compatible with platforms that provide the minimum requirements of the framework; this way, agents can be implemented in heterogeneous hardware and used for various applications.

Although Jason provides a good starting point for designing the framework, it is desired that, unlike Jason, which is Java-based, the framework is implemented using a language commonly used in the development of embedded systems. This will ensure that low-level abstractions can be easily implemented and hardware is managed wisely. Good candidates are the C and C++ programming languages.

The main reasoning cycle of the agent, where beliefs are updated, events are processed, goals are reviewed, and intentions are adopted, must be kept, as this is an essential feature of BDI agents and what distinguishes them from other computational systems.

Jason employs some elements of logic programming on the agents' knowledge representation and reasoning. These elements include predicates and unification. Due

to the incremental development methodology, it is recommended that the framework first provides support for simpler instructions, such as propositions. Support for predicates can be added later – the memory management and processing load of predicates, which can assume a multiple ranges of types of values, is a big challenge for embedded platforms and can lead to potential performance issues or incompatibility of the framework with the target platforms.

Since the unification algorithm in plans and rules can take a large amount of processing [Junior 2015, Miller and Esfandiari 2021], it is recommended that the unification operators supported by the framework are simple, as complex algorithms will use too much processing time to evaluate plans and goals. Likewise, belief update and action functions should be simplified, so the agent does not spend too much processing updating beliefs or performing actions in the environment.

The internal data structures of the framework should be optimized to reduce memory usage and program size. For example, the framework can use integer variables to represent beliefs, even though the programmer provides these as text values. In addition, all internal data structures, such as the event and intention queues, must have limited size, and memory allocation should be static to minimize the risk of having the agent running out of memory due to dynamic allocation of memory.

Edge cases for full data structures must be taken into consideration and designed so that the risk of dropping a running intention is minimized. For example, if a belief update event results in adopting an intention and the intention queue is full, the agent should discard the new intention instead of replacing one of the existing (and running) intentions with it. Otherwise, the agent will stop executing an ongoing plan and may leave the state of the world in an unexpected state. For example, if the agent is embedded in a vehicle, the vehicle may stop moving abruptly due to the intention related to movement being dropped to process a new intention. However, it is important to note that this behavior can be adapted and improved with the addition of priorities when handling intentions, as some intentions can be more important than others.

4.3. Requirements of the Implemented Agents

Unlike the Jason interpreter, the framework should allow implementing fully-compiled agents for execution. Agent code in AgentSpeak format is parsed and converted into corresponding C/C++ code for compilation, avoiding iterative parsing of the agent code and allowing better performance.

Because each hardware platform has distinct I/O interfaces, the functions to update beliefs and act in the environment should be provided to the framework by the programmer, since adding internal actions to the framework will increase the size of the program and can raise incompatibility for multi-platform support.

It is also important to note that the agent reasoning cycle must be considered when implementing functions to update beliefs and act in the environment: hardware features such as interruptions must be avoided on these functions, as they can interfere with the reasoning cycle and break agent functionalities.

Lastly, programmers should also have the option to configure the size of the internal data structures of the agent, such as the event and intention queues. Because memory

is limited and recursion functions are allowed, events are constantly generated, and plans can be stacked during execution, the size of the internal data structures can vary depending on implementation and usage. Table 1 summarizes the requirements and proposals from this section.

Requirement	Proposal
Development Methodology	Incremental
Platform Support	Multi-Platform
Programming Language for Framework Development	C and/or C++
Programming Language for Agent Implementation	AgentSpeak
Main AgentSpeak Features Supported in Initial Versions	Propositions Simple Unification Algorithms
Memory Allocation	Static
Size of Internal Data Structures	Configurable in the Framework
Internal Methods	No Internal Methods
Belief Update and Action Functions	Provided by the Agent Programmer
Hardware Interruptions	Not Supported

Table 1. Summary of Framework Requirements

5. Conclusion

We discuss the implementation challenges and design considerations for creating a framework that allows implementing BDI agents in common low-cost hardware used in embedded systems.

Although the incremental development methodology suggested may result in a small set of features being added to the framework at times, the ability to be able to constantly review and evaluate which features should be added to the framework and how they can be implemented ensures that multi-platform support can be reached more easily.

In addition, our insights about the framework's requirements aim to tackle one of the main challenges for embedding agents; the limited hardware resources, combined with the advanced features of the BDI architecture, make it difficult to implement complex reasoning algorithms in simple hardware.

We believe that taking Jason as basis for software architecture and features, it is possible to implement the framework proposed, given that the strategies suggested in this paper are followed.

References

Aliyuda, A. (2016). Towards the design of cyber-physical system via multi-agent system technology. In *International Journal of Scientific & Engineering Research*, volume 7.

Barros, R., Heringer, V., Lazarin, N. M., and Moraes, L. (2014). An agent-oriented ground vehicle's automation using jason framework. pages 261–266.

Berger, A. S. (2002). *Embedded Systems Design: An Introduction to Processes, Tools, and Techniques*. CMP Books. Taylor & Francis.

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

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

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

Bratman, M. (1987). *Intention, plans, and practical reason*. Harvard University Press, Cambridge, MA.

Bucheli, S., Kroening, D., Martins, R., and Natraj, A. (2015). From agentspeak to c for safety considerations in unmanned aerial vehicles. pages 69–81.

Horvath, I. and Gerritsen, B. (2012). Cyber-physical systems: Concepts, technologies and implementation principles.

Junior, M. F. S. (2015). Melhorando o desempenho de agentes bdi jason através de filtros de percepção.

Lazarin, N. M. and Pantoja, C. (2015). A robotic-agent platform for embedding software agents using raspberry pi and arduino boards.

Marwedel, P. (2010). *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems*. Springer Publishing Company, Incorporated, 2nd edition.

Mascardi, V., Demergasso, D., and Ancona, D. (2005). Languages for programming bdi-style agents: an overview. pages 9–15.

Miller, J. and Esfandiari, B. (2021). Analyzing the execution time of the jason bdi reasoning cycle. 9th International Workshop on Engineering Multi-Agent Systems.

Park, K.-J., Zheng, R., and Liu, X. (2012). Cyber-physical systems: Milestones and research challenges. *Comput. Commun.*, 36:1–7.

Rajkumar, R., Lee, I., Sha, L., and Stankovic, J. (2010). Cyber-physical systems: The next computing revolution. In *Design Automation Conference*, pages 731–736.

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.

Santos, F. R. (2015). Avaliação do uso de agentes no desenvolvimento de aplicações com veículos aéreos não-tripulados.

Wooldridge, M. (2009). *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd edition.

Wooldridge, M. and Jennings, N. R. (1995). Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2):115–152.

Um Protocolo para Comunicação entre Sistemas Multi-Agentes Embarcados

Nilson Mori Lazarin[1], **Carlos Eduardo Pantoja**[1], **Vinicius Souza Jesus**[2]

[1]Centro Federal de Educação Tecnológica (CEFET/RJ)
– Rio de Janeiro – RJ – Brazil

[2]Universidade Federal Fluminense (UFF)
– Rio de Janeiro – RJ – Brazil

{nilson.lazarin,pantoja}@cefet-rj.br, souza.vdj@gmail.com

Resumo. *As opções de comunicação entre SMA baseados em BDI embarcados em plataformas robóticas atualmente são através de comunicação serial para sistemas fechados ou através de integração com redes de para sistemas abertos. Este trabalho apresenta um protocolo de comunicação não orientado à conexão que possibilita comunicação unicast, multicast ou broadcast em uma rede de difusão através de dispositivos radiofrequência (RF), possibilitando a comunicação SMA Embarcado em situação de emergência ou ambientes sem infraestrutura de comunicação prévia.*

Abstract. *The communication options between BDI-based SMA embedded in robotic platforms today are through serial communication for closed systems or integration with open systems networks. This work presents a non-connection-oriented communication protocol that enables unicast, multicast, or broadcast communication in a diffused network through radio frequency devices, enabling Embedded SMA communication in an emergency or environment without prior communication infrastructure.*

1. Introdução

Agentes Inteligentes são entidades autônomas e pro-ativas construídas em software ou hardware capazes de interagirem entre si através da troca de mensagens. Um Sistema Multi-agentes (SMA) é um conjunto de agentes com os objetivos comuns ou conflitantes sob determinada área de um ambiente real ou simulado [Wooldridge 2009]. Já um SMA Embarcado [Souza de Castro et al. 2020] é um conjunto de agentes em um SMA atuando em prol do gerenciamento de um dispositivo físico onde estão inseridos, interfaceando sensores e controlando atuadores. Um SMA — Embarcado ou não — pode ser Fechado, quando seus agentes não podem se mover para outros sistemas ou plataformas, ou Aberto quando é permitido a saída e entrada de agentes no sistema. Esta mesma classificação pode ser aplicada para a comunicação: se agentes em um SMA podem se comunicar com agentes de outros SMA, utilizando alguma infraestrutura de comunicação, é dito que a comunicação é aberta, e quando os agentes só podem se relacionar com agentes do seu mesmo sistema, a comunicação é dita Fechada. Dessa forma, podemos ter SMA totalmente fechados, que não permite a comunicação externa com outros agentes de outros SMA nem a existência de agentes móveis na sua composição, assim como a existência de SMA totalmente abertos, permitindo o inverso [Pantoja 2019].

Existem diversos domínios que exploram a aplicação de agentes e SMA embarcados para possibilitar o tratamento da informação e uma resposta pro-ativa na ponta de dispositivos individuais, em redes de sensores, redes veiculares e redes IoT [Jesus et al. 2018], [Pantelimon et al. 2019], [Venglář et al. 2020]. Contudo, alguns destes domínios possuem dependências de infraestrutura de rede comunicação (i.e., TCP/IP, middleware IoT, redes 3G), o que pode inviabilizar parcialmente a funcionalidade de SMA embarcados tornando-os totalmente fechados, seja momentaneamente ou permanentemente.

Por exemplo, em um cenário onde carros autônomos — embarcados com SMA — estão em uma rodovia distante sem infraestrutura de telecomunicação, onde ocorra um acidente em determinado local desta rodovia, estes veículos não seriam capazes de se comunicar, a não ser que estejam dotados de atuadores próprios que permitam a comunicação a partir de difusão de dados. Ainda sim, mesmo com a possibilidade de envio dos dados por difusão, estes poderiam ser recebidos por todos os veículos. Caso existam nessa rodovia veículos dos tipos civis e de segurança pública (polícia e bombeiro), é interessante que eles possam se comunicar individualmente (*unicast*), em grupos (*multicast*) ou por difusão (*broadcast*).

Atualmente, poucos trabalhos exploram a embarcação de SMA em plataformas robóticas, especialmente quando se trata do modelo Belief-Desire-Intention (BDI) [Bratman 1987], visto que o processo de percepção, processamento de crenças e intenções podem gerar atrasos indesejados se não tratados adequadamente [Stabile et al. 2018]. Alguns frameworks e soluções atuais para este fim permitem a integração de SMA Embarcados totalmente fechados através de interface serial [Lazarin and Pantoja 2015], [Pantoja et al. 2016], ou integração com redes de dados [Pantoja et al. 2018], [Manoel et al. 2020]. Contudo, ou tais soluções são SMA totalmente fechados ou dependentes de infraestrutura de comunicação.

O objetivo deste trabalho é propor um protocolo de comunicação para a criação de SMA embarcados utilizando agentes BDI de forma que seja possível a difusão de informação/comunicação para outros SMA embarcados em distâncias reduzidas. O protocolo poderá ser utilizado em conjunto com outras infraestruturas de comunicação (3G, IoT, etc.) ou quando não existir outros métodos disponíveis para comunicação, permitindo assim que um SMA totalmente Fechado possa transmitir e enviar informações a outros dispositivos.

Para isso, a interface serial Javino [Lazarin and Pantoja 2015], será estendida para permitir comunicação ponta a ponta usando Radio Frequência (RF) através de um protocolo para envio de mensagens de *broadcast*, *multicast* e *unicast*. Também será utilizado o JaCaMo *framework* [Boissier et al. 2013] para a criação do SMA e agentes Argo [Pantoja et al. 2016] para permitir a criação de SMA embarcados com capacidade de interfaceamento de *hardware*. Será apresentado uma prova de conceito com três protótipos de carros para cada um dos modos de comunicação. Este trabalho está organizado da seguinte forma, na Seção 2 os detalhes do protocolo proposto e de sua implementação serão mostrados; a prova de conceito será demonstrada na Seção 3. Por fim, nas Seções 4 e 5, serão apresentados os Trabalhos Relacionados e a Conclusão respectivamente.

2. O Protocolo de Comunicação de SMA Embarcados

Em dispositivos com sistemas embarcados, incluindo SMA, a comunicação é ponto importante pois permite que dados obtidos a partir da leitura sensores possam ser encaminhados para outros dispositivos. Da mesma forma, através de um meio de comunicação, dados também podem ser recebidos de outros dispositivos. Caso contrário, qualquer sistema embarcado, mesmo que dotado de capacidade cognitiva e autônomo, dependeria apenas de suas próprias observações e conclusões para a tomada de decisões e aprendizado. Ao se estabelecer um meio de comunicação, o aprendizado e a tomada de decisão poderá também considerar informações obtidas de um coletivo de sistemas embarcados. Um SMA Embarcado é responsável pelo sensoriamento de um ambiente real através de um conjunto de sensores, pelo processo de análise dos dados capturados e deliberação, e por agir no ambiente usando seus atuadores, além da comunicação com outros dispositivos. Para isso, é necessário que estes dispositivos adotem uma arquitetura de interligação de seus diversos componentes físicos e de fluxo de informação [Matarić 2007].

