# Uma Proposta de Emulador de Portas Seriais para Sistemas Multiagentes Embarcados

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

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

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

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

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

## 1. Introdução

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

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

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

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

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

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

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

## 2. Fundamentação Teórica

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

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

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

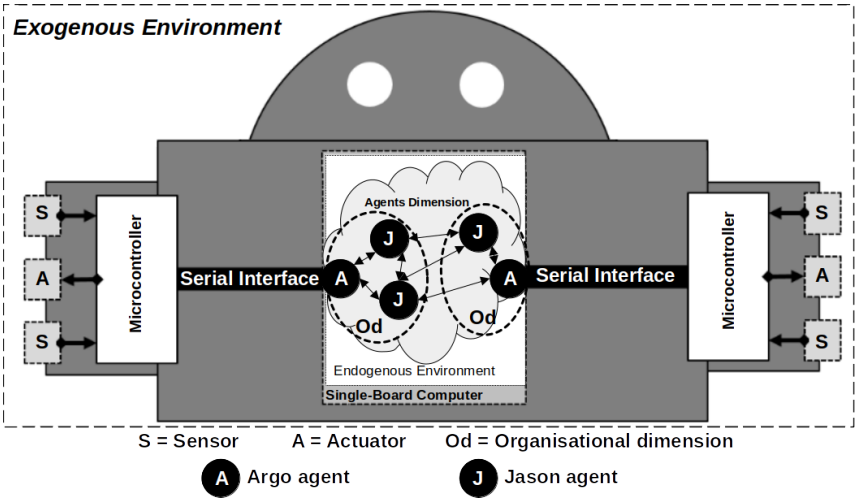



**Figura 1. Arquitetura de SMAs embarcados**

### 2.1. Arquitetura de Dispositivos Seriais

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

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

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

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

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

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

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

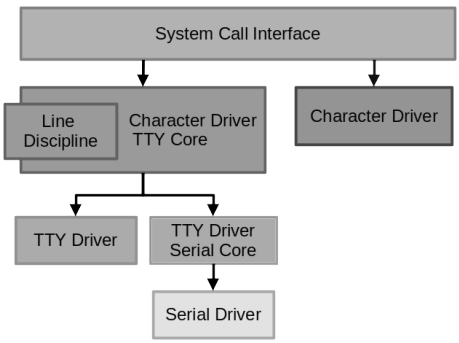



**Figura 2. Arquitetura de drivers seriais no GNU/Linux [Free Electrons 2017].**

## 3. Trabalhos relacionados

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

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

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

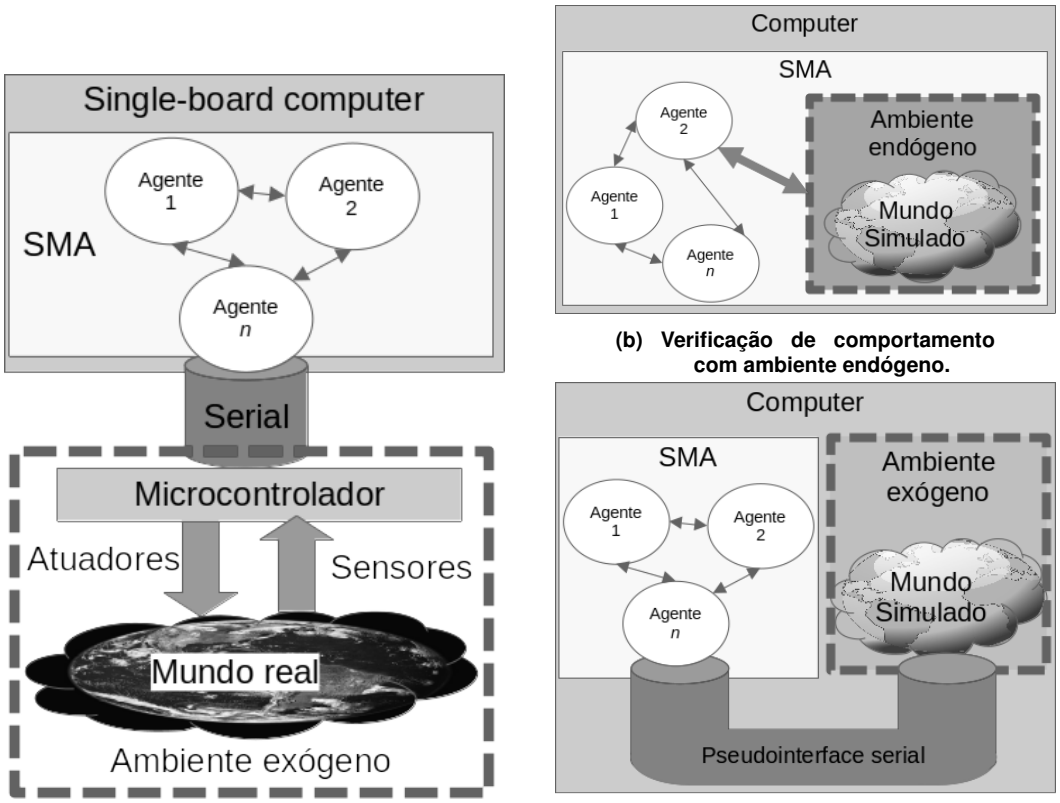
Este trabalho, diferente de [Adi Putra et al. 2018], apresenta uma abordagem que dispensa a necessidade de hardwares físicos para verificar o comportamento do agente, suas ações e percepções do ambiente. Além disso, permite a execução da simulação diretamente no computador do desenvolvedor, dispensando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

## 4. Metodologia

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

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

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




**(a) Abordagem atual.**

**(b) Verificação de comportamento com ambiente endógeno.**

**(c) Abordagem proposta**

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

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

### 4.1. Implementação

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

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

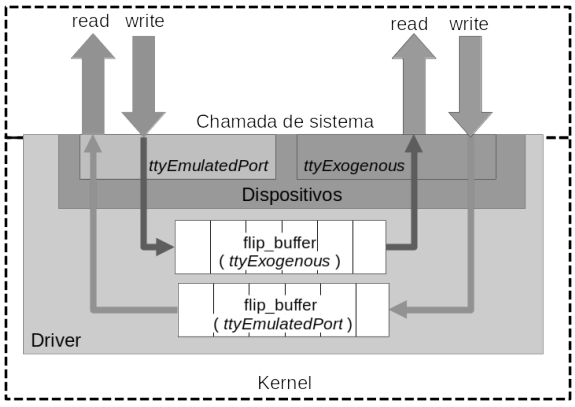



**Figura 4. Implementação do canal serial emulado**

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

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

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

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

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

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

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

## 5. Estudo de caso

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

**Actions**

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

**Perceptions**

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

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

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

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

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

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

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

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

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


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

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

**Figura 6. Uso da comunicação serial emulada.**

## 6. Discussão

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

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

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

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

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

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

## Referências

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

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

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

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

Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A., and Santi, A. (2013). Multi-agent oriented programming with jacamo. *Science of Computer Programming*, 78(6):747–761. Special section: The Programming Languages track at the 26th ACM Symposium on Applied Computing (SAC 2011) Special section on Agent-oriented Design Methods and Programming Techniques for Distributed Computing in Dynamic and Complex Environments. https://doi.org/10.1016/j.scico.2011.10.004.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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