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

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

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

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

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

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

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

1. Introdução

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

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

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

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

2. Fundamentação teórica

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

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

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

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

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

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

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

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

3. Implementação de um sistema de robôs ROS coordenados

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

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

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

3.1. Descrição da simulação

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

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

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

3.2. Implementação de agentes e integração com ROS

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

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

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

Figura 1. Trechos do código do agente

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

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

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

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

Figura 2. Especificação feita no arquivo yaml

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

3.3. Organização

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

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

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

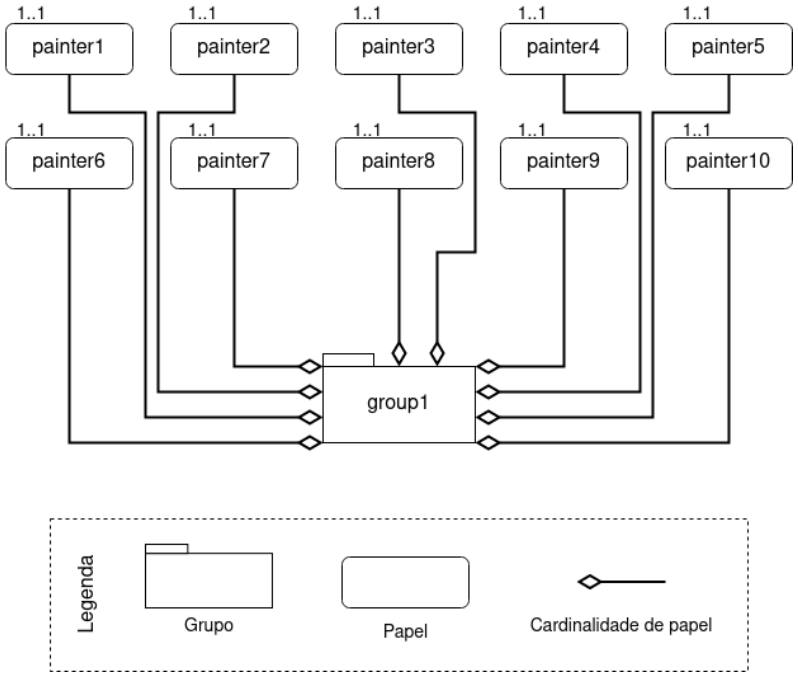


Figura 3. Especificação estrutural

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

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

Tabela 1. Especificação normativa

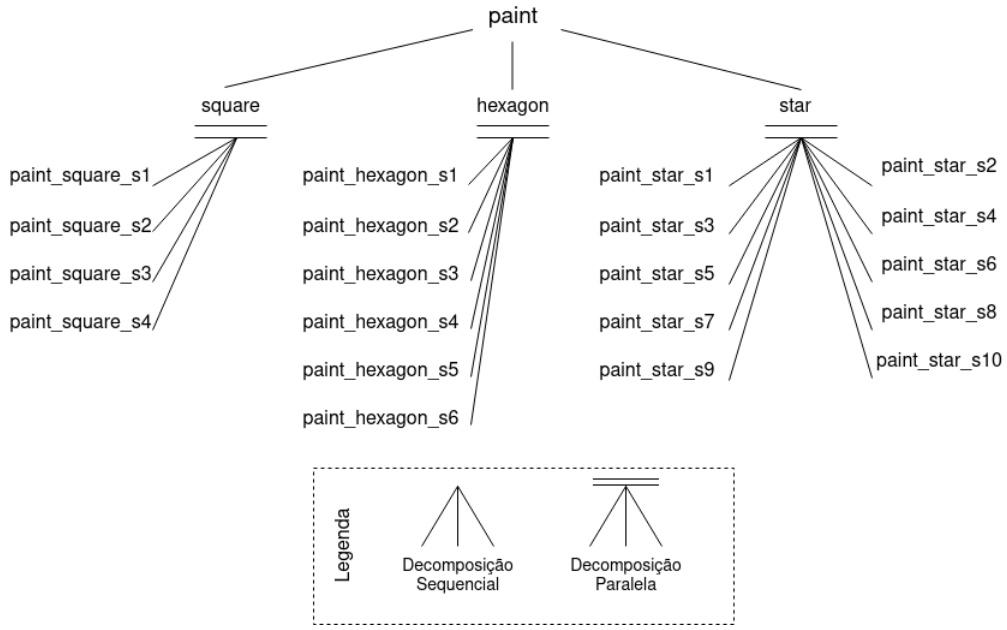