Neste trabalho, será adotada uma arquitetura física e lógica que permite interligação de componentes de *hardware* para que seja possível habilitar um fluxo de percepções direta de sensores para agentes em um SMA cognitivo [Pantoja et al. 2016]. Esta arquitetura será modificada fisicamente para permitir a comunicação com outros dispositivos com SMA embarcados e logicamente para permitir que mensagens e percepções cheguem a estes SMA. A arquitetura tem quatro camadas lógicas:

a) **Reasoning**: É a camada responsável pela cognição do dispositivo. Um SMA é adotado para interfacear os sensores e atuadores disponíveis através de uma interface serial recebendo percepções ou mensagens de outros dispositivos, deliberando e respondendo com ações a serem executadas nos atuadores ou mensagens para outros dispositivos.
b) **Serial**: É a camada que faz interfaceamento entre os dados que trafegam entre o SMA e o hardware adotado. A comunicação se dá através de um protocolo de troca de informações entre a porta serial de uma plataforma microprocessada (i.e., computadores e placas) e microcontroladores. O protocolo de comunicação entre SMA embarcados é implementada nessa camada, onde as mensagens serão captadas e enviadas aos demais dispositivos através de acesso ao hardware de RF disponível.
c) **Firmware**: É o programa embarcado no microcontrolador onde os comportamentos de acesso aos sensores e atuadores são programados. As percepções são enviadas ao agente que faz o interfaceamento uma vez por ciclo ou quando este a requisita. As ações para controle de atuadores e de mensagens são enviadas de acordo com a necessidade do agente.
d) **Hardware**: São todos os sensores e atuadores disponíveis no dispositivo e que estão conectados ao microcontrolador escolhido. Nessa camada inclui-se o sensor de RF para a comunicação com os demais dispositivos. São estes sensores e atuadores que produzem as informações que serão transformadas em percepções e recebem as mensagens que serão enviadas ao SMA e outros dispositivos.

Fisicamente, este trabalho embarca o SMA em computadores microprocessados, podendo ser desde um mini computador (i.e., Raspberry) até um computador tradicional, desde que sejam capazes de manter um sistema operacional. Os sensores e atuadores são conectados a uma plataforma microcontrolada (i.e., Arduino, ATMEGA) que utilizará a comunicação serial entre o sistema operacional e o microcontrolador através de um

middleware ou interface serial (i.e., Javino) para a troca de mensagens e informações. Para que o protocolo proposto possa ser utilizado na prática é preciso que o dispositivo com o SMA Embarcado seja construído a partir de uma arquitetura conforme a especificada acima e que adote atuadores RF em sua composição para a comunicação. A arquitetura pode ser vista na Figura 1.

Apesar da arquitetura especificar um determinado microcontrolador na camada do *firmware*, ao adotar o Javino como interface serial, é possível manter a independência entre as tecnologias utilizadas em alto nível das camadas de *firmware* e *hardware*, podendo o microcontrolador ser trocado sem afetar o processamento em alto nível, assim como a linguagem do SMA. Para isso, o projetista deverá disponibilizar para o microcontrolador selecionado, a implementação adequada do protocolo proposto neste trabalho e utilizar alguma linguagem com suporte ao Javino.

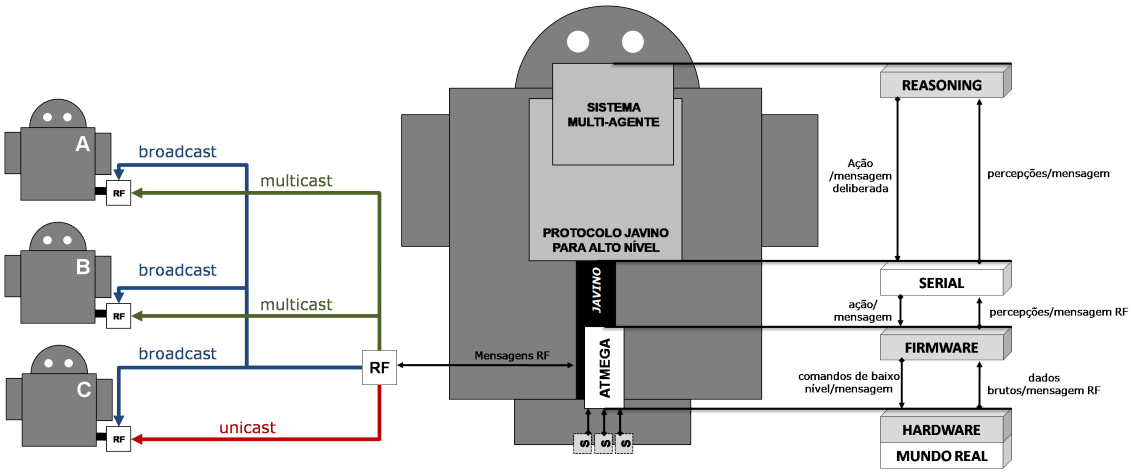


Figura 1. A arquitetura física e lógica para um SMA Embarcado considerando o protocolo proposto.

Quando um dispositivo envia uma mensagem para o outro dispositivo, um agente do SMA que esta enviando a mensagem, transfere o conteúdo da mensagem através da interface serial usando o Javino em linguagem de alto nível, que redireciona para o microcontrolador através da porta serial. A mensagem é recebida pelo Javino em baixo que nível, que envia a mensagem com as informações de destino. Uma vez que o dispositivo de destino recebe uma mensagem endereçada a ele captada pelo RF, essa mensagem é tratada pelo Javino e enviada a um agente do SMA através da porta serial. O agente ficará a cargo de tratar a mensagem uma vez que esta chegar.

2.1. Comunicação Ponta a Ponta

O protocolo apresentado neste trabalho visa permitir a comunicação entre SMA Embarcados através de transmissor e receptor de radiofrequência (RF) para ambientes sem infraestrutura de comunicação. A decisão de envio de uma mensagem parte da deliberação de um agente pertencente ao SMA de origem, que encaminha essa mensagem para o hardware que controla o RF, responsável pela difusão na rede.

O protocolo possibilita a troca de mensagens *Unicast* e *Multicast* na rede de difusão, dessa forma, um SMA embarcado poderá enviar informações para outro individu-

almente, para um grupo ou para todos os no alcance de transmissão, utilizando um dos três modos de envio:

- ***Unicast***: mensagem enviada de um dispositivo para um outro dispositivo específico dentro do alcance de transmissão.
- ***Multicast***: mensagem enviada de um dispositivo a qualquer outro inscrito em determinado grupo de dispositivos dentro do alcance de transmissão.
- ***Broadcast***: mensagem enviada de um dispositivo a qualquer outro no alcance da transmissão.

O hardware controlador de RF nos dispositivos no alcance da difusão irá analisar a mensagem recebida, conforme definido na Figura 2 e armazenar no *buffer*. Uma vez que for recebida uma transmissão, o cabeçalho da mensagem será analisado. Caso seja uma mensagem válida (formato esperado pelo protocolo), será verificado o endereço de destino. São aceitas mensagens do tipo: *Broadcast*; Mensagens *multicast* destinadas a um dos grupos em que o dispositivo esteja inscrito; Ou mensagem *unicast* destinada a uma de suas identificações na rede. Por fim, somente será armazenada no buffer a mensagem que for considerada completa, através da comparação do valor no campo tamanho da mensagem, informado no cabeçalho, e o tamanho da mensagem recebida.

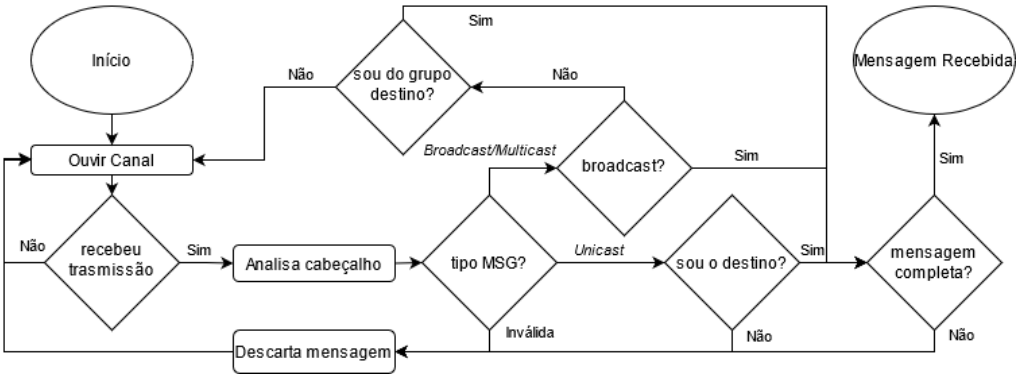


Figura 2. Recebimento de mensagem no meio de difusão.

O protocolo proposto utiliza endereçamento de 24 bits. Os 12 bits mais significativos identificam o grupo e os 12 bits menos significativos representam o membro do grupo. O formato do endereço é apresentado na Figura 3a. O endereço é representado utilizando Alfabeto Base 64 [Josefsson 2006], dessa forma podem ser utilizados caracteres de *A* à *Z*, maiúsculo e minúsculo e os caracteres + e */*.

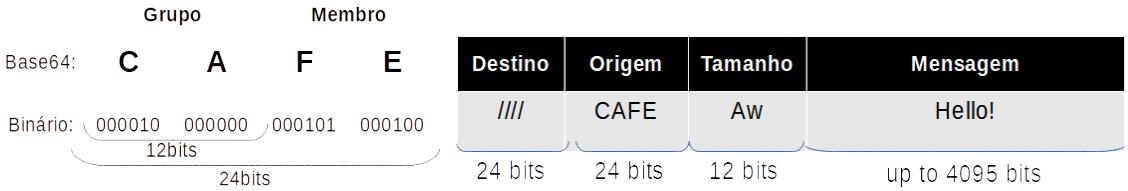
É possível identificar até 16.252.928 dispositivos em 3968 grupos de até 4096 membros cada, iniciado em *AAAA* até *9///*. Os endereços especiais *//AA* até *///+* são utilizados para comunicação *multicast*, um endereço *multicast* inicia com // seguido de dois caracteres que representam o grupo destinatário da comunicação. O endereço especial *////* indica uma comunicação *broadcast*. O endereço especial ++++ indica uma comunicação serial para acesso a hardware. A faixa de endereçamento é apresentada na Tabela 1.

O cabeçalho da mensagem do protocolo proposto possui 3 campos. Endereço de destino com 24 bits, representando um endereço de dispositivo em Alfabeto Base 64; Endereço de origem com 24 bits, representando um endereço de dispositivo em Alfabeto

FAIXA DE ENDEREÇOS										
Endereçável	A	A	A	A	até	9	/	/	/	3968 grupos [AA até 9/] 4096 membros por grupo
Reservado	+	A	A	A	até	+	+	+	9	para uso futuro
Especial	+	+	+	+	comunicação serial acesso ao hardware					
Reservado	+	+	+	/	até	/	+	/	/	para uso futuro
Multicast	/	/	A	A	até	/	/	/	+	formato //[Grupo]
Especial	/	/	/	/	comunicação *broadcast* na rede de difusão					

Tabela 1. Faixa de endereços

Base 64; e Tamanho da mensagem com 12 bits, representado em Alfabeto Base 64, indicando o tamanho do campo Mensagem em bits. A Figura 3b apresenta uma mensagem *broadcast* originada pelo membro FE do grupo CA.



(a) Endereço **(b)** Formato da mensagem

Figura 3. Endereço e formato da mensagem

2.2. Comunicação Agente-Controlador

Para que um agente envie uma mensagem de um SMA embarcado em um dispositivo para outro agente em outro SMA embarcado, é preciso que seja informado três campos obrigatórios: o SMA de destino, a força ilocucionária e a mensagem em si. O destinatário será composto pelo endereço de *hardware* do SMA em base 64 bits. Contudo, é necessário endereçar a mensagem a um agente específico para que a mensagem seja tratada adequadamente visto que o SMA não é capaz de processar mensagens sem a figura de um agente. Dessa forma, é estabelecido que um SMA possua um Agente Difusor responsável por tratar apenas as mensagens vindas de RF. Este agente poderá ter qualquer nome, mas terá anotada em sua crença ou intenção recebida o endereço em base 64 do SMA remetente. Neste primeiro momento, apenas serão consideradas as forças ilocucionárias *tell*, *untell*, *achieve*, e *unachieve* por terem um tamanho de mensagem menor do que as *tellHow* e *AskHow*, e dada a limitação do campo da mensagem do protocolo e do meio de difusão, além de não necessitarem de respostas ao remetente.

Um agente Difusor envia uma mensagem fazendo a chamada a uma ação interna chamada de *difuse* criada especificamente para enviar mensagens através de difusão em RF usando o protocolo proposto. Primeiramente, quando esta ação é chamada, o agente codifica a mensagem separando os campos endereço de *hardware* do remetente, a força ilocucionária e a mensagem por ponto-e-vírgula e encaminha através do Javino que adiciona o endereço do destinatário ao preâmbulo e calcula o tamanho total da mensagem. Em baixo nível, no microcontrolador, a mensagem é difundida com o endereço identificado

no preâmbulo uma vez que esta não está endereçada para ação em atuadores ou captura de percepção ou de mensagens (endereço de destino ++++). Quando o *hardware* do SMA de destino recebe a mensagem, ele a armazena em um *buffer* temporário até que o agente Difusor do destinatário solicite as mensagens recebidas. Como agentes BDI possuem um ciclo de raciocínio bem definido, não é possível o microcontrolador enviar diretamente a mensagem ao SMA. É preciso que haja a sincronização entre o ciclo de raciocínio do agente e o recebimento de mensagens.

Para isso, foi criada uma extensão da arquitetura customizada de agentes ARGO para que seja possível se comunicar com o *hardware* através da porta serial para solicitar percepções e mensagens difundidas. Os agentes ARGO fazem uma solicitação de percepções (*getPercepts*) no passo de captura das percepções do ciclo BDI. Igualmente, agentes Difusores farão a solicitação de mensagens (*getMessages*) no passo onde o agente faz a checagem de recebimento de mensagens (*check mail*). Caso o agente tenha alguma mensagem para ser recebida, ele a decodificará, para criar as mensagens entendíveis para o agente. Perceba que todo agente Difusor também é um agente ARGO, mas o contrário não é verdade, ou seja, os agentes Difusores são uma extensão dos agentes ARGO com uma capacidade adicional de se comunicarem com RF. A Figura 4 mostra o ciclo de raciocínio do agente Difusor.

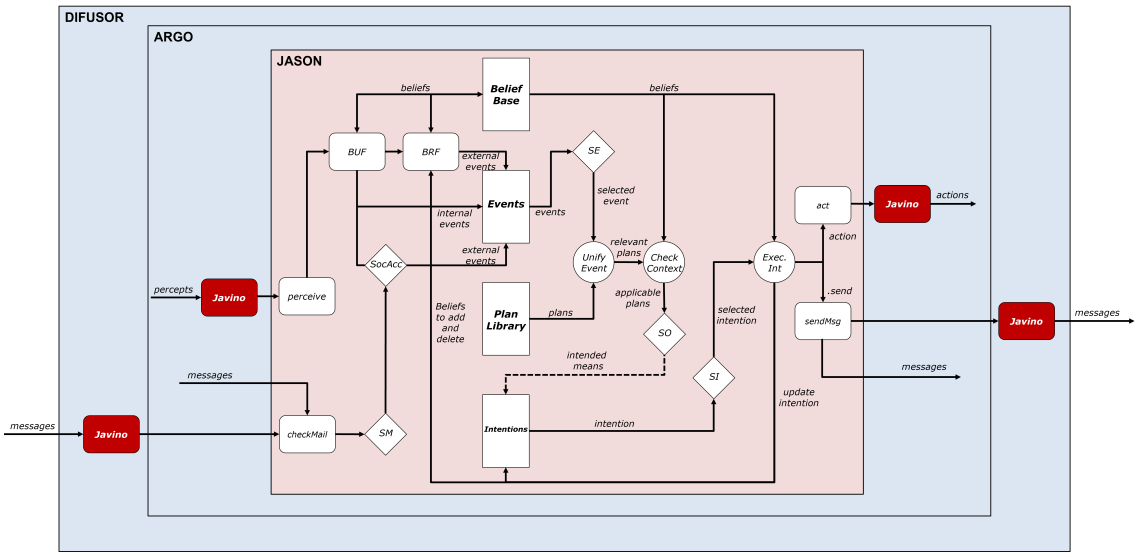


Figura 4. O ciclo de raciocínio de um Agente Difusor.

Para que a abordagem proposta funcione, é necessário que o SMA possua no máximo um agente Difusor para que o endereçamento da mensagem seja centralizado em apenas uma entidade. A existência de mais de um agente Difusor não inviabiliza o funcionamento do SMA, contudo, vai ser gerado uma competição pela aquisição das mensagens, obrigando ao programador e projetista do SMA embarcado a reprodução dos planos nos agentes ou a comunicação caso uma mensagem não seja de responsabilidade de quem a capturou. Além disso, é recomendável que cada SMA adote pelo menos um microcontrolador dedicado para a comunicação RF para atuar separadamente como um suporte caso a infraestrutura de comunicação ou o microcontrolador principal falhem. Novamente, a utilização de um agente Difusor para gerenciar outros atuadores e sensores além do RF não inviabiliza o projeto e não interfere na captura das percepções nem das mensa-

gens, pois estas ocorrem em passos diferentes do ciclo de raciocínio. A recomendação se dá para uma melhor organização de tarefas entre os agentes pertencentes ao SMA. A limitação deste tipo de aplicação centralizada é que caso o agente Difusor seja morto ou vá para outro SMA é necessário ter um mecanismo para preservação de seus conhecimentos [Souza de Jesus. et al. 2021].

3. Prova de Conceito

Para avaliar o procolo proposto, é apresentada uma prova de conceito considerando um cenário de uma rodovia onde trafegam veículos civis e de segurança pública, como bombeiro e polícia, que são controlados por SMA embarcados sem acesso a nenhuma infraestrutura de comunicação, exceto a comunicação por RF existente em cada veículo. Em caso de algum acidente, o carro que identificar o acidente ou o que sofrer o acidente (caso seus atuadores não tiverem sido afetados) podem enviar uma mensagem (*broadcast*) para os demais veículos na proximidade para que estes fiquem atentos ou simplesmente parem devido ao acidente ocorrido. Além disso, a mensagem pode ser difundida para veículos de segurança pública, que poderão se comunicar com indivíduos do mesmo grupo (*unicast* e *multicast*) ou para demais veículos civis (*unicast* e *broadcast*).

Para isso, foram implementados 4 SMA embarcados usando o Jason e a extensão de agentes ARGO proposta, rodando em 3 *laptops* distintos e 1 em uma Raspberry representando um veículo cada. Além disso, cada um estará vinculado a uma placa microcontrolada com o ATMEGA328 e módulos TX/RX — RF 433Mhz, além de LEDS para sinalização de movimentação. Um dos computadores representará um veículo civil que sofreu um acidente e enviará uma mensagem de *broadcast* solicitando apoio aos demais. Um outro computador e a Raspberry serão veículos de segurança pública em um mesmo grupo que receberão a mensagem de acidente e se comunicarão para alinhar um possível resgate *multicast* e enviarão uma mensagem de *acknowledge* informando que o socorro está a caminho *unicast*. Por fim, o último computador representará um veículo civil que apenas receberá a solicitação de apoio do acidentado porém não fará nada a respeito. A utilização de computadores ou *laptops* apenas adicionam mais poder de processamento ao SMA, não interferindo na comunicação entre o hardware e o software ou com a difusão por RF.

Cada SMA terá apenas um agente ARGO interfaceando o hardware com planos específicos para deliberação dependendo de em qual veículo ele esteja embarcado. No caso do veiculo acidentado, o agente enviará uma mensagem de *broadcast* utilizando a nova ação interna criada informando que ele precisa de apoio. No caso do veículo civil que receber essa mensagem, ele apenas informará que a mensagem foi recebida acendendo um LED. Um dos veículos de segurança pública receberá a mensagem de solicitação de apoio e enviará uma mensagem de *multicast* para todos os integrantes do grupo, que informarão estarem de pronto atendimento acendendo um LED. Este mesmo veículo enviará a mensagem de *acknowledge* para o veículo acidentado informando que o apoio já está a caminho. A Tabela 2 exibe a identificação de cada veiculo através do protocolo, os planos dos agentes e o comportamento esperado antes e depois da execução.

O execução do sistema retornou o comportamento esperado de todos os SMA embarcados e do protocolo, contudo ainda é preciso aplicar o protocolo em SMA mais robustos e com um nível de complexidade maior, inserindo mais agentes em cada sis-

Veículo	Id	Comportamento Esperado	Planos	Comportamento Obtido
Acidentado	Brok	Solicita apoio	askForHelp(Everyone) wait : help(coming)	LED aceso após mensagem de acknowledge
Bombeiro 1	CBv1	Recebe solicitação de apoio e contata membros do grupo CB	contactTroop: help[Brook]	Mensagem de unicast para o Acidentado e de multicast para o grupo CB enviadas
Bombeiro 2	CBv2	Aguarda contato do Bombeiro 1	wait : accident	LED aceso após mensagem de multicast do Bombeiro 1
Civil	NONE	Nenhum	help(coming)[Brook]	LED aceso após mensagem de broadcast do Acidentado

Tabela 2. Papéis e comportamentos

tema de forma que seja possível medir o custo de processamento mínimo (utilizando o mínimo de componentes da arquitetura possível) e de execução do domínio da aplicação. De toda forma, o resultado esperado foi atingido em um tempo satisfatório considerando se tratar de uma arquitetura alternativa para comunicação dada a indisponibilidade de outros meios, tendo todo os veículos deliberado em menos de 2s. Além disso, é importante embarcar todos os SMA em placas como a Raspberry ou similares para garantir uma análise entre a utilização de computadores e placas microprocessadas. Um critério para a utilização de placas ou computadores pode ser o tamanho do dispositivo em si. Por exemplo, em um carro real é possível utilizar um computador de bordo similar ou mais potente do que um computador com configurações atuais mas, conforme dito anteriormente, a principal diferença se dará em capacidade de processamento para o SMA embarcado, não interferindo na interconexão de *hardware* e *software*.

Alguns pontos relacionados à segurança da transmissão não são abordados no protocolo proposto. A difusão das mensagens trafegam em claro, podendo permitir que mensagens de *unicast* ou *multicast* sejam capturadas por outros veículos não pertencentes a um determinado grupo ou qualquer outro dispositivo que esteja no alcance de transmissão e seja capaz de captar dados de RF. Ou ainda, um agente mal intencionado enviar mensagens forjando sua identidade (se passando por outro dispositivo) durante a transmissão.

Para uma comunicação segura pode-se implementar algum algoritmo de criptografia simétrica no dispositivo responsável pela transmissão RF, dada a facilidade de implementação e menor custo computacional e realizar a troca segura das chaves de transmissão através do algoritmo Diffie-Hellman. Assim, o emissor antes de transmitir a mensagem deverá derivar a chave de transmissão utilizando sua chave privada e a chave pública do destinatário. O receptor, por sua vez, ao receber a mensagem irá derivar a chave de transmissão através de sua chave privada e a chave pública da origem. Dessa forma estará garantido o sigilo, a integridade, a autenticidade e a irretratabilidade da mensagem.

Por fim, existe uma limitação de endereçamento de dispositivos na rede ad-hoc, contudo o objetivo é atuar em áreas menores e não em contextos de cidades inteligentes envolvendo grandes quantidades de dispositivos visto que o protocolo é uma forma de comunicação quando não houver um outro meio de comunicação disponível.

4. Trabalhos Relacionados

Na literatura de SMA embarcados existem diversas aplicações que exploram troca de informações entre SMA distintos. Os agentes *Communicator* [Pantoja et al. 2018] são agentes BDI estendidos capazes de se conectar a uma rede e enviar/receber informações

de outros SMA. Além disso, esses agentes *Communicator* posteriormente foram utilizados para a criação de protocolos bioinspirados para a transferência de agentes entre SMA distintos [Jesus et al. 2018]. Com isso, diferentes SMA não só conseguiam se trocar informações pela rede, mas como também, enviar e receber agentes.

Atualmente, foi percebida um crescente número de aplicações de SMA embarcados em drones. O survey [Pantelimon et al. 2019] reúne diversas aplicações de SMA embarcados em drones buscando identificar nestes sistemas como é feito o gerenciamento do controle de missão e as estratégias de comunicação para implantações desses drones. Os principais resultados foram obtidos através de duas estratégias: centralizada e descentralizada.

Além disso, o trabalho [Venglář et al. 2020] apresenta uma implementação de rede *mesh* para Wi-Fi móvel, considerando que nem todos os dispositivos são habilitados para essa rede Wi-Fi *mesh*. Com isso, foi apresentado um novo método de comunicação para SMA embarcados com resultados que mostram que é possível utilizar esse tipo de rede, ganhando em alcance operacional.

Portanto, todos os trabalhos descritos acima utilizam um tipo de infraestrutura de rede (i.e., TCP/IP, middleware IoT, redes 3G) para a troca de informações. Em caso de falha, os SMA embarcados perdem o canal de comunicação e não possuem nenhuma medida de tratamento. Sendo assim, este trabalho apresenta um protocolo de comunicação secundário que pode ser utilizado, por exemplo, em caso de falha da infraestrutura de rede ou até mesmo para a comunicação de dispositivos em curta distâncias.