Figura 4. Especificação funcional

3.4. Experimentos

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

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

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

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

Tabela 2. Resultados dos experimentos

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

4. Resultados

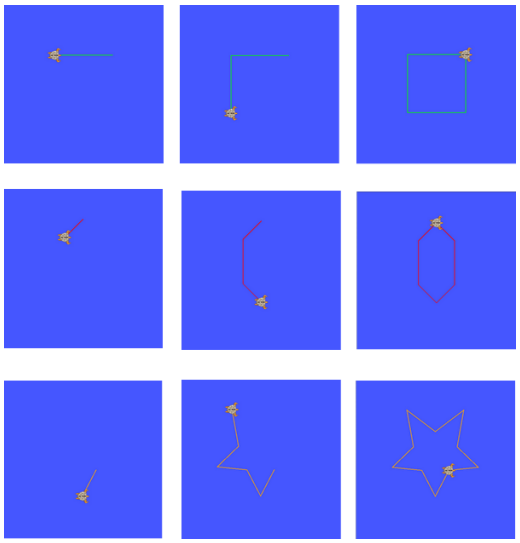
Os robôs cooperaram de forma bem sucedida ao atuar no sistema descrito na Seção 3, desenhando todas as formas geométricas em diferentes combinações de agentes e papéis. Em todas essas combinações, os robôs atuaram de forma coordenada para desenhar as formas geométricas especificadas pela organização.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

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

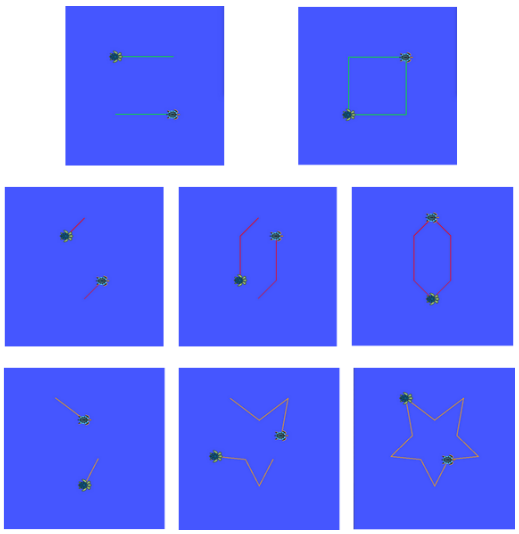
É possível afirmar, também, que a eficiência da operação é afetada pelo número de agentes disponíveis, bem como pela distribuição de papéis e pela definição desses papéis na organização. De modo geral, mais robôs produzem um resultado mais eficiente porque mais tarefas podem ser executadas simultaneamente.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

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

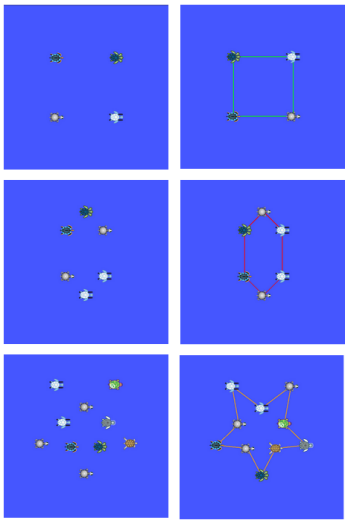
Figura 5. Distribuições de papéis



(a) Um agente construindo as formas geométricas



(b) Dois agentes construindo as formas geométricas



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

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

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

5. Trabalhos relacionados e conclusões

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

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

Referências

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

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

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

Hübner, J. F., Sichman, J. S., and Boissier, O. (2007). Developing organised multiagent systems using the MOISE+ model: Programming issues at the system and agent levels. *IJAOSE*, 1(3/4):370–395.

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

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

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

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

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

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