5. Conclusão

Este trabalho apresentou um protocolo que permite a comunicação entre dois SMA embarcados através de RF para quando não existe nenhum outro meio de comunicação disponível no momento e local onde estejam os dispositivos. O protocolo prevê três modos de comunicação em nível de *hardware* que auxilia na organização e troca de mensagens em grupos, comunicação individual e envios de mensagens de *broadcast*. Esses modos de comunicação permitem manter um nível mínimo de interação entre os dispositivos e os SMA embarcados para que estes não se tornem totalmente fechados em uma situação onde não tenha disponível uma estrutura de comunicação para IoT, 3G, etc. De toda forma, ainda é uma possibilidade e escolha do projetista, adotar o protocolo mesmo que haja uma estrutura de comunicação disponível, não sendo estas duas conflitantes caso coexistam.

Para permitir a criação do protocolo diversas tecnologias foram utilizadas e estendidas. Primeiramente, por se tratar de comunicação em nível de Hardware, o modo de endereçamento do Javino foi modificado para permitir a criação de grupos e endereçamento individual de dispositivos. Essa extensão utiliza uma base de endereçamento em Base64, diferente da base anterior, contudo não há impacto no funcionamento de soluções que adotaram a versão anterior. Como a modificação foi internamente e transparente ao usuário, foi dedicado um endereço específico (++++) para comunicação entre linguagem de alto nível e hardware.

Da mesma forma, o agente ARGO foi estendido para criar um novo agente chamado Difusor, onde foi adicionado no seu ciclo de raciocínio um passo para captura das

mensagens armazenadas no *buffer* do hardware. A opção de se criar um novo agente se de deu para definir bem as responsabilidades dos agentes possíveis de se usar em um SMA embarcado, de forma a não sobrecarregar um ou outro com demais funcionalidades. Um agente Difusor é na prática um agente ARGO com a capacidade de fazer leitura de mensagens. Dessa forma, considerando a aplicabilidade de arquiteturas estendidas de agentes para o JaCaMo, é possível criar agentes de interfaceamento de sensores e atuadores (ARGO), agentes que se comunicam por uma rede IoT (Comunicadores) e agentes de difusão em redes de RF (Difusores).

O protocolo ainda precisa ser testado em cenários mais robustos com uma aplicação mais extensa. Contudo, testes iniciais comprovaram que é possível fazer com que agentes BDI embarcados possam se comunicar como outros agentes embarcados de um SMA distinto de forma efetiva. Como trabalhos futuros, será criado o mecanismo de segurança através de criptografia AES e chaves públicas e privadas, para que as mensagens difundidas no meio não trafeguem em claro. Além disso, novos testes de performance e uma análise mais aprofundada serão apresentadas.

Referências

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). *Intentions, Plans, and Practical Reason*. Harvard University Press.

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

Josefsson, S. (2006). The Base16, Base32, and Base64 Data Encodings. RFC 4648.

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

Manoel, F., Pantoja, C. E., Samyn, L., and de Jesus, V. S. (2020). Physical artifacts for agents in a cyber-physical system: A case study in oil & gas scenario (eeas). In *SEKE*, pages 55–60.

Matarić, M. (2007). *The Robotics Primer*. Mit Press.

Pantelimon, G., Tepe, K., Carriveau, R., and Ahmed, S. (2019). Survey of multi-agent communication strategies for information exchange and mission control of drone deployments. *Journal of Intelligent Robotic Systems*, 95:1–10.

Pantoja, C. E. (2019). *An Architecture To Support the Integration of Embedded Multi-Agent Systems*. PhD thesis, Universidade Federal Fluminense.

Pantoja, C. E., Jesus, V. S., Manoel, F. C. P. B., and Viterbo, J. (2018). A heterogeneous architecture for integrating multi-agent systems in ami systems. *The Thirtieth International Conference on Software Engineering and Knowledge Engineering (SEKE 2018)*.

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

International Workshop on Engineering Multi-Agent Systems, pages 136–155. Springer.

Souza de Castro, L., Manoel, F., Souza de Jesus, V., Pantoja, C., Borges, A., and Vaz Alves, G. (2020). Integrando sistemas multi-agentes embarcados, simulação urbana e aplicações de iot. In *XIV Workshop Escola de Sistemas de Agentes, seus Ambientes e Aplicações (WESAAC 2020)*.

Souza de Jesus., V., 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. INSTICC, SciTePress.

Stabile, M. F., Pantoja, C. E., and Sichman, J. S. (2018). Experimental analysis of the effect of filtering perceptions in bdi agents. *International Journal of Agent-Oriented Software Engineering*, 6(3-4):329–368.

Venglář, V., Králík, J., Chin, H.-L., and Chen, K.-S. (2020). Mesh wi-fi infrastructure for multi-agent robotic system. In *2020 19th International Conference on Mechatronics - Mechatronika (ME)*, pages 1–4.

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

Reengenharia de uma Arquitetura de Gerenciamento de Recursos para Agentes Utilizado Golang

Maria Alice Trinta[1], **Fabian C. B. Manoel**[1], **Carlos Eduardo Pantoja**[1]

[1]Centro Federal de Educação Tecnológica (CEFET–RJ)
20785-220 – Rio de Janeiro – RJ – Brasil

maria.trinta@aluno.cefet-rj.br,fabiancpbm@gmail.com

pantoja@cefet-rj.br

Abstract. *The Resource Management Architecture (RMA) is an architecture that allows the management of multi-agent systems in an IoT network. However, the architecture still cannot be applied on a large scale as it supports a low number of concurrent connections. Therefore, this work presents the reengineering of RMA using a Golang programming language to allow a more significant number of references.*

Resumo. *A Arquitetura de Gerenciamento de Recursos (RMA) é uma arquitetura que permite o gerenciamento de sistemas multiagente em uma rede IoT. Entretanto, a arquitetura ainda não consegue ser aplicada em larga escala pois suporta um baixo numero de conexões simultâneas. Sendo assim, este trabalho apresenta uma reengenharia da RMA utilizando a linguagem de programação Golang, com o objetivo de permitir um maior numero de conexões.*

1. Introdução

A cada dia novas tecnologias surgem afim de criar sistemas que possibilitem o compartilhamento e o acesso a informações de qualquer lugar e momento. Entretanto, muitas enfrentam problemas de escalabilidade e performance, o que pode implicar em limitações quanto ao número de conexões e lentidão. Sendo assim, para resolver estes problemas, é necessário utilizar de meios que possibilitem a otimização do sistema, para que então seja possível a aplicação destes em larga escala, como a interligação de equipamentos em grandes hospitais e hotéis. Pode ser atribuído à um sistema inteligente a capacidade de controlar de maneira autônoma os recursos e a execução de ações em diferentes domínios. Dentre as abordagens para o desenvolvimento de tais sistemas, está a utilização de agentes, que contribuem com um certo nível cognitivo a partir de uma análise de dados para tomada de decisões. Pode-se definir um Agente Inteligente como uma entidade autônoma que pode analisar um ambiente e agir sobre ele. Logo, um Sistema Multiagente (SMA) seria um grupo de agentes inteligentes, organizados para a cooperação ou competição findados a atingir certo objetivo [Wooldridge 2009].

Os sistemas inteligentes podem ser associados à tecnologias de gerenciamento, compartilhamento de informações ou *middlewares* a fim da possibilitar sua utilização em larga escala. Neste caso, a Internet das Coisas (*Internet of Things* - IoT) pode ser aplicada a fim da criação de uma rede que interligue o sistema em diferentes pontos possibilitando a cobertura de grandes áreas, pois, ao estender a internet ao mundo físico, torna

possível o gerenciamento remoto de qualquer dispositivo, o seu rastreamento através da internet, e o compartilhamento de informações [Zhang et al. 2012]. A Arquitetura de Gerenciamento de Recursos (RMA) [Pantoja et al. 2019] é uma arquitetura que possibilita organizar sistemas multiagentes em uma rede IoT afim de permitir o controle remoto e o compartilhamento de informações em larga escala. Entretanto, a RMA possui limitações quanto ao número de dispositivos em conexões simultâneas, o tempo de processamento das mensagens e no tempo de resposta de conexão. Isto faz com que o sistema gere *delays* indesejados, impossibilitando novas conexões e dificultando a ampliação do seu campo aplicação. Atualmente, a arquitetura perde performance ao empregar em torno de 50 objetos inteligentes gerenciados por SMA embarcados.

Portanto, o objetivo deste trabalho é refatorar a RMA para que o número atual de conexões suportadas aumente, assim como sua performance. Tendo em vista que a RMA foi construída utilizando a linguagem de programação Java [Byous 1998], é proposta que ao utilizar a linguagem Go [Donovan and Kernighan 2015], tais problemas podem ser minimizados, uma vez que esta linguagem pode ser compilada três vezes mais rápido que java e possui resultados melhores em testes comparativos entre as duas linguagens no que diz a respeito à performance de concorrência [Togashi and Klyuev 2014]. Com a troca de linguagens na estrutura da RMA, é necessário repensar a maneira como é estabelecida a comunicação entre a IoT e o SMA Embarcado utilizando o JaCaMo, uma vez que anteriormente utilizava-se o *middleware ContextNet* [Endler et al. 2011] que possui suporte somente para a linguagem Java e Lua. Para tal, foi escolhido o NATS, que é um *middleware* orientado a mensagens que possui suporte para as linguagens mais utilizadas incluindo Go, Java, Rubi, Python, entre outras. Além disso, o *middleware*, que possui como princípios a escalabilidade, a performance e a simplicidade, consegue enviar mais de sete milhões de mensagens por segundo [Quevedo 2018]. Além disso, o agente Comunicador responsável pela comunicação, que estava preparado para utilizar o *ContextNet*, deverá ter sua estrutura reorganizada para ser capaz de se comunicar usando o NATS.

Para comprovar o desempenho da RMA-GO, um cenário de testes foi criado para analisar o desempenho do envio de uma mensagem, desde o emissor até o banco de dados que armazena informações dos recursos virtualizados. Os testes foram realizados com múltiplos dispositivos simulados, afim de testar também a capacidade de conexões simultâneas.

Este trabalho está organizado da seguinte forma: na seção 2 é apresentado o referencial teórico, em seguida, na seção 3 é apresentada a metodologia para a construção da RMA-GO. A avaliação experimental será apresentada na seção 4. Em seguida, serão discutidos alguns trabalhos relacionados e por fim, a conclusão.

2. Referencial Teórico

Nesta seção serão apresentados os referenciais teóricos necessários para um entendimento acerca das tecnologias utilizadas durante o desenvolvimento do trabalho. A primeira delas é a Internet das Coisas (*Internet of Things* - IoT) [Bandyopadhyay et al. 2011] que pode ser definida como uma rede que interliga objetos físicos, possibilitando o controle e o compartilhamento de informações a distância. Para estabelecer esta rede é necessário utilizar um *middleware* para IoT, uma vez que este possibilita a comunicação entre meios heterogêneos, nesse caso, entre camadas diferentes em uma mesma arquitetura.

Ambos NATS e *ContextNet* são *middlewares* para IoT que possibilitam a comunicação à distância, porém o *ContextNet* possui suporte apenas para Java e Lua, enquanto o NATS pode ser implementado através de diferentes linguagens de programação. Além disso, o NATS por ser um *middleware* orientado a mensagens, permite a comunicação assíncrona pois possui um fraco acoplamento e permite um número reduzido de conexões [Nilsson and Pregén 2020]. A alteração do middleware vem como uma alternativa para permitir que o processamento dos pedidos de conexão, reconexão, de compartilhamento de informações e de ações a serem executadas em hardware sejam feitas de forma mais eficiente.

A RMA é responsável pela virtualização de componentes para serem consumidos e acessados por clientes de forma remota em uma rede IoT. A RMA é composta de três camadas que se interconectam através da troca de mensagens. Em uma visão *top-down*, na camada de clientes estão localizadas as aplicações que consomem de forma remota as informações de recursos provenientes de dispositivos conectados e virtualizados na Camada de Gerenciamento de Recursos (*Resource Management Layer* - RML), além de ser responsável por enviar ações para serem executadas por algum recurso de dispositivo (um atuador). A RML tem a responsabilidade de gerenciar os pedidos de conexão, o processamento e o armazenamento das informações dos recursos de dispositivos em determinado ambiente configurado pelo usuário. Na camada de dispositivos, encontram-se todos os dispositivos que possuem um sistema embarcado e recursos (sensores e atuadores) para sensoriamento e atuação em determinado ambiente real. Tais dispositivos inteligentes podem empregar SMA embarcados utilizando o JaCaMo [Boissier et al. 2013].

Compreende-se também que um Agente Inteligente é uma entidade física ou virtual capaz de deliberar e tomar decisões que influenciem ou não sobre o ambiente em que estejam inseridos. Assim, um conjunto de agentes é denominado um SMA, onde estes podem estar organizados para atuar e exercer influência sobre um ambiente simulado ou físico para atingir determinado objetivo. Neste trabalho, dispositivos inteligentes utilizam um SMA embarcado para permitir o controle de sensores e atuadores conectados a um microcontrolador, que possa se conectar em uma rede IoT onde a RML esteja disponível, e que possa ser ao mesmo tempo autônomo e possa receber pedidos de ações enviados por clientes que podem ser atendidos ou não (dependendo da deliberação através do conjunto de planos disponibilizado pelo projetista do dispositivo).

Este SMA embarcado traz consigo todo o necessário para a execução de tarefas autônomas, uma vez que pode, através de sensores e atuadores, recolher dados sobre o ambiente, analisá-los e devolver uma resposta por meio de uma atuação sem depender de enviar informações para um servidor ou aplicação centralizadora. Desta forma, é possível aumentar o nível de autonomia e cognição na borda de sistemas (*Edge Computing*) [Khan et al. 2019] através da utilização de agentes que adotam o modelo *Belief-Desire-Intention* (BDI).

3. RMA-GO

Nesta seção será apresentada a metodologia utilizada durante o desenvolvimento da RMA-GO, descrevendo como estão organizadas as camadas após a reformulação, explicando o que está sendo utilizado e como manter a integração entre elas. A começar pela camada de cliente, em seguida a camada de gerenciamento de recursos e, por último,

a camada de dispositivos, que interagem entre si através da troca de mensagens e formam a Arquitetura de Gerenciamento de Recursos, como pode-se ver através da Figura 1.

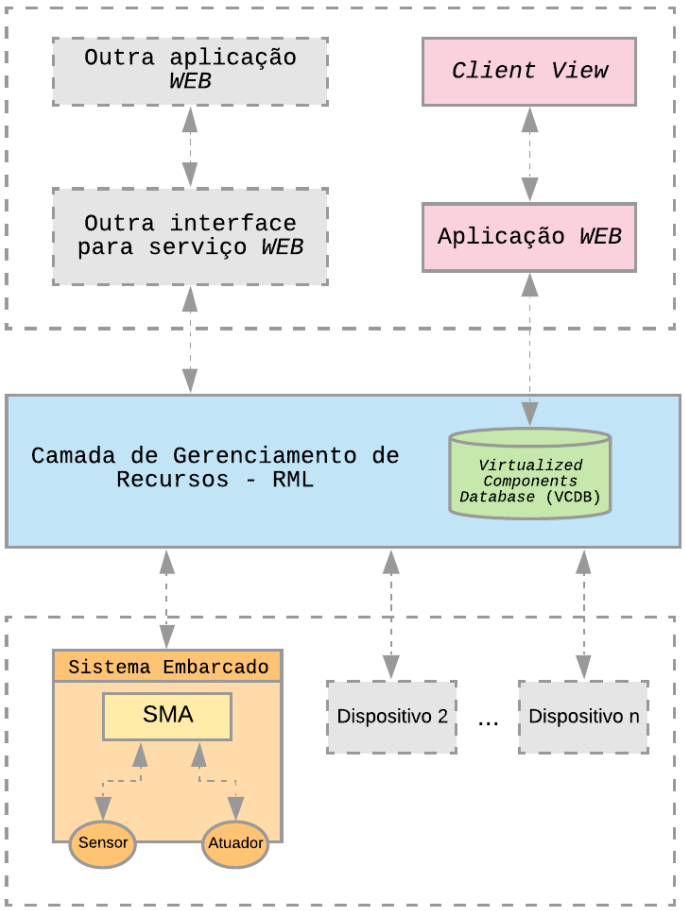


Figura 1. Arquitetura de Gerenciamento de Recursos

3.1. Camada de cliente

Na Camada de cliente estão todas as aplicações capazes de consumir dados dos dispositivos e exibi-los aos usuário além de permitir uma interface para envio de comandos para serem executados nos atuadores dos dispositivos. As possibilidades de implementação nesta camada podem ser desde aplicativos móveis à aplicações *WEB*, utilizando qualquer diferentes tipos de tecnologias — incluindo SMA — necessitando somente fazer com que tal aplicação interaja com a Camada de Gerenciamento de Recursos, que é responsável pelo tratamento dos dados que trafegam entre as camadas dos dispositivos e de cliente.

Nesta arquitetura, uma aplicação *WEB* consume os dados gerados pela camada de dispositivos e também é capaz de enviar comandos aos dispositivos. Na Figura 2 é possível ver a organização da arquitetura. Nota-se que a interação com a RML se dá através do banco de dados (*Virtualized Components Database* - VCDB) utilizado pela RML, refatorada em Golang, para armazenar os dados que fluem pela arquitetura. O

VCDB foi implementado utilizando o MongoDB, um banco de dados orientado a documentos que utiliza da formatação JSON. Dessa forma, a aplicação *WEB*, também desenvolvida utilizando Golang, consome os dados armazenados no VCDB, e os disponibiliza ao usuário através da *view* que foi implementada utilizando HTML e Bootstrap.

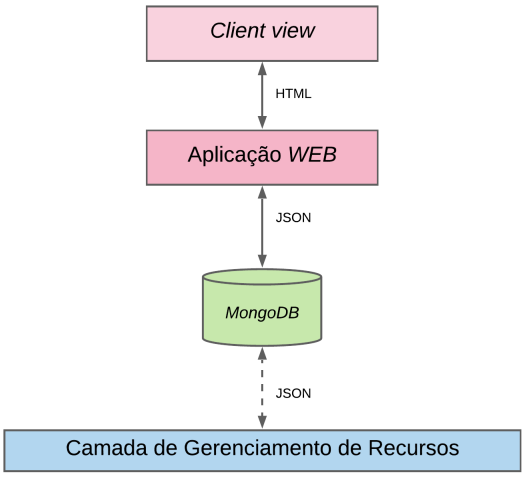


Figura 2. Arquitetura de Gerenciamento de Recursos

3.2. Camada de Gerenciamento de recursos

A Camada de Gerenciamento de Recursos é a camada intermediária responsável por gerenciar os dados gerados e consumidos dentro da arquitetura. A RML é responsável por cadastrar os dispositivos, definir seu número de identificação, controlar seu *status* indicando se este está ativo ou inativo, receber todos os dados que os dispositivos conectados enviam e armazená-los no VCDB, e por fim, designar a determinado dispositivo as ordens que o cliente envia através das aplicações, seja para desligar ou ligar um atuador ou o próprio dispositivo. A RML recebe dos dispositivos que querem se cadastrar um pedido de conexão e as informações para o cadastro e, em ambos os casos, responde com o status de conectado, caso tudo tenha ocorrido dentro do esperado. Após a conexão do dispositivo, este dispositivo envia os dados de seus recursos (e.g., dados dos sensores) para serem armazenados no VCDB e posteriormente consumidos por aplicações clientes. Caso existam ações para serem executadas, a RML repassa tais ações para os dispositivos específicos. Para exemplificar, a Figura 3 demonstra a RML desempenhando suas diversas funções.

Nesta versão refatorada, a RML foi implementada utilizando a linguagem Golang, para estabelecer a conectividade entre as camadas foi utilizado o NATS, um *middleware* de comunicação que funciona a partir de um aplicação servidora em conjunto da implementação de suas funções em código da aplicação específica (cliente NATS). O NATS é um *middleware* orientado a mensagem, que dentre suas funcionalidades, possibilita a criação de canais para a troca de mensagens através do modelo *publish-subscribe*, e também onde uma mensagem enviada possa ser respondida, através do modelo denominado *reply-response*. Desse modo, foram implementados canais de comunicação para que a RML compreenda que tipo de dado está chegando.

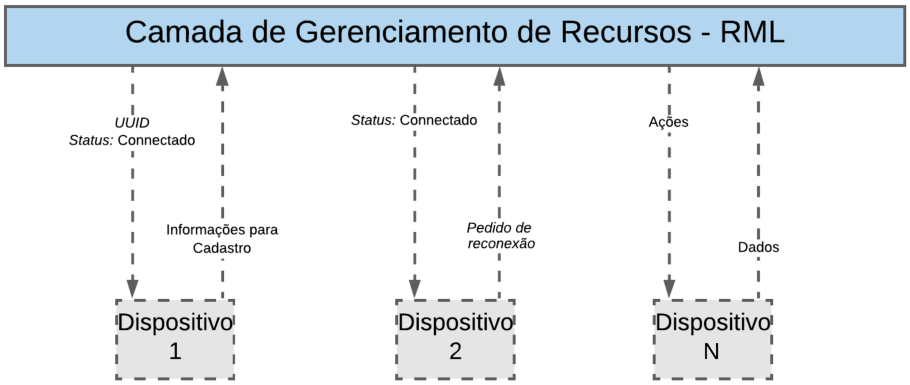


Figura 3. Exemplo de funcionamento da RML

3.3. Camada de dispositivos

A camada de dispositivos representa os possíveis dispositivos e suas tecnologias inseridos em um ambiente real que podem se conectar a RML para deixarem seus recursos virtualizáveis às aplicações clientes. Um dispositivo usado na arquitetura da RMA é composto de sensores e atuadores ligados a um controlador e gerenciado por um sistema embarcado. Neste trabalho, um dispositivo é embarcado com um SMA, que é capaz de interagir com o ambiente podendo obter percepções, processa-las e enviar tais informações e conclusões para RML ou então, executar uma ação na ponta de um sistema (*edge computing*), sem necessidade de encaminhar tais dados para tomada de decisão em outro local.

O *framework* adotado para a programação do SMA foi o JaCaMo. Para a integração com o hardware e com o *middleware* é necessário empregar os agentes Argo [Pantoja et al. 2016], que recolhe as informações do ambiente, e o agente Comunicador [Pantoja et al. 2018], que foi adaptado neste trabalho para ser um instância cliente do NATS para realizar a comunicação com a RML refatorada em Golang. Para isso, o Agente Comunicador possui duas ações internas adaptadas para realizar os modelos de comunicação *publish-subscribe* e *reply-response* utilizando o NATS com o objetivo de se conectar e enviar as informações para a RML: *connectToRml* e a *sendToRML*.

4. Avaliação experimental

Para avaliar a performance de conexão da RMA-GO, os mesmos testes realizados para avaliar a performance da RMA [Pantoja et al. 2019] em Java foram replicados. Para isso, foram criados 50 dispositivos simulados que enviam um pedido de conexão a RML e, uma vez conectados, começam a enviar mensagens com latência de 1 segundo. Então, será medido o tempo de conexão desde o pedido da requisição até a recebimento da confirmação para avaliar se a RML tem uma performance melhor que anterior conforme dispositivos são conectados e começam a enviar informações. Os resultados dos testes podem ser vistos na Figura 4.

Na RMA Java, conforme a quantidade de dispositivos aumenta, o fluxo de dados cresce na mesma proporção e portanto, o tempo de conexão cresce. Porém, na RMA-GO, pode-se concluir que cerca de 50 dispositivos ainda não influenciam negativamente

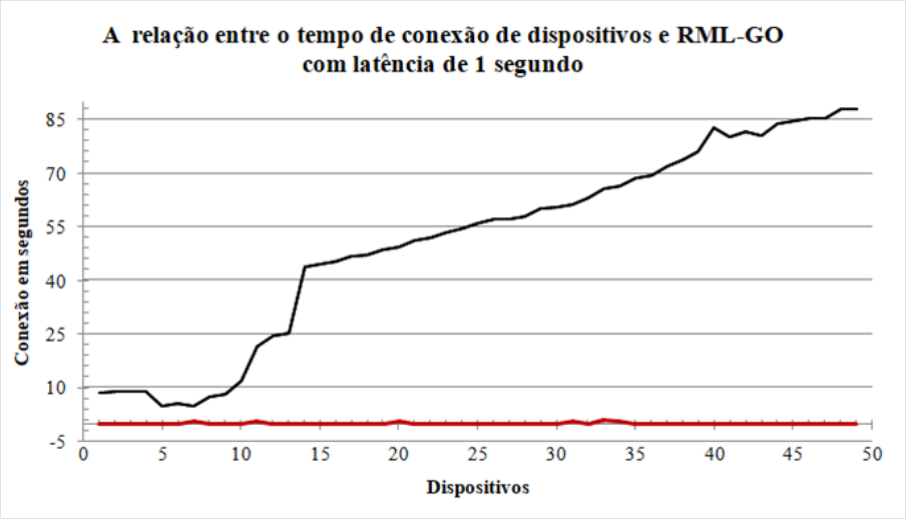


Figura 4. Teste de performance do tempo de conexão da RML-GO.

no tempo de conexão de novos IoT Objects. Este resultado ajuda a reduzir uma das preocupações em implementações de SMA em uma rede IoT. Mesmo assim, ainda é necessário avaliar o tempo de conexões em um cenário onde a quantidade de dispositivos conectados é maior do que 100.

5. Trabalhos Relacionados

Diversos trabalhos focam no desenvolvimento de plataformas de gerenciamento de recursos para IoT integrando SMA ou agentes. Muitas dessas plataformas focam na centralização de agentes [Chaouche et al. 2014][Zschörnig et al. 2019] ou em soluções ad-hoc [Sánchez-Pi et al. 2010][Villarrubia et al. 2014], o que de fato não demonstra o potencial da capacidade distributiva de agentes e também limitam ou dificultam a utilização em domínios diferentes. Além disso, existe uma lacuna nos testes e validação em larga escala de tais soluções. A RMA inicial [Pantoja et al. 2019] demonstra alguns resultados iniciais com uma quantidade de dispositivos que demonstram que sua utilização se limitam a poucos dispositivos por solução (entre 50 e 100), o que pode limitar a sua utilização em grandes escalas e em soluções maiores. Apesar da RMA ser genérica considerando o domínio e ser capaz de ser replicada de forma individual, a intercomunicação entre essas réplicas não seria possível, tendo assim que uma aplicação cliente estar conectada a diversos servidores. Neste trabalho, foi adotada a RMA refatorada para a virtualização de dispositivos embarcados com SMA visando a ampliação da capacidade de processamento de mensagens e conexões de dispositivos como uma indicação de que seja possível unificar diversas soluções em um único servidor ou então permitir o aumento da capacidade de dispositivos empregados em uma única solução.

6. Conclusão

Este trabalho apresentou uma refatoração da RMA para virtualização de recursos de dispositivos embarcados com SMA. Os recursos dos dispositivos são consumidos por aplicações clientes, que acessam seus dados através da RML. Para permitir que mais números de conexões e de mensagens processadas pudessem ser enviadas e tratadas, a RML foi refatorada utilizando a Golang, por ser uma linguagem com menos custo de

processamento que a anterior utilizada, o Java. Dessa forma, também foi modificada o *middleware* utilizado na comunicação de dados entre as camadas para o NATS. Testes iniciais de performance indicaram que a refatoração demonstrou resultados satisfatórios e promissores.

Apesar dos testes iniciais, a RMA-GO ainda precisa de ajustes e de uma bateria de testes mais apropriados que estressem todos os pontos possíveis da arquitetura. Além disso, também é necessário aplicar em um domínio ou estudo de caso para garantir que o sistema como um todo responderá adequadamente. Portanto, como trabalhos futuros, a RMA-GO será aplicada em um cenário mais específico utilizando diversos dispositivos com SMA embarcados onde estes devem se organizar para coletivamente gerenciarem um ambiente baseado no que perceberem, e lidar com as ações que as aplicações clientes precisam que seja executadas em ambiente e que sejam conflitantes com as intenções dos agentes.

Referências

Bandyopadhyay, S., Sengupta, M., Maiti, S., and Dutta, S. (2011). A survey of middleware for internet of things. In *Recent trends in wireless and mobile networks*, pages 288–296. 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.

Byous, J. (1998). Java technology: an early history. *URL: http://java. sun. com/features/1998/05/birthday. html, Artigo pesquisado em 07 de Junho de 2002*.

Chaouche, A.-C., Seghrouchni, A. E. F., Ilié, J.-M., and Saïdouni, D. E. (2014). A higher-order agent model with contextual planning management for ambient systems. In *Transactions on Computational Collective Intelligence XVI*, pages 146–169. Springer.

Donovan, A. A. and Kernighan, B. W. (2015). *The Go programming language*. Addison-Wesley Professional.

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.

Khan, W. Z., Ahmed, E., Hakak, S., Yaqoob, I., and Ahmed, A. (2019). Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235.

Nilsson, E. and Pregén, V. (2020). Performance evaluation of message-oriented middleware.

Pantoja, C. E., Jesus, V. S., Manoel, F. C. P. B., and Viterbo, J. (2018). A heterogeneous architecture for integrating multi-agent systems in ami systems. *The Thirtieth International Conference on Software Engineering and Knowledge Engineering (SEKE 2018)*.

Pantoja, C. E., Soares, H. D., Viterbo, J., Alexandre, T., Seghrouchni, A. E.-F., and Casals, A. (2019). Exposing iot objects in the internet using the resource management architecture. *International Journal of Software Engineering and Knowledge Engineering*, 29(11n12):1703–1725.

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 *International Workshop on Engineering Multi-Agent Systems*, pages 136–155. Springer.

Quevedo, W. (2018). Introduction to nats. In *Practical NATS*, pages 1–18. Springer.

Sánchez-Pi, N., Mangina, E., Carbó, J., and Molina, J. M. (2010). *Multi-agent System (MAS) Applications in Ambient Intelligence (AmI) Environments*, pages 493–500. Springer Berlin Heidelberg, Berlin, Heidelberg.

Togashi, N. and Klyuev, V. (2014). Concurrency in go and java: performance analysis. In *2014 4th IEEE International Conference on Information Science and Technology*, pages 213–216. IEEE.

Villarrubia, G., De Paz, J. F., Bajo, J., and Corchado, J. M. (2014). Ambient agents: Embedded agents for remote control and monitoring using the PANGEA platform. *Sensors*, 14(8):13955–13979.

Wooldridge, M. (2009). *An Introduction to Multi-Agent Systems*. Wiley.

Zhang, D., Ning, H., Xu, K. S., Lin, F., and Yang, L. T. (2012). Internet of things j. ucs special issue. *Journal of Universal Computer Science*, 18(9):1069–1071.

Zschörnig, T., Wehlitz, R., and Franczyk, B. (2019). A fog-enabled smart home analytics platform. In *ICEIS (1)*, pages 616–622.

Descoberta de tamanho de mapas ilimitados através da cooperação entre agentes BDI

Vitor Luis Babireski Furio[1], Maiquel de Brito[1],Tiago L. Schmitz[2], Cleber J. Amaral[1,3], Robson Zagre Junior[1], Maicon R. Zatelli[1], Mauri Ferrandin[1], Timotheus Kampik[4]

[1]Universidade Federal de Santa Catarina (UFSC)

[2]Universidade do Estado de Santa Catarina (UDESC)

[3]Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina (IFSC)

[4]Umeå University, Sweden

Abstract. *Agents can cooperate in exploring and mapping unknown environments. The produced maps may be unbounded. In this case, an agent moving straight in the same direction eventually arrives back at the starting point. The map size, that is the distance back to the starting point, is relevant information in the exploration of unknown environments. If it is not known, the same point may be mapped several times as distinct spots. However, this information is not available in some systems. Even a global position reference may be unavailable. This paper describes an algorithm based on the collaboration among the agents to discover the size of unbounded maps. The results are experimentally evaluated in the Multi-Agent Programming Contest Scenario.*

Resumo. *Agentes podem cooperar na exploração e mapeamento de ambientes desconhecidos. Os mapas produzidos podem ser ilimitados, de forma que, ao andar em uma mesma direção, o agente eventualmente retornará ao ponto de origem. O tamanho do ambiente, que é a distância de uma volta completa até retornar ao ponto de origem, é uma informação relevante na exploração de ambientes desconhecidos. Sem ela, um mesmo ponto pode ser mapeado múltiplas vezes como se todos os mapeamentos fossem pontos distintos. No entanto, alguns sistemas não fornecem essa informação e não fornecem sequer um referencial global de posicionamento. Este artigo descreve um algoritmo baseado na colaboração entre agentes para a descoberta do tamanho de mapas ilimitados. Os resultados são avaliados experimentalmente usando o cenário do Multi-Agent Programming Contest.*

1. Introdução

O estudo de corpos planetários por meio de exploração direta usando espaçonaves robóticas e tripuladas é uma área de pesquisa de fronteira, assim como a exploração de oceanos, tanto na Terra como em outros planetas. Há um interesse renovado na exploração planetária com planos de missões robóticas ou tripuladas à Lua, Marte e além. Já, a exploração de oceanos tanto na Terra como em outros planetas oferece a oportunidade de buscar uma origem da vida, e também de avançar as capacidades humanas para explorar e compreender a vida nos oceanos da Terra [Bhandari 2008, Hand and German 2018].

Agentes, devido às suas características inerentes, podem cooperar na exploração de ambientes desconhecidos, sejam eles reais, como no caso de robôs que exploram um território (e.g., em um novo planeta ou oceano), sejam virtuais, como por exemplo em simulações ou jogos eletrônicos. Nessa exploração, os agentes possivelmente produzem um *mapa* do ambiente, em que registram as informações sobre os elementos descobertos. Estes mapas são *ilimitados* em ambientes em que um agente que segue em uma mesma direção, sem encontrar obstáculos ao longo do percurso, dará uma volta completa e eventualmente retornará ao ponto de origem [Nieuwenhuisen et al. 2011]. Este é o caso de ambientes que têm a forma de esferas ou toróides. A distância de uma volta completa em uma certa direção é o *tamanho* do mapa naquela direção. Por exemplo, o Planeta Terra tem um tamanho aproximado de 40.075 Km na direção da Linha de Equador e de 40.008 Km no Meridiano de Greenwich [Aeronautics and Administration 2020, PLC 2018].

O tamanho dos mapas ilimitados é uma informação relevante para agentes que exploram ambientes desconhecidos. Sem ela, eles poderiam andar continuamente no ambiente sem saber que estão passando por zonas já exploradas. Como consequência, um mesmo ponto poderia ser mapeado múltiplas vezes em posições distintas. Em certos cenários, no entanto, é possível que os agentes não possuam essa informação. Além disso, é possível que não haja sequer um referencial de posicionamento global que os permita calcular esse tamanho. Nesse caso, os agentes precisam, de alguma forma, descobri-lo.

Este artigo descreve um algoritmo para a descoberta do tamanho de mapas ilimitados. O algoritmo é distribuído pois é baseado na colaboração entre agentes BDI [Bratman 1987]. Estes agentes são desenvolvidos com base em três abstrações essenciais: crenças (ou *beliefs*), que são as informações que o agente possui sobre o sistema; desejos, que são os estados do sistema que o agente gostaria de atingir; e intençoes, que são os estados do sistema que o agente está efetivamente atuando para atingir. Diferentes agentes utilizam seu conhecimento parcial a respeito do mapa, para, colaborando entre si, descobrirem uma informação global sobre ele, que, nesse caso, são as suas dimensões. Assume-se que os agentes não possuem qualquer informação inicial sobre o tamanho do mapa e nem sobre um referencial comum de posicionamento. O algoritmo é descrito através das transições no estado interno de cada um dos agentes envolvidos na interação — em particular, de suas crenças e desejos — para descoberta do tamanho do mapa. Os resultados são avaliados de forma experimental através da aplicação do algoritmo proposto ao cenário do 15^{th} *Multi-Agent Programming Contest*.[1] Nele, os agentes atuam em um ambiente que apresenta as restrições descritas anteriormente.

O restante deste artigo está estruturado da seguinte forma: a Seção 2 descreve o ambiente considerado pelo algoritmo proposto; o algoritmo é descrito na Seção 3 e avaliado na Seção 4; por fim, a Seção 5 descreve alguns trabalhos relacionados, conclusões e trabalhos futuros.

2. Descrição do ambiente

Esta seção descreve o ambiente considerado pelo algoritmo descrito na Seção 3. Para descrevê-lo, utiliza-se a noção de *mapa*, que é uma representação deste ambiente. Nesta seção, para fins de descrição do ambiente, considera-se que o mapa é uma representação completa do ambiente, como se este fosse inteiramente conhecido por quem o construiu.

[1] https://multiagentcontest.org/2020/

Já o algoritmo proposto na Seção 3 considera que o mapa é gradualmente descoberto pelos agentes.

O ambiente pode ser representado por um mapa bidimensional cujas coordenadas compõem um plano cartesiano. Cada ponto do mapa é identificado por um par de coordenadas (x, y) em que $x \in \mathbb{R}$ e $y \in \mathbb{R}$. Qualquer ponto do mapa pode ser tomado como o ponto de *origem*, com coordenadas $(0, 0)$. As demais coordenadas são incrementadas no eixo x à medida em que desloca-se à direita e no eixo y à medida que desloca-se para cima. Coordenadas à esquerda do ponto zero no eixo x e abaixo do ponto zero no eixo y assumem valores negativos.

Apesar de *ilimitado*, o mapa, assim como o conjunto de coordenadas que a ele pertencem, é *finito*. A partir de qualquer coordenada, há limites acima e abaixo do eixo y e à esquerda e à direita do eixo x. Neste artigo, W representa o conjunto de todas as coordenadas que pertencem ao mapa. Assim, para qualquer valor de $q \in \mathbb{R}$, (i) $max_y(x)$ é o maior valor de y acima de um ponto (x, q); (ii) $min_y(x)$ é o menor valor de y abaixo de um ponto (x, q); (iii) $max_x(y)$ é o maior valor de x à direita de um ponto (q, y); e (iv) $min_x(y)$ o menor valor de x à esquerda de um ponto (q, y), conforme as expressões 1 a 4:

$$max_y(x) = \{\psi | (x, \psi) \in W \wedge \forall_{(x,y)\in W}\ y \neq \psi \leftrightarrow y < \psi\} \tag{1}$$
$$min_y(x) = \{\psi | (x, \psi) \in W \wedge \forall_{(x,y)\in W}\ y \neq \psi \leftrightarrow y > \psi\} \tag{2}$$
$$max_x(y) = \{\psi | (\psi, y) \in W \wedge \forall_{(x,y)\in W}\ x \neq \psi \leftrightarrow x < \psi\} \tag{3}$$
$$min_x(y) = \{\psi | (\psi, y) \in W \wedge \forall_{(x,y)\in W}\ x \neq \psi \leftrightarrow x > \psi\} \tag{4}$$

Além disso, o mapa é retangular, obedecendo às seguintes propriedades:

$$\forall_x \forall_{x'}\ max_y(x) = max_y(x') \tag{5}$$
$$\forall_x \forall_{x'}\ min_y(x) = min_y(x') \tag{6}$$
$$\forall_y \forall_{y'}\ max_x(y) = max_x(y') \tag{7}$$
$$\forall_y \forall_{y'}\ min_x(y) = min_x(y') \tag{8}$$

No mapa ilustrado na Figura 1, para quaisquer valores de x e y, tem-se $max_x(y) = 6$, $min_x(y) = -3$, $max_y(x) = 4$ e $min_y(x) = -1$.

Em um mapa ilimitado, caso o agente movimente-se permanentemente em linha reta, chegará, eventualmente, ao ponto de partida. Seja a relação denotada por $\equiv$ tal que $(x, y) \equiv (v, w)$ significa que os pontos (x, y) e (v, w) referem-se ao mesmo local no mapa. Para qualquer coordenada (x, y), tem-se:

$$(x, y) \equiv \begin{cases} (x, y - max_y(x) + min_y(x) - 1) & \text{se } y > max_y(x) \\ (x, y - min_y(x) + max_y(x) + 1) & \text{se } y < min_y(x) \\ (x - max_x(y) + min_x(y) - 1, y) & \text{se } x > max_x(y) \\ (x - min_x(y) + max_x(y) + 1, y) & \text{se } x < min_x(y) \end{cases} \tag{9}$$

Pelas características até aqui descritas, verifica-se o tamanho do mapa pode ser dado pelo tamanho das suas dimensões horizontal e vertical, notados respectivamente como $size_x$ e $size_y$, que satisfazem as seguintes igualdades considerando alguma coordenada (x, y):

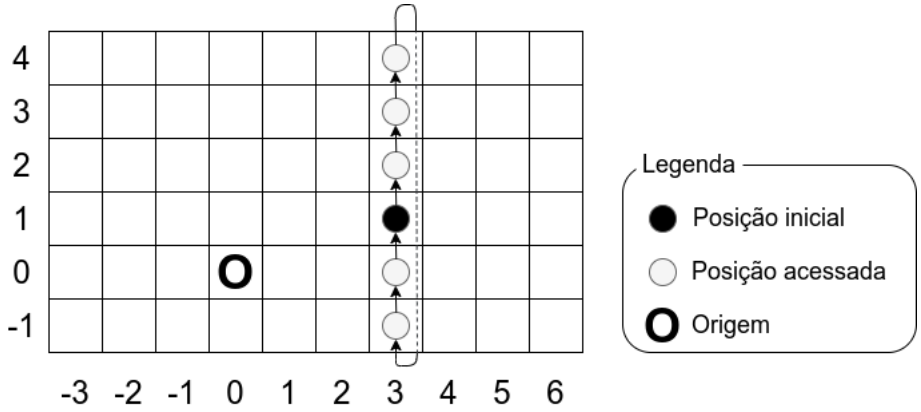


Figura 1. Exemplo de mapa ilimitado: um agente que se desloca em linha reta a partir do ponto (3,1) retornará evenualmente àquele ponto.

$$size_x = max_y(x) - min_y(x) + 1 \tag{10}$$
$$size_y = max_x(y) - min_x(y) + 1 \tag{11}$$

A partir dessas informações, é possível calcular novas informações sobre o mapa, tais como menores distâncias entre dois pontos, ou perímetro e área do mapa. Assume-se que os agentes conheçam os pressupostos definidos pelas expressões 5 a 11 mas não conheçem os valores de max_x, min_x, max_y e min_y. Assim, não podem calcular e, consequentemente, utilizar os valores definidos por aquelas expressões. Além disso, os agentes não têm acesso a um sistema global de coordenadas. Os próprios agentes precisam manter um sistema de coordenadas próprio, assumindo como origem o ponto em que se encontravam ao serem inicializados e assumindo, para cada ponto (x, y) do mapa, que os valores de x e y são as distâncias que separam este ponto do ponto de origem. Assim, os agentes não compartilham um referencial de localização comum. Um mesmo objeto pode estar localizado em coordenadas diferentes sob a perspectiva de diferentes agentes. Por fim, os agentes conseguem perceber outros elementos (incluindo outros agentes) localizados a uma distância máxima d_m tal que: $d_m < size_x \vee d_m < size_y$. Ou seja, os agentes não têm visão de todo o mapa.

3. Algoritmo para cálculo do tamanho de mapas contínuos

O algoritmo descrito nesta seção leva os agentes a descobrirem o tamanho do mapa em que se encontram a partir da interação entre eles. Por isso, não é possível expressá-lo com clareza na forma de um algoritmo tradicional composto por uma sequência única de passos. Em vez disso, o algoritmo será expresso através das transformações que ocorrem no estado interno de cada um dos agentes ao longo de sua execução. Esse estado será representado aqui por uma tupla $C = \langle B, D \rangle$, em que B é o conjunto das crenças do agente a respeito do estado do mundo e D é o conjunto dos seus desejos. As crenças são representadas através de predicados que indicam propriedades e relações acerca dos elementos do mundo em que os agentes atuam. As transformações no estado de um

agente serão representadas através de regras de transição inspiradas no formalismo proposto por [Plotkin 2004]. Nele, expressa-se que uma série de pré condições $p_1, p_2, \cdots, p_n$ leva um sistema de uma configuração C para C' da seguinte forma:

$$\frac{p_1 \quad p_2 \quad \cdots \quad p_n}{C \rightarrow C'} \tag{12}$$

O algoritmo trata da descoberta do tamanho do mapa sob a perspectiva individual de um agente. A informação descoberta pode ser oportunamente compartilhada entre os demais agentes que cooperam em um sistema. O algoritmo se divide em duas partes: a adoção de um referencial comum por parte dos agentes, descrita na Seção 3.1, e o cálculo do tamanho do mapa, descrito na seção 3.2.

3.1. Adoção de referencial comum

Assume-se que há um conjunto A de agentes atuando no ambiente tal que $|A| > 1$. Ou seja, há mais de um agente. Esta é condição essencial para o funcionamento do algoritmo pois ele se baseia na colaboração entre os agentes. Assim, para $|A| = n$, nota-se $A = \{a_1, \cdots, a_n\}$. O estado do agente a_i é dado por $C^i = \langle B^i, D^i \rangle$.

Cada agente assume, inicialmente, um ponto de origem particular, com coordenadas $(0, 0)$, que é aquele em que se localizava quando começou a atuar no sistema. Como o algoritmo proposto depende da colaboração entre os agentes, eles precisam compartilhar um referencial comum, que é definido por eles mesmos ao longo de sua execução. Para isso, podem (i) adotar o referencial de origem de outros agentes com os quais se encontram ou (ii) compartilhar seu referencial de origem com outros agentes.[2] Cada agente $a \in A$ possui a crença $origin(O)$, em que $O \in A$ identifica o agente cujo referencial é adotado pelo agente que possui a crença. Todo agente possui uma crença $my\_id(Id)$, em que Id é o seu identificador. Conforme a regra de transição 13, se um agente não possui qualquer crença referente ao referencial adotado, então passa a crer que o referencial é o seu prório ponto de origem (que é, de fato, a condição inicial).

$$\frac{my\_id(me) \in B \quad \neg \exists_x \, origin(x) \in B}{B \rightarrow B \cup origin(me)} \tag{13}$$

Ao se deslocarem, os agentes podem se encontrar uns com os outros. Um *encontro* acontece quando dois agentes estão separados a uma distância menor que o alcance máximo de suas percepções.[3] Ao se encontrarem, os agentes comunicam suas posições diretamente por troca de mensagens ou utilizando algum recurso disponível no sistema, como por exemplo um *blackboard*.

Pela regra de transição 14, um agente que acredita (i) estar em uma posição (x, y) e (ii) haver um agente $agent\_id$ em uma posição próxima, passa a ter o desejo[4] de lhe enviar

[2]Possíveis estrategias para encontrar um referencial comum estão fora do escopo deste artigo e são discutidas, por exemplo, em [Cardoso et al. 2019, Uhlir et al. 2019, Jensen and Villadsen 2019]

[3]Caso um agente encontre mais de um agente ao mesmo tempo, todos os encontros são tratados sumultaneamente porém separadamente conforme as regras descritas nesta seção.

[4]Conforme o modelo BDI, um desejo depende de um processo de deliberação para gerar uma intenção que será executada segundo um plano de ação definido pelo processo de raciocínio prático. Assume-se que os desejos do agente eventualmente produzirão intenções que serão executadas com sucesso através de um plano de ação.

a informação $meet(me, x, y, p_x, p_y, pid, now)$ em que (1) me é o identificador do agente que envia a mensagem, (2) (x, y) são as coordenadas do próprio agente, (3) p_x e p_y são as distâncias horizontal e vertical que separam o agente emissor do receptor da mensagem, (4) pid é um identificar único do encontro entre os agentes, necessário para que os agentes gerenciem múltiplos encontros ao longo de sua execução e (5) now identifica o momento em que o encontro aconteceu baseado em um relógio comum que pode ser percebido por todos os agentes.

$$\frac{\{my\_id(me), mypos(x, y), time(now), teammate(agent\_id, x + p_x, x + p_y)\} \subseteq B}{D \rightarrow D \cup send(tell, agent\_id, meet(me, x, y, p_x, p_y, pid, now))} \tag{14}$$

Conforme a abordagem mentalista para a comunicação entre agentes, uma informação i enviada de um agente a para um agente b, com uma performativa apropriada, faz com que b passe possuir a crença i [Adam and Gaudou 2010, Labrou and Finin 1994]. Este é o caso da performativa *tell*, definida pela linguagem para comunicação de agentes KQML (*Knowledge Query and Manipulation Language*) e utilizada neste artigo a partir daquela definição. Assim, se o desejo $send(tell, agent\_id, meet(sender\_id, x, y, p_x, p_y, pid, time))$ for satisfeito, então, eventualmente, $meet(sender\_id, x, y, p_x, p_y, pid, time) \in B^{agent\_id}$.

A regra de transição 15 define a mudança de referencial de origem de um agente. Essa mudança acontece quando um agente possui a crença $meet(sender\_origin, x, y, p_x, p_y, pid, meet\_time)$, que indica que o agente em questão encontrou-se com outro agente. Essa mudança está sujeita a duas condições. Primeiro, o indicador de referencial do outro agente precisa ser menor do que o agente em questão. Define-se essa condição para que, por ocasião do encontro de dois agentes, haja um critério claro sobre qual deles deve adotar referencial do outro. Essa é uma estratégia possível entre outras. A segunda condição é que o agente em questão deve ter uma visão de mundo simultânea ou posterior àquela que os agentes tinham no momento do encontro. Isso garante que as atualizações nas crenças serão feitas com base em informações atualizadas. Se essas condições forem satisfeitas, o agente (i) modifica sua crença a respeito de qual é o seu indicador de referencial e (ii) ajusta as coordenadas de sua posição atual segundo o novo referencial.

$$\frac{\begin{array}{c}\{meet(sender\_origin, x, y, p_x, p_y, pid, meet\_time), mypos(m_x, m_y), \\ origin(o), now(time)\} \subseteq B \\ time \geq meet\_time \wedge sender\_origin < o\end{array}}{\begin{array}{c}B \rightarrow B \setminus \{mypos(m_x, m_y), origin(o)\} \cup \\ \{mypos(m_x + p_x, m_y + p_y), origin(sender\_origin)\}\end{array}} \tag{15}$$

3.2. Cálculo do tamanho do mapa

Conforme a regra 14, os agentes trocam mensagens sempre que se encontram uns com os outros. Se dois agentes que compartilham o mesmo indicador de origem se encontram,

pode-se concluir que eles já se encontraram anteriormente (do contrário não compartilhariam a mesma origem). Se um agente conhece as distâncias que lhe separam de outro agente que compartilha a mesma origem, então pode calcular quais são as coordenadas deste outro agente. Se as coordenadas calculadas forem diferentes das coordenadas informadas pelo outro agente com relação aos eixos x, y ou ambos, então essa diferença é o tamanho do mapa em um desses eixos. Por exemplo, sejam dois agentes A e B que compartilham um mesmo referencial de origem. O agente A está na posição $(3, 6)$ e avista o agente B a 3 unidades de distância à direita e 2 a unidades de distância acima. O agente A pode concluir que é a posição de B é $(3 + 3, 6 + 2) = (6, 8)$. Suponha-se que B acredite estar na posição $(26, 8)$ (ou seja, $mypos(26, 8) \in B^B$). Quando A comparar a posição calculada de B com a posição informada por B, concluirá que o raio do mapa na direção horizontal é 20 ($(26 - 6, 8-, 8) = (20, 0)$). Isso acontece porque o agente A avista B em sua posição real $(6, 8)$ enquanto o agente B acredita estar em $(26, 8)$ pois deu uma volta completa e, por não conhecer o tamanho do ambiente, continuou a contar as distâncias mesmo após ter se deslocado por 20 unidades de medida.

As regras 16 e 17 definem como um agente conclui o tamanho dos eixos x e y do ambiente.

$$\frac{\begin{array}{c}\{meet(sender\_origin, x, y, p_x, p_y, pid, meet\_time), mypos(m_x, m_y), \\ origin(o), now(time)\} \subseteq B \\ time \geq meet\_time \wedge sender\_origin = o \wedge x \neq m_x + p_x\end{array}}{B \rightarrow B \cup \{size_x(|x - (m_x + p_x)|)\}} \tag{16}$$

$$\frac{\begin{array}{c}\{meet(sender\_origin, x, y, p_x, p_y, pid, meet\_time), mypos(m_x, m_y), \\ origin(o), now(time)\} \subseteq B \\ time \geq meet\_time \wedge sender\_origin = o \wedge y \neq m_y + p_y\end{array}}{B \rightarrow B \cup \{size_y(|y - (m_y + p_y)|)\}} \tag{17}$$

A partir das informações sobre o tamanho do mapa, o agente pode calcular os valores definidos pelas expressões 5 a 11, bem como recacular o posicionamento de objetos que tenham sido mapeados em coordenadas que estão fora dos limites calculados. Estes aspectos, entretanto, estão fora do escopo deste artigo.

Idealmente, quaisquer dois agentes envolvidos nas interações descritas anteriormente devem chegar aos mesmos valores para os tamanhos dos eixos x e y. Entretanto, pode haver discrepâncias nesses valores se, por exemplo, o sistema de localização de um dos agentes falhar e ele não mantiver um registro consistente de suas próprias coordenadas. O tratamento deste problema pode ser feito de diferentes maneiras, a depender da aplicação específica. Por exemplo, se o número de agentes atuando no sistema for conhecido, pode-se utilizar algum algoritmo de consenso a respeito das medidas calculadas.

4. Experimentos e avaliação

O cenário do 15^{th} *Multi-Agent Programming Contest* (MAPC) inclui as características e limitações descritas na Seção 2. O algoritmo descrito na Seção 3 foi aplicado a um time

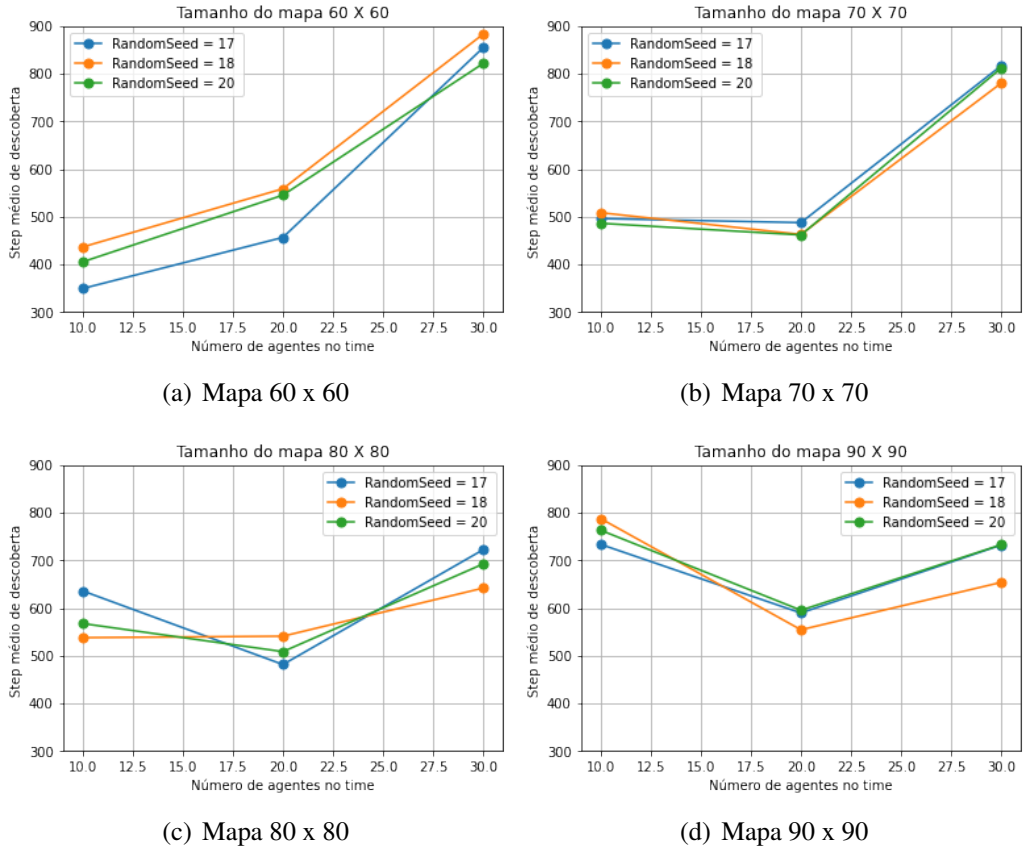
de agentes[5] que se dedicam a várias e diferentes tarefas. Em meio à realização destas tarefas, são capazes, através do algoritmo proposto, de descobrir o tamanho do ambiente em que estão atuando. Todos os agentes têm a capacidade de descobrir o tamanho do mapa. Quando este tamanho é descoberto por um agente, ele é compartilhado com todos os demais. Os experimentos realizados confirmam que o algoritmo proposto permite que os agentes descubram as dimensões do mapa. Além disso, os experimentos avaliam a eficiência do algoritmo em função da quantidade de agentes colaborando na descoberta do ambiente e em função do tamanho do próprio ambiente. Para isso, mede-se quantos *steps*, que são os passos em uma simulação MAPC, são necessários para que o tamanho do ambiente seja descoberto.

Nos experimentos, foram usados mapas gerados a partir de diferentes *random seed*. Cada *random seed* produz um mapa semelhante, porém com obstáculos diferentes, gerados aleatoriamente a cada nova execução. Além disso, um mapa gerado a partir de um mesmo *random seed* pode ter diferentes tamanhos. Ao explorar o ambiente, os agentes podem deparar-se com obstáculos que surgem aleatoriamente e também com agentes de times adversários. Assim, em diferentes execuções, os agentes podem precisar percorrer diferentes trajetórias para chegar a um mesmo ponto. Consequentemente, diferentes execuções com as mesmas configurações podem requerer uma quantidade diferente de *steps* para que tamanho do ambiente seja descoberto. Assim, para cada configuração, houve uma série de execuções, das quais foram obtidas as médias da quantidade de steps requerida para a descoberta do tamanho do ambiente. Nos experimentos, para cada *random seed*, foram feitas 10 execuções em mapas de tamanho 60x60, 70x70, 80x80 e 90x90, com times de 10, 20 e 30 agentes. Estes parâmetros foram definidos em função dos recursos computacionais e do tempo disponíveis para os experimentos.

Os resultados dos experimentos são sumarizados na Figura 2 e podem ser avaliados sob duas perspectivas: (i) desempenho de times de mesmo tamanho em mapas de tamanhos diferentes e (ii) desempenho de times diferentes em mapas de mesmo tamanho. Sob a primeira perspectiva, observa-se que o desempenho de times com a mesma quantidade de agentes têm variação semelhante em mapas com o mesmo tamanho, mesmo que tenham *random seeds* diferentes. Times com menos de 20 agentes pioram seu desempenho à medida que o tamanho do mapa aumenta. Isso acontece porque há menos agentes para mapear áreas maiores. Já o desempenho de times com mais de 20 agentes tende a melhorar quanto maior for o tamanho do mapa. Justifica-se isso pois, para mapas pequenos, há maior probabilidade de vários agentes tentarem explorar o mesmo ponto. Isso não é possível e os agentes precisam modificar sua trajetória ou esperar até que aquele ponto esteja livre para ser explorado. Estes aspectos são também observados ao analisar desempenhos de times diferentes em mapas de mesmo tamanho. No caso de mapas de tamanho 60X60, tem-se melhor desempenho quanto menor for o time de agentes pois, sendo este um mapa pequeno, há menos colisões. No caso dos mapas maiores (70x70, 80x80 e 90x90), tem-se 20 como o número ideal de agentes, em que se atinge um equilíbrio entre quantidade de agentes que exploram o ambiente e espaço disponível para o deslocamento destes agentes. Com quantidades menores que 20, há muito espaço para poucos agentes. Isso pode ser verificado ao observar-se que, quanto maior o mapa, pior o desempenho dos times pequenos. Por outro lado, nestes mesmos mapas, também há uma degradação

[5]Disponível em https://github.com/jacamo-lang/mapc2020/tree/map_size_tests

no desempenho à medida que o tamanho dos times aumenta devido às colisões entre os agentes. Essa degradação é menor à medida que o tamanho dos mapas aumenta pois esse aumento reduz a chance de colisões.



(a) Mapa 60 x 60

(b) Mapa 70 x 70

(c) Mapa 80 x 80

(d) Mapa 90 x 90

Figura 2. Resultados dos experimentos

5. Trabalhos relacionados e considerações finais

Há diversos trabalhos que tratam da exploração de ambientes desconhecidos por agentes. Alguns deles consideram a exploração de ambientes cujos mapas *podem* ser ilimitados, sem considerar essa característica como essencial [Hert et al. 1996, Cao et al. 1988, Moorehead 2001]. Este artigo, por sua vez, considera mapas que são necessariamente ilimitados. O escopo do trabalho de [Nieuwenhuisen et al. 2011] tem essa mesma delimitação. Independente disso, todos os trabalhos citados tratam do mapeamento sob a perspectiva de um único agente [Hert et al. 1996, Cao et al. 1988, Moorehead 2001]. Este artigo, por sua vez, trata do mapeamento feito necessariamente por múltiplos agentes que colaboram entre si. Além disso, os trabalhos citados tratam da descoberta de elementos que pertencem ao mapa, enquanto este artigo trata da descoberta do tamanho do mapa.

Os experimentos realizados mostram que o algoritmo descrito neste artigo faz com que os agentes descubram o tamanho do mapa em que estão trabalhando. Como o algoritmo é baseado na colaboração entre os agentes, seu desempenho é sensível (i) à quantidade de agentes envolvidos nessa colaboração e (ii) à capacidade de os agentes explorarem o mundo de maneira eficiente. Isso fica claro ao observar-se que, (i) o desempenho do algoritmo piora quando há poucos agentes para explorar grandes áreas e

(ii) quando a capacidade de movimentação dos agentes é comprometida pela quantidade de agentes e pelo tamanho reduzido do mapa. Essas limitações podem ser tratadas com mudanças na estratégia adotada pelos agentes para explorar o mapa e também por algoritmos como abordagens diferentes, como por exemplo como detecção de padrões de locais já visitados. Planeja-se tratar destes aspectos em trabalhos futuros.

Referências

Adam, C. and Gaudou, B. (2010). Une sémantique unifiée des actes de langage. aspects intentionnels et institutionnels des actes de langage. *Revue d'Intelligence Artificielle*, 24(3):291–323.

Aeronautics, N. and Administration, S. (2020). Earth fact sheet. https://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html. Acesso: 30-04-2020.

Bhandari, N. (2008). Planetary exploration: scientific importance and future prospects. *Current Science*, 94(2):185–200.

Bratman, M. (1987). *Intention, plans, and practical reason*. Harvard University Press, Cambridge, MA.

Cao, Z. L., Huang, Y., and Hall, E. L. (1988). Region filling operations with random obstacle avoidance for mobile robots. *J. Field Robotics*, 5(2):87–102.

Cardoso, R. C., Ferrando, A., and Papacchini, F. (2019). LFC: combining autonomous agents and automated planning in the multi-agent programming contest. In *The Multi-Agent Programming Contest*, pages 31–58.

Hand, K. P. and German, C. R. (2018). Exploring ocean worlds on earth and beyond. *Nature Geoscience*, 11(1):2–4.

Hert, S., Tiwari, S., and Lumelsky, V. J. (1996). A terrain-covering algorithm for an AUV. *Auton. Robots*, 3(2-3):91–119.

Jensen, A. B. and Villadsen, J. (2019). GOAL-DTU: development of distributed intelligence for the multi-agent programming contest. In *The Multi-Agent Programming Contest*, pages 79–105.

Labrou, Y. and Finin, T. W. (1994). A semantics approach for KQML - A general purpose communication language for software agents. In *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94), Gaithersburg, Maryland, USA, November 29 - December 2, 1994*, pages 447–455. ACM.

Moorehead, S. J. (2001). *Autonomous Surface Exploration for Mobile Robots*. PhD thesis, USA. AAI3043383.

Nieuwenhuisen, M., Schulz, D., and Behnke, S. (2011). Exploration strategies for building compact maps in unbounded environments. In Jeschke, S., Liu, H., and Schilberg, D., editors, *Intelligent Robotics and Applications - 4th International Conference, ICIRA 2011, Aachen, Germany, December 6-8, 2011, Proceedings, Part I*, volume 7101 of *Lecture Notes in Computer Science*, pages 33–43. Springer.

PLC, S. F. (2018). How big is earth. https://www.space.com/17638-how-big-is-earth.html. Acesso: 30-04-2020.

Plotkin, G. D. (2004). A structural approach to operational semantics. *J. Log. Algebraic Methods Program.*, 60-61:17–139.

Uhlir, V., Zboril, F., and Vidensky, F. (2019). Multi-agent programming contest 2019 FIT BUT team solution. In *The Multi-Agent Programming Contest*, pages 59–78